Thesis for The Degree of Doctor of Philosophy

Systematic Design Methodologies for Multi-Engine Deep Learning Accelerators

Fareed Mohammad Qararyah

Division of Computer Engineering
Department of Computer Science & Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden, 2026

Systematic Design Methodologies for Multi-Engine Deep Learning Accelerators

Fareed Mohammad Qararyah

Supervisor:
Prof. Pedro Petersen Moura Trancoso, Chalmers University of Technology and Gothenburg University

Co-Supervisor:
Dr. Mohammad Ali Maleki, Chalmers University of Technology and Gothenburg University
Dr. Muhammad Waqar Azhar, Chalmers University of Technology and Gothenburg University [1]

Examiner:
Prof. Ioannis Sourdis, Chalmers University of Technology and Gothenburg University

Opponent:
Prof. Fabrizio Ferrandi, Politecnico di Milano, Italy

Grading Committee:
Prof. Christos Bouganis, Imperial College London, England
Assoc. Prof. Ivy Peng, KTH, Sweden
Assoc. Prof. Pedro Tomás, IST, University of Lisbon, Portugal

Deputy Committee:
Prof. Miquel Pericàs, Chalmers University of Technology and Gothenburg University

ISBN 978-91-8103-471-4
Doktorsavhandlingar vid Chalmers tekniska högskola, Ny serie nr 5928.
ISSN 0346-718X

Department of Computer Science & Engineering
Division of Computer Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden

[1] Affiliation at the time of supervision.

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2026.

Abstract

Domain-Specific Accelerators (DSAs) have become a key driver of performance and efficiency improvements in the post-Moore's Law era. These improvements stem from specializing the hardware for domain workloads and exploiting the workloads' inherent parallelism. In the domain of Deep Learning (DL), workloads (models) comprise multiple operations, known as layers, that exhibit parallelism opportunities and diverse computational characteristics. Consequently, DSAs with multiple computational units (engines) provide a natural architectural paradigm to fully exploit the specialization and parallelism potential inherent in such multi-layered models.

Multi-engine DL accelerators generally fall into two categories: model-specific and flexible. Model-specific accelerators are co-designed to efficiently execute one or a few closely related models. Flexible accelerators, by contrast, are designed to support a broad range of DL workloads. Designing and implementing accelerators in either category that fully exploit specialization and parallelism, and thus optimize performance and efficiency, requires systematic exploration based on quantitative evaluation of design alternatives. Existing multi-engine DL accelerator design approaches range from intuition-driven to exploration-based methodologies. However, even the latter typically leave key architectural parameters unexplored by fixing them a priori based on expert knowledge and intuition. In many cases, the fixed parameters are more consequential for accelerator specialization and parallelism than the explored parameters. Consequently, the full potential of the multi-engine paradigm is often left unexploited.

To fully exploit the potential of multi-engine DL accelerators, this thesis presents design methodologies that increase the extent to which key design choices are based on exploration and quantitative evaluation of design alternatives. The **first contribution** of this thesis is the Fixed Budget Hybrid CNN Accelerator (FiBHA). FiBHA proposes a hybrid, model-specific, multi-engine architecture and an accompanying design methodology. FiBHA targets a specific class of DL models and relies primarily on empirical analysis to exploit opportunities for specialization and parallelism. To expand the scope and co-design accelerators for a broader class of models, the work moved to a more analytical approach. The **second contribution** of this thesis comprises MCCM, a fast analytical cost model for evaluating model-specific multi-engine accelerators, and MCExplorer, a design space exploration framework built upon it. Together, they enable orders-of-magnitude faster evaluation and systematic exploration of model-specific multi-engine accelerator designs. Unlike existing approaches that rely on predefined design choices, MCExplorer quantitatively evaluates alternative architectural configurations across a broader design space. To further expand the scope, the work extends to flexible, in addition to model-specific, multi-engine accelerators. The **third contribution** of the thesis is a design methodology for flexible multi-engine DL accelerators, termed MEDEM. To support a wide range of diverse DL workloads, a flexible multi-engine accelerator must have an engine combination with complementary capabilities to ensure that different layers across these diverse workloads are processed efficiently. Existing work builds flexible multi-engine accelerators by combining expert-selected, independently optimized engines. However, inde-

pendently optimized engines may perform best on largely overlapping subsets of workloads, and thus their combination does not necessarily improve overall workload coverage. MEDEM presents an alternative design methodology where the engines are co-designed, then curated to find a combination that maximizes the coverage of diverse workloads.

Using a systematic approach based on modeling and quantitative evaluation of a wider space of design alternatives, the proposed methodologies identify accelerator architectures that better exploit the specialization and parallelism inherent in DL workloads. This applies to both model-specific accelerators, as FiBHA, MCCM, and MCExplorer demonstrate, and to flexible ones, as MEDEM shows. As a result, these methodologies identify architectures that consistently outperform the state-of-the-art, achieving considerable improvements in latency, throughput, energy, and energy-delay product (EDP).

Keywords:

Design Methodology, Accelerators, Co-design, Multi-engine Accelerators, Deep Neural Networks (DNNs), Deep Learning (DL), FPGA

Acknowledgment

I would like to first thank my supervisor, Prof. Pedro Petersen Moura Trancoso, for giving me the opportunity to pursue my studies and providing continuous guidance and support, and for having an open mind and being understanding whenever there is a difference in opinions. I would also like to thank my co-supervisors, Mohammad Ali Maleki and Muhammad Waqar Azhar, for their guidance and advice, feedback, assistance, and patience.

I would also like to thank my examiner, Prof. Ioannis Sourdis, for his support. I am also thankful for my friends and colleagues, Mateo, Stavroula, Alessio, Xu, Luigi, André, and others, who have been helpful and created a friendly environment.

Finally, I would like to give special thanks to my family for their unwavering support and continuous encouragement.

This work would not have been possible without the research grants from the VEDLIoT project, which received funding from the European Union's Horizon 2020 research and innovation program under grant agreement 957197. This work was also partially funded by the Swedish Foundation for Strategic Research under the PRIDE project (grant number CHI19-0048) and under the SSF-FuSS QuantumStack project (grant number FUS21-0063).

List of Publications

List of Publications

This thesis is based on the following publications:

I Fareed Qararyah, Muhammad Waqar Azhar, and Pedro Trancoso "FiBHA: Fixed Budget Hybrid CNN Accelerator"
2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD) (pp. 180-190). IEEE.

II Fareed Qararyah, Muhammad Waqar Azhar, and Pedro Trancoso "An Efficient Hybrid Deep Learning Accelerator for Compact and Heterogeneous CNNs"
ACM Transactions on Architecture and Code Optimization, 21.2 (2024), 1-26.

III Fareed Qararyah, Mohammad Ali Maleki, and Pedro Trancoso "An Analytical Cost Model for Fast Evaluation of Multiple Compute-Engine CNN Accelerators"
2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) (pp. 1-13). IEEE.

IV Fareed Qararyah, Mohammad Ali Maleki, and Pedro Trancoso "MCExplorer: Exploring the Design Space of Multiple Compute-Engine Deep Learning Accelerators"
ACM Transactions on Architecture and Code Optimization, 22.4 (2025), 1-27.

V Fareed Qararyah, Mohammad Ali Maleki, and Pedro Trancoso "MEDEM: Multi-Engine DL Accelerator Design Methodology"
Submitted for publication.

Other Publications

The following work was also published during my PhD studies. However, it is not included in this thesis as it is not central to the thesis topic.

I Fareed Qararyah, Muhammad Waqar Azhar, Mohammad Ali Maleki, and Pedro Trancoso "Fusing Depthwise and Pointwise Convolutions for Efficient Inference on GPUs"
Workshop Proceedings of the 53rd International Conference on Parallel Processing, pp. 58-67. 2024.

Contents

Chapter 1

Introduction

Domain-Specific Accelerators (DSAs) leverage the massive inherent parallelism of Deep Learning (DL) workloads through highly parallel architectures composed of many processing elements. These processing elements can be organized as a single unit (an engine), yielding a monolithic accelerator, or as multiple engines, yielding a multi-engine accelerator. As improvements in monolithic DL accelerators have plateaued [172], multi-engine accelerators offer new optimization potential by exploiting greater opportunities for specialization and parallelism. As a result, multi-engine accelerators are becoming increasingly dominant in both research and industry [13, 25, 39, 43, 45, 70, 86, 131, 136, 145, 158, 172, 201].

Multi-engine DL accelerators are composed of independently controllable computing engines, where each engine mainly consists of an array of Processing Elements (PEs), as well as local and global buffers. The engines typically exchange data through on-chip buffers connected by a high-bandwidth interconnect fabric. Multi-engine accelerators can be model-specific or flexible. Model-specific ones are typically deployed on reconfigurable hardware and are optimized for single or a few very similar workloads (models) [9,49,168,197]. In contrast, flexible accelerators are optimized for diverse DL workloads [13,70,86,145].

Multi-engine DL accelerators offer several benefits over monolithic ones. First, they enable greater specialization, which is the primary source of efficiency in DSAs [28,94]. DL models have layers with diverse computational characteristics, and having multiple engines enables engines to be dedicated to such varying characteristics, resulting in better performance and efficiency [10,14,29,86,197]. Second, multi-engine accelerators exploit the coarse-grained model-level parallelism [14, 39, 43, 86, 201]. DL models are composed of several operators, known as layers. Having multiple engines unlocks parallelism across layers by enabling their simultaneous execution. Third, multi-engine accelerators' modular architecture offers a scalable and flexible solution for processing large DL models [45, 140, 145]. Fourth, multi-engine accelerators enable more optimization opportunities for the increasingly common case of multi-model workloads [13, 45, 120, 169, 172].

To fully realize these benefits, multi-engine accelerators must be systematically designed with careful evaluation of trade-offs. But the exponentially large number of design possibilities, combined with the time required to evaluate each design point, makes such systematic design challenging. As a result,

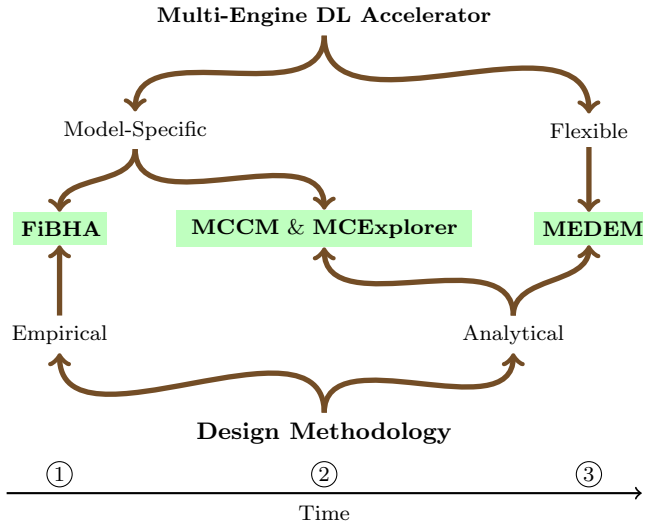


Figure 1.1: Taxonomy and chronology of the thesis contributions. FiBHA is first introduced in Section 1.3.1, MCCM in Section 1.3.2, MCExplorer in Section 1.3.3.3, and MEDEM in Section 1.3.4. The classification of a work as empirical or analytical is based on its predominant approach, particularly the one driving key design decisions. Since DL accelerator design encompasses multiple aspects, the works exhibit elements of both approaches.

existing work on designing both model-specific [2, 9, 49, 168, 197, 199] and flexible [13, 29, 70, 145, 185] multi-engine accelerators presupposes key design choices or makes them based on experts' knowledge and intuition.

However, the quantitative approach to computer architecture design emphasizes that design choices must be made based on measurable metrics and quantitative comparison of design alternatives [54]. The literature on monolithic DL accelerator design provides abundant evidence that expert decisions are consistently outperformed by *exploration-based design methodologies* where design alternatives are quantitatively evaluated [31, 56, 77, 111, 122, 138, 195]. Moreover, although expert decisions can yield optimized architectures in specific cases, it does not offer a reliable general approach. DL models are constantly evolving, modifying layer compositions or introducing novel layers with specialized computational characteristics. Consequently, expert decisions and intuitions that are effective for certain models or layers may not generalize to others. Moreover, DL models are deployed across the compute continuum with widely varying resource budgets. The optimal accelerator architecture under a given resource budget is not necessarily optimal under another, even for a single DL model.

This thesis presents design flows and methodologies that aim to identify multi-engine DL accelerator architectures that better exploit the specialization and parallelism in DL workloads. Consequently, it contributes to the design of high-performance, efficient accelerators, both model-specific and flexible ones. Figure 1.1 contextualizes the thesis contributions. The work in this thesis follows an evolving trajectory from an earlier approach relying mainly on empirical analysis towards design methodologies more grounded in analytical evaluation

of design alternatives through design space exploration (DSE). The proposed design tools and methodologies identify multi-engine accelerator architectures that exploit the specialization and parallelism opportunities in DL models, enabling better performance and efficiency. The rest of this introductory Chapter is organized as follows: Section 1.1 provides high-level background, the problem statement is presented in Section 1.2, followed by a discussion of the objectives and contributions of this thesis in Section 1.3.

1.1 Background

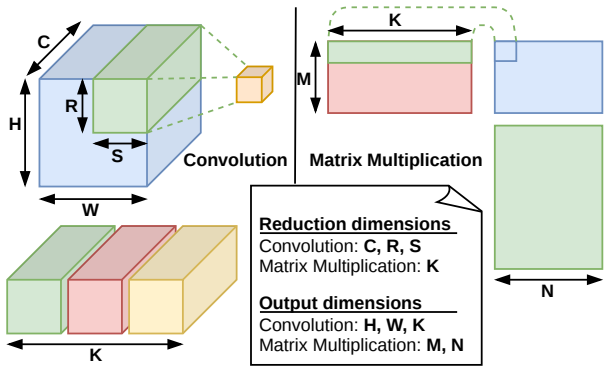


Figure 1.2: The 6 dimensions of convolution, 3 dimensions of matrix multiplication; reduction and output dimensions.

1.1.1 Deep Learning Models and Layers

Deep Learning (DL) models, such as Convolutional Neural Networks (CNNs) [91] and Transformers [165], consist of a sequence of layers. Each layer performs a specific tensor operation to transform input data into alternative feature representations. Two core operations of DL models are convolution and matrix multiplication, which are characterized by their multi-dimensional iteration spaces [87, 122]. A layer combines *input* activations with learned *weights* (known as *filters* in convolutional layers) to produce *output* activations. A standard convolutional layer operates within a 6-dimensional space (K, C, H, W, R, S) shown in Figure 1.2, while a matrix multiplication operation has a 3-dimensional space (M, N, K), assuming no batching. Modern DL models, despite having tens to thousands of layers, are often composed of only a few unique layers that repeat throughout the model. Where a unique layer is one that has a specific set of values for all dimensions.

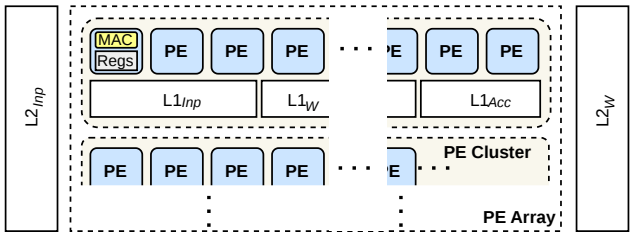


Figure 1.3: Engine abstraction: a high-level overview.

1.1.2 Computing Engines

We use the terms **engine** and **compute engine (CE)** interchangeably throughout the thesis[1]. An engine mainly consists of a spatial array of Processing Elements (PEs) and a multilevel memory hierarchy, including private register files (RFs), local and global buffers. Figure 1.3 shows an example of an engine. In this example, each PE contains a MAC unit and a small set of local registers; the PEs can be grouped into PE clusters and have local buffers (L1). The engine may have global buffers accessible by all its PEs (L2).

1.1.3 Key Bottlenecks and Sources of Inefficiencies in DL Accelerators

1.1.3.1 Processing Element Underutilization

Processing element (PE) underutilization can significantly *degrade both the performance and energy efficiency* of DL accelerators [10, 106, 195]. Since PEs are responsible for executing the core arithmetic operations of a DL model, idle or underutilized PEs directly reduce effective computational throughput and increase end-to-end execution time. In practice, many DL models, especially edge or compact ones, operate far below the accelerator's peak throughput because large portions of the PE array remain inactive during execution [10]. This issue is particularly severe for models with irregular layer dimensions and varying arithmetic intensities.

Unlike the impact of PE underutilization on execution time, which cannot be mitigated, the energy inefficiency it causes can be reduced through techniques such as *clock gating*[2] and *dynamic voltage/frequency scaling (DVFS)*. However, these techniques have their own limitations. For example, clock-gating circuitry incurs additional area and power overhead, preventing its application at very fine granularity [82]. Consequently, inactive hardware blocks may still consume some dynamic power. Moreover, data often continues to traverse unused regions of the chip, leading to significant interconnect energy consumption even when portions of the hardware are disabled. Finally, clock gating primarily targets dynamic power reduction, while leakage power remains in inactive regions. Consequently, *maintaining high PE utilization is critical for maximizing throughput, reducing*

[1]The terms **multiple-CE** and **multi-engine** are also used interchangeably throughout the thesis.

[2]Power gating is not practical at PE granularity due to the area overhead of power-management circuitry, wake-up energy, and latency costs.

inference latency, and achieving high performance-per-watt in modern DL accelerators.

1.1.3.2 Large On-Chip Buffers

Large on-chip buffers are used in DL accelerators to reduce expensive off-chip memory accesses and improve data reuse. On the one hand, unnecessarily large buffers can negatively impact both performance and energy efficiency. The buffer area, leakage power, and access latency and energy grow with the buffer size. This means wasting silicon area and static power, and making memory accesses more energy-intensive and potentially limiting operating frequency. On-chip buffers can be a dominant contributor to overall power and energy consumption, reaching up to 48% [10]. This is observed in both ASIC and FPGA-based accelerators [10,125,191]. On the other hand, having small buffers that do not exploit the data reuse opportunities in DL layers increases data movement, leading to a loss in time and energy. Consequently, *buffer sizes must be co-designed through systematic exploration to maintain good performance and efficiency.*

1.1.3.3 Off-Chip Accesses

Off-chip memory accesses are a primary performance and energy bottleneck in modern DL accelerators due to relatively limited bandwidth, high latency, and higher per-access energy compared to on-chip data movement and computation [2, 18, 62, 64, 67, 195]. Consequently, *reducing off-chip accesses is essential for achieving high-performance and energy-efficient DL.*

To reduce off-chip memory accesses, DL accelerators employ various optimizations. These optimizations can be broadly classified into two orthogonal categories. First, optimizations that reduce the volume of data transferred. These include quantization, compression, sparsity encoding, and pruning. These methods reduce the bandwidth and storage requirements of data movements between on-chip and off-chip memory. Second, optimizations that reduce off-chip transfers by maximizing on-chip data reuse and avoiding redundant data movement. Dataflow reorganization techniques exploit temporal and spatial locality to retain data on-chip as long as possible and maximize its use per transfer. Layer fusion further reduces off-chip accesses by eliminating intermediate data write-backs and reloads, directly forwarding data between consecutive layers [2, 67]. These two categories of optimizations are largely orthogonal and can work synergistically. For example, fused layers may operate on quantized or compressed tensors, simultaneously reducing both transfer frequency and transfer volume.

1.1.4 The Need for Systematic DL Accelerator Design Methodologies

This section highlights the importance of making DL accelerator design decisions based on the quantitative evaluation of design alternatives rather than on expert knowledge and intuition. It presents an example of the cost of a commonly followed approach in which accelerator microarchitectures are assumed first and optimized second. Then it shows that the interactions between performance

and efficiency bottlenecks of a DL accelerator make finding optimized ones a task too complex to be overcome by an architect's intuition.

1.1.4.1 The Cost of Expert Knowledge-Based Accelerator Design

Achieving efficient DL is a challenging task that requires careful consideration of a wide range of software and hardware parameters. One way to simplify this task is to fix (predefine or presuppose) some of these parameters based on expert knowledge and intuition. An example of this is the use of 2D systolic arrays (SAs) in DL accelerators. Fixing such a parameter, *i.e.* assuming an SA-based microarchitecture, is an intuitive design choice. SAs are highly efficient for dense matrix multiplication, offering excellent data reuse, simple PEs, and regular communication patterns. This makes them particularly well-suited for many DL models dominated by matrix multiplications. For example, Google's Tensor Processing Units (TPUs) use SAs as their core engines [70, 71].

As a result of their wide adoption, a prevalent approach has been to transform diverse DL operations, most notably convolutions, into matrix multiplication form. However, this transformation is not for free. Converting convolution operations into matrix multiplication typically requires explicit data rearrangement (e.g., via the im2col transformation), followed by matrix multiplication and subsequent reshaping of results. Such transformations introduce additional overhead in both memory usage and data movement. In particular, im2col-based approaches expand input tensors into larger intermediate matrices with redundancy, increasing memory footprint, bandwidth demand, and energy consumption. Previous work has shown that this lowering process can incur significant performance and memory overheads [103, 202]. Consequently, while SA-based designs excel at accelerating dense matrix multiplication, an over-reliance on such transformations can lead to inefficiencies when handling the broader diversity of DL workloads. In fact, this issue reflects a broader principle of *reduction* [94]. Whenever a problem A is transformed into another B so that it can be solved using existing efficient hardware or software, inefficiencies arise both from the cost of performing the transformation itself and from using a more general solution that is not tailored to the original problem.

To optimize the performance and efficiency when designing DL accelerators, one should follow a more exploration-based quantitative design approach. That is, *make more and more key design choices based on modeling and evaluating design alternatives rather than expert knowledge and intuition.*

1.1.4.2 Interdependence Among DL Accelerator Bottlenecks

Intricate interactions between PE utilization, on-chip buffer sizes, and off-chip accesses govern the performance and efficiency of DL accelerators. This section presents some examples of the implications of such interactions.

First, *no single optimization or execution policy is universally effective.* For example, layer-fusion and inter-layer pipelining are commonly used in the literature to consume the data while closer to the processing elements and avoid costly data movements to both large on-chip buffers and off-chip memory. However, always applying such optimizations *without modeling the PE array shape, the on-chip buffer sizes, and the off-chip bandwidth and access cost*

can degrade the performance and efficiency [15, 16, 67]. For example, fusion may increase intermediate tensor sizes and create excessive on-chip memory or register pressure, and may even increase off-chip data movement. Fusion can also reduce parallelism and reuse opportunities, and limit computation scheduling flexibility.

Second, *optimizing for one of the performance and efficiency bottlenecks without considering the implications on the others can lead to overheads that outweigh the achieved benefits.* A simple example of that is focusing only on optimizing PE utilization. Optimizing utilization across layers of diverse computational characteristics can be achieved using relatively large and multi-banked on-chip buffers. Such buffers can keep enough data on-chip, preventing frequent access latencies from being a bottleneck, and can keep up with the PEs' data demands every cycle. However, such buffers require a large area and can dominate the overall power consumption. Another common example in the literature is prioritizing the reduction of off-chip accesses as the primary optimization objective [2], motivated by the observation that off-chip accesses are the most energy-expensive operations in DL accelerators. However, reducing off-chip accesses does not necessarily lead to minimal energy usage, as it can come at the cost of exacerbating other bottlenecks. For example, TimeLoop reports up to an 11× difference in energy efficiency among 6582 designs that all achieve the minimum number of DRAM accesses [122].

The interactions between the bottlenecks make accelerator design a multidimensional optimization problem, where improving one aspect of the architecture may inadvertently degrade others. This makes it unlikely to achieve optimal performance and efficiency by relying solely on conventional wisdom and broadly accepted optimization practices. Instead, achieving high performance, energy, and area efficiency requires *methodologies for modeling, exploring, and quantitatively evaluating various design choices.*

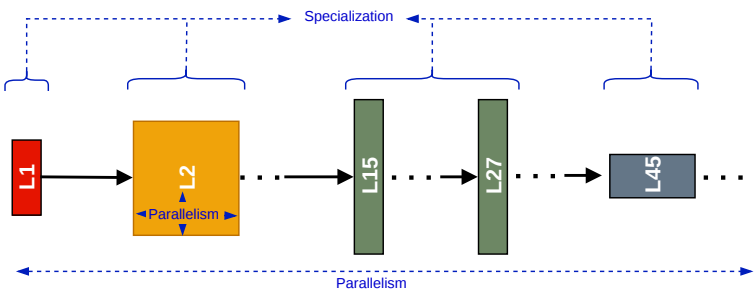


Figure 1.4: Abstract representation of specialization and parallelism opportunities in DL models. Colors and shapes indicate different computational characteristics across layers.

1.1.5 Rationale for Multi-Engine DL Accelerators

Exploiting Specialization and Parallelism in DL Workloads

DSAs gain in performance and efficiency by *specializing* the hardware in the workloads of the domain and exploiting the inherent *parallelism* of the

workloads [28, 94]. Specialization allows hardware simplification by removing unnecessary overheads, given the target domain's workload characteristics. It also allows tailoring the memory system given the workload's memory access patterns, maximizing data locality and reducing data movement costs. Parallelism improves the execution time by allowing more work to be performed simultaneously. Parallelism also amortizes per-operation overheads. For example, by reducing the effective cost of memory accesses through spatial and temporal data reuse. The more opportunities for specialization and parallelism a DSA exploits, the greater the potential improvements in performance and efficiency. As Figure 1.4 shows, layers of DL models are heterogeneous (*i.e.* have diverse computational characteristics), they differ in their arithmetic intensities, memory access and reuse patterns, input and output sizes and shapes. This heterogeneity offers potential for specialization across layers. Moreover, in addition to parallelism within a layer, there is parallelism across layers. Designing a multi-engine architecture is a natural way to exploit the inter-layer specialization and parallelism.

Scalability, Flexibility, and Processing Multi-Model Workloads

The inability of monolithic DL accelerators to fully exploit the optimization opportunities in DL models results in a gap between the peak and the achievable performance and efficiency of these accelerators. This gap widens when scaling these accelerators by adding more PEs [141, 172]. Some scalability bottlenecks can be mitigated, but at the cost of additional overhead [10, 172]. For example, designing a flexible PE array that can be configured at runtime to suit the characteristics of different layers may mitigate the inefficiencies caused by PE underutilization [190]. However, runtime reconfiguration brings overheads in terms of area and energy [26, 198]. Moreover, multi-model workloads are becoming more widespread. The increase in heterogeneity within a multi-model workload requires more flexibility. In the case of a reconfigurable monolithic engine, this incurs more frequent reconfiguration overheads.

Multi-engine architectures offer a better balance between scalability and the flexibility needed to maintain resource utilization, leading to better performance and efficiency. Even in cases of underutilization, unused engines can be completely powered down, reducing both dynamic and leakage power more effectively than partially gating resources within a single engine. Moreover, multi-engine accelerators offer more optimization opportunities for the multi-model workloads [13, 45, 120, 169, 172]. First, if the engines in a multi-engine accelerator are co-designed considering the characteristics of representative models, these engines can exploit specialization across different models in a workload. Secondly, multiple engines provide the flexibility to run different models simultaneously.

1.2 Problem Context and Statement

1.2.1 Context

The literature is rich with sophisticated and highly optimized multi-engine DL accelerators. The design of such accelerators involves varying degrees of model-

ing and design space exploration, ranging from almost none, where decisions are justified only through post-design comparisons with prior work, to more systematic exploration-based methodologies. However, even in exploration-based works, key design choices, such as those discussed in Sections 1.3.3.2 and 1.3.4.2, are still often guided by expert knowledge or intuition, while exploration is primarily applied to secondary design choices given the presupposed key ones.

To highlight the limitations of these approaches and motivate the evolution of the multi-engine accelerator design flows and methodologies proposed in this thesis, we base our discussion on two premises:

- **Premise 1:** The ultimate goal of designing domain-specific accelerators is to continue *improving performance and efficiency* in the post–Moore's law era [28, 94].
- **Premise 2:** The *quantitative* approach, a fundamental paradigm in modern computer architecture, emphasizes using measurable *metrics* and quantitative comparison of *alternatives* to guide design choices [54].

From these two premises, it follows that designing domain-specific accelerators, including multi-engine DL accelerators, should be based on modeling and evaluating the *performance and efficiency* of design *alternatives*. Adopting a quantitative approach is especially essential in the design of DL accelerators because the vast design space and the intricate interactions among its components make it unlikely that expert knowledge and intuition alone can reliably identify optimized solutions. Even when expert knowledge and intuition identify highly optimized designs in specific scenarios, they do not offer reliable and reusable methodologies. Moreover, the continuous evolution of DL workloads and their widely varying deployment scenarios across the compute continuum necessitate reliable design methodologies.

We observe that existing work on designing both model-specific and flexible multi-engine DL accelerators either presupposes key design choices or bases them on expert knowledge and intuition. This thesis aims to present *more systematic* multi-engine DL accelerator design flows and methodologies. That is, increase the extent to which key design choices are based on modeling and quantitative evaluation of design alternatives.

1.2.2 Problem Statement

The main problem that this thesis tries to tackle can be stated as follows:

In the design of multi-engine deep learning accelerators, making key design choices based on expert knowledge and intuition instead of systematic quantitative evaluation of design alternatives produces architectures that leave substantial performance and efficiency potential untapped.

Along the path toward addressing this problem, several research questions naturally emerge, guiding the development of the thesis contributions.

Research Question 1 (RQ1): *How can a model-specific DL accelerator*[3] *be designed to optimize performance-resource trade-offs across different resource budgets?*
Research Question 2 (RQ2): *How to systematically identify model-specific multi-engine DL accelerator architectures that fully exploit the potential of the multi-engine design paradigm?*

Research Question 3 (RQ3): *How to systematically identify flexible multi-engine DL accelerator architectures that maximize coverage across a diverse DL workload landscape, thereby minimizing aggregate execution costs?*

The next section (Section 1.3) lists the contributions of this thesis, discussing how each of them addresses one of these questions.

1.3 Thesis Objectives and Contributions

This section presents summaries of works constituting this thesis and how they contribute to addressing the research questions posed in the previous section. Section 1.3.1 presents a hybrid, model-specific architecture together with a design flow to derive optimized accelerator instances under a fixed resource budget. This work addresses **RQ1** (model-specific design under fixed budgets). The work relies mainly on empirical analysis to identify opportunities for specialization and parallelism within a given DL model and to design an accelerator that optimizes performance per resource budget. Sections 1.3.2 and 1.3.3 present a cost model and a DSE framework that address **RQ2** (systematic design of model-specific multi-engine accelerators). The cost model offers fast evaluation of model-specific multi-engine DL accelerators, and the exploration framework leverages the fast evaluation to conduct a DSE, quantitatively evaluating a wide range of architectural alternatives. This broad exploration better exploits the potential of these accelerators. Section 1.3.4 addresses **RQ3** (design of flexible multi-engine accelerators for diverse workloads). It presents a design methodology of flexible multi-engine DL accelerators that co-designs and identifies engine combinations with complementary capabilities that maximize workload coverage across diverse DL models, thereby minimizing aggregate execution costs.

1.3.1 Designing Model-Specific Multi-Engine DL Accelerators for Compact and Heterogeneous CNNs

The first contribution of the thesis is the Fixed budget hybrid CNN accelerator (FiBHA), a model-specific multi-engine DL accelerator with a hybrid architecture optimized for compact and heterogeneous Convolutional Neural Networks (CNNs), and a design flow to derive instances of that architecture under a resource budget. This work addresses **RQ1** (Section 1.2.2) in the context of compact CNNs. FiBHA's design methodology is to make key architectural choices based on an empirical analysis of model layer characteristics, to enable optimization of a target objective under a fixed resource budget.

[3]At this point, being multi-engine was not decided yet.

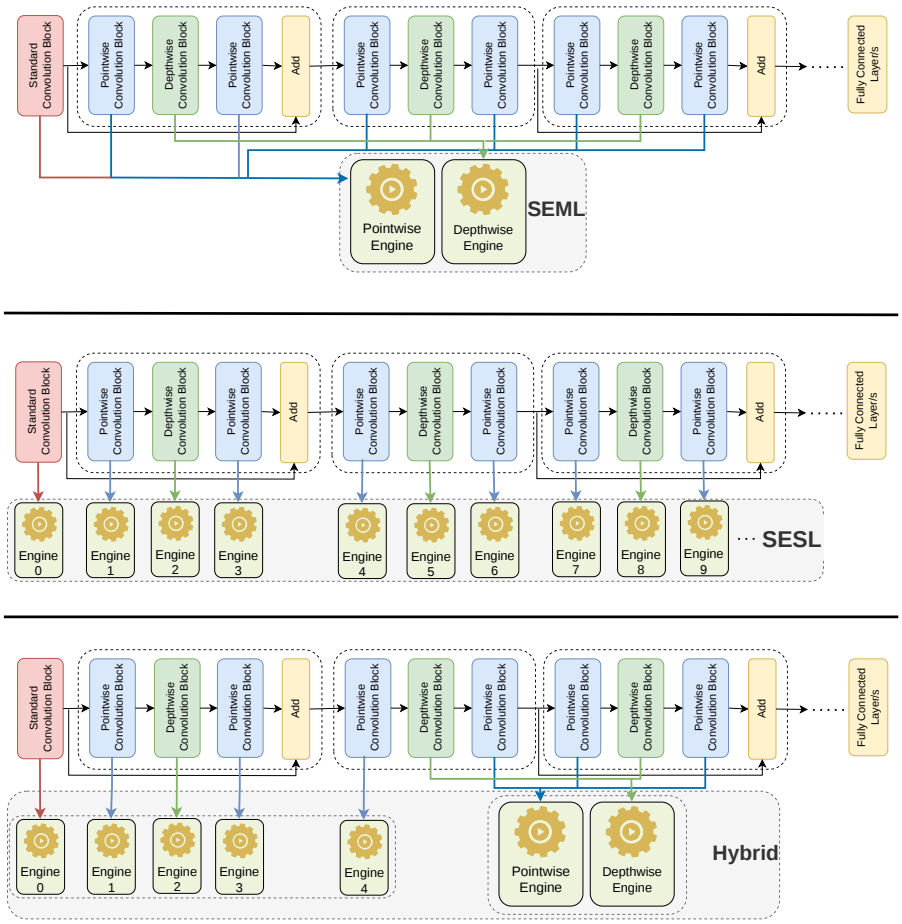


Figure 1.5: High-level overview of the Single-Engine Multiple-Layer (SEML), Single-Engine Single-Layer (SESL), and Hybrid accelerators. The figure focuses on the mapping from the convolutional layers of a CNN to the compute engines of the accelerators; the internals of the engines, their connectivity, and the memory system are omitted for simplicity.

1.3.1.1 Background

Compact CNNs, also known as heterogeneous, resource-efficient, or edge CNNs, balance the accuracy-efficiency trade-off [10, 184]. They either improve accuracy without increasing model weights and computations [21] or reduce the model weights and computations considerably at the cost of a negligible loss in accuracy [11, 57, 142, 160, 196]. The core layers in a CNN, the convolutional layers, have various computational characteristics [10, 184]. We designate these variations as intra-layer-type heterogeneity, as they exist among layers of the same type. In addition, compact CNNs are composed of modules containing different types of convolutions, including depthwise separable convolution (DSC) and the inverted bottlenecks [21, 179, 181]. This introduces another form of heterogeneity, which we refer to as inter-layer-type heterogeneity.

1.3.1.2 Existing Model-Specific CNN Accelerators

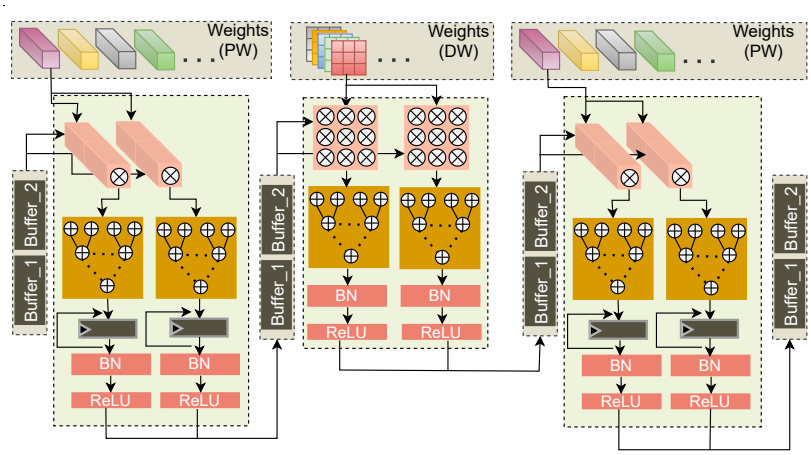
We categorize existing model-specific CNN accelerators, depending on the mapping from CNN layers to engines, into four categories:

- **Monolithic accelerators:** accelerators in which all the convolutional layers are executed using the same engine [5, 18].
- **Single-Engine Multiple-Layer (SEML):** accelerators in which all the convolutional layers of a certain type are executed using the same engine [5, 98, 100, 152, 176, 179, 186].
- **Single-Engine Single-Layer (SESL):** accelerators in which each layer is mapped to a distinct engine. In these accelerators, the chain of engines is pipelined to work simultaneously [9, 97, 163, 166].
- **Multiple-Engine Multiple-Layer (MEML):** accelerators in which a single layer is processed by multiple engines, where each engine processes a tile of that single layer, and these multiple engines are reused across multiple layers [42, 107].

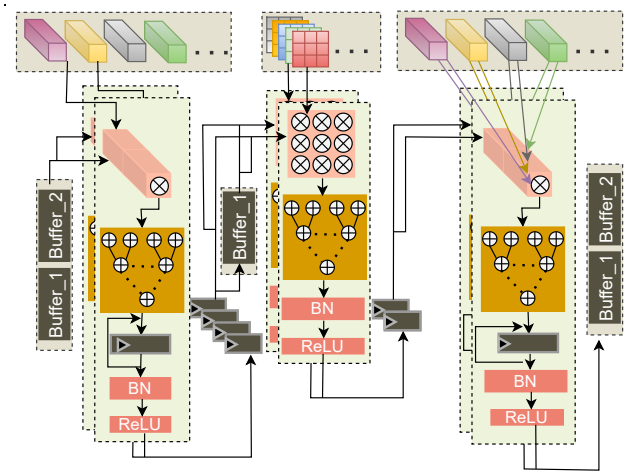
We mainly focus on the second and third categories, i.e., SEML and SESL. This is because MEML is more tailored for resource-demanding scenarios like having large DNNs or the training phase. Figure 1.5 shows examples of both SEML and SESL mapping approaches.

1.3.1.3 Fixed budget hybrid CNN accelerator (FiBHA)

Presupposing pure SEML or pure SESL architecture is a restrictive design choice. Both SEML and SESL architectures have their own bottlenecks. While SEML accelerators do not capture intra-layer-type heterogeneity, pure SESL accelerators are resource-demanding and do not scale well. Based on an empirical analysis of the characteristics of compact CNN layers, this work proposes FiBHA, a hybrid architecture that combines both SESL and SEML architectures as shown in Figure 1.5. The SESL part of FiBHA computes the initial layers of a CNN, and the SEML part computes the rest. This hybrid



(a)



(b)

Figure 1.6: **(a)**Traditional inter-layer pipelining-based implementation of an Inverted Residual Bottleneck module. **(b)** FIRB-based implementation of an Inverted Residual Bottleneck module.

architecture allows capturing more heterogeneity than SEML accelerators can, while being more resource-efficient than pure SESL. We also propose a heuristic "Split a CNN" (*SplitCNN*) that derives a FiBHA instance under a fixed resource budget and splits the CNN layers and the resources between that instance's SESL and SEML parts. Finally, we propose *Fused Inverted Residual Bottlenecks (FIRB)* module to implement the SESL part of FiBHA. As shown in Figure 1.6, FIRB applies fine-grained pipelining that results in a considerable reduction in the double-buffering required by SESL, which scales with the pipeline length. The contributions of this work are as follows:

- It proposes Fixed Budget Hybrid CNN Accelerator (FiBHA), a novel hybrid architecture composed of a SESL part and an SEML part, each computing a part of a CNN model.
- It proposes SplitCNN, a heuristic that derives a FiBHA instance given a fixed resource budget and splits a CNN and the computational resources between its SESL and SEML parts to optimize performance per resource.
- It proposes Fused Inverted Residual Bottlenecks (FIRB), a fine-grained and memory-light pipeline building block, and uses it to implement the SESL part of FiBHA efficiently.
- It implements and evaluates FiBHA, demonstrating its performance and resource and energy efficiency using representative compact CNNs and multiple evaluation boards representing various resource budgets.

We evaluated FiBHA using different FPGAs with varying resource budgets representing different points in the compute continuum, and state-of-the-art compact CNNs [11, 57, 142, 159]. FiBH achieves 2.5× and 4× of the throughput achieved by state-of-the-art model-specific accelerators, under the same hardware budget. It also allows better scaling in cases of memory-constrained hardware compared to SEML. FIRB modules reduce the required memory by up to 54%, and energy requirements by up to 35% compared to traditional inter-layer pipelining.

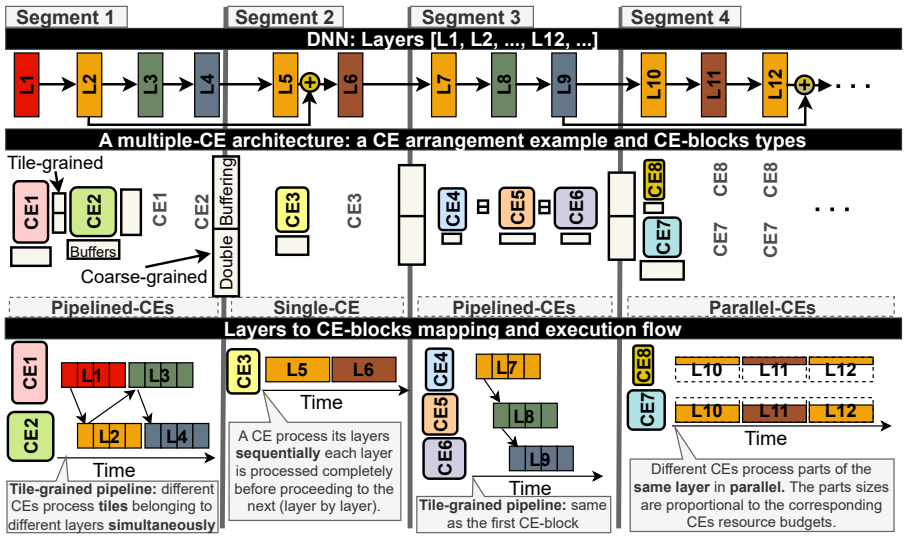


Figure 1.7: DNN to multi-engine architecture mapping example, and multi-engine accelerator building blocks. The CEs and the buffers have different sizes, which are proportional to the segment layers' compute and memory requirements, respectively.

1.3.2 Fast Modeling and Evaluation of Model-Specific Multi-Engine CNN Accelerators

The second contribution of this thesis addresses the second research question (**RQ2**). To enable the systematic design of model-specific multi-engine accelerators, a fast evaluation methodology is essential. To achieve such fast evaluation, this work proposes the multi-engine accelerator analytical cost model (MCCM), a fast analytical cost model for evaluating model-specific multi-engine DL accelerators. MCCM estimates the performance, efficiency, and resource requirements of multi-engine accelerators several orders of magnitude faster than conventional evaluation approaches. This speedup is achieved through high-level modeling that captures only the dominant performance and resource bottlenecks.

1.3.2.1 Background

Existing model-specific multi-engine accelerators are usually deployed on FPGAs. These accelerators differ in how they arrange their engines and distribute FPGA hardware resources and DL model layers among engines. Such architectural choices define a large design space with numerous possible configurations. The lack of fast methodologies for modeling and evaluating the impact of engine arrangement and the distribution of resources and workload computations among them on the accelerator's performance and efficiency renders exploring such a large space impractical. While High-Level Synthesis (HLS) shortens the design time, synthesizing a single accelerator instance can take hours.

1.3.2.2 Representing and Modeling a Model-Specific Multi-Engine DL Accelerator

Figure 1.7 shows an example of a multi-engine accelerator and a DL model (DNN) to accelerator mapping. Two basic Compute Engine (CE) blocks, shown in the figure, are used across the existing multi-engine accelerators. These are *single-CE* blocks and *pipelined-CEs* blocks [129]. The single-CE simply consists of a single engine. It processes its assigned DNN layers in a layer-by-layer (LBL) fashion, where a layer is processed completely before moving to the next. The pipelined-CEs block comprises a set of pipelined engines, which process the layers in a layer-pipelined (LP) fashion, where multiple layers are processed simultaneously. FiBHA is an example of a multi-engine accelerator that uses these two blocks in its SESL and SEML parts. In addition to these two blocks, the *parallel-CEs* block is a block where the PEs are arranged into distinct CEs non-uniformly. These CEs process the same layer simultaneously. Parallel-CEs blocks improve scalability and enable higher performance when further scaling of a single-CE block becomes ineffective.

Any model-specific multi-engine accelerator can be constructed using combinations of the three CE blocks, where a DNN is split into *segments*, each of which is processed using one of these blocks. As DNN layers have diverse computational characteristics, the choice of blocks to be used to process certain layers has an impact on PE utilization, reuse opportunities, and resource requirements, which affects the achievable performance and efficiency. Modeling a model-specific multi-engine accelerator can be done by modeling these building blocks and the interfacing between them.

1.3.2.3 Multi-Engine Accelerator Analytical Cost Model (MCCM)

Fast modeling and evaluation of multi-engine accelerators are prerequisites for conducting a systematic design based on quantitatively evaluating key design choices. To enable such fast evaluation, we propose a multi-engine accelerator analytical cost model (MCCM) and an evaluation methodology built around it. The proposed model and methodology enable the expression of a multi-engine accelerator with any possible engine arrangement, using simple notation, and provide a fast and accurate evaluation. MCCM is fast because it models the *bottlenecks* of an architecture rather than conducting detailed modeling. The bottlenecks of DL accelerators, in general, have three root causes, **(1)** *Processing Element (PE) underutilization* [10, 106, 195], **(2)** *large on-chip buffers* [10, 125, 191], **(3)** and the time and energy-costly *off-chip access* [106, 195]. MCCM models those bottlenecks and their impact on performance and efficiency[4]. The contributions of this work are as follows:

- It proposes a multi-engine accelerator analytical cost model (MCCM). MCCM applies bottom-up modeling of the impact of engine arrangement possibilities on the performance and efficiency of multi-engine accelerators.
- It proposes an MCCM-based evaluation methodology, which enables expressing any multi-engine accelerator and evaluates its latency, through-

[4]MCCM is not a replacement for the detailed evaluation produced by synthesis tools; it functions as a high-level modeling that can help guide a DSE.

put, on-chip buffer requirements, and off-chip access orders of magnitude faster than the traditional approach.

- It presents three use cases of MCCM. First, an end-to-end evaluation of existing multi-engine accelerators considering different metrics, CNNs, and resource budgets. Second, fine-grained evaluation of multi-engine accelerators' performance bottlenecks. Third, a design space exploration of multi-engine accelerators enabled by MCCM fast evaluation.

MCCM is in the order of 100000× faster than synthesis-based evaluation and has an average accuracy of $> 90\%$. The presented MCCM use cases using three state-of-the-art multi-engine accelerators, five CNNs, and four FPGA boards show that no single multi-engine architecture is the best, given different metrics, CNN models, and resource budgets. Moreover, the evaluation shows the impact of engine arrangements on multi-engine accelerators' performance. Finally, using MCCM fast evaluation as a basis for multi-engine accelerators design space exploration enables identifying designs that outperform the state-of-the-art.

1.3.3 Model-Specific Multi-Engine DL Accelerator Design Methodology

The work discussed in this section complements the previous one (MCCM) to fully address the second research question (**RQ2**). The work proposes MCExplorer, a framework that leverages the fast modeling of MCCM to automate the design space exploration of model-specific multi-engine DL accelerators. Compared to existing design methodologies, MCExplorer explores a broader design space. Rather than relying on predetermined engine arrangements and configurations, it systematically explores various design alternatives. This enables a fuller exploitation of the potential of multi-engine DL accelerators.

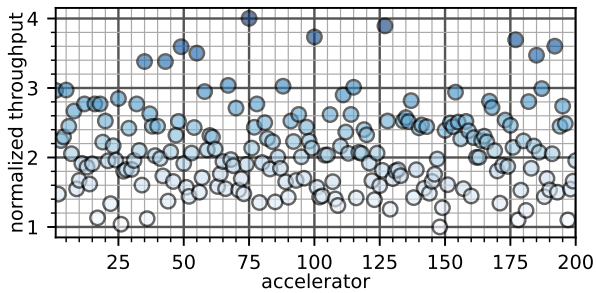
1.3.3.1 Background

Model-specific multi-engine accelerators have an exponentially large design space. Exhaustively exploring this space is impractical. For example, considering a simple architecture using only contiguous single-CE blocks (Section 1.3.2.2), there are $\binom{\#Layers\ -\ 1}{\#CEs\ -\ 1}$ possible layer-to-engine mappings. The design space of only the mappings in such a simple accelerator for ResNet152 [50], using 10 engines, is roughly 10^{14}. Existing work explores a limited portion of this space by restricting key design choices. These design choices are either predefined or based on expert intuition. Such restriction of the explored design space hinders existing work's ability to identify highly optimized architectures. Moreover, existing work optimizes for one performance metric and measures improvements in other metrics as a by-product. While some mainly optimize latency [174], others target throughput [9, 150].

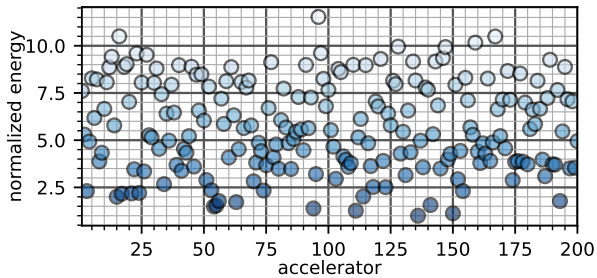
1.3.3.2 Expert-Driven Key Design Choices for Model-Specific Multi-Engine DL Accelerators

Existing multi-engine DL accelerator design methodologies differ substantially in the extent of modeling and design space exploration employed, from ap-

proaches with little or no explicit modeling or design space exploration, where design choices are validated primarily by comparison with existing solutions, to systematic frameworks that rely on extensive exploration of the design space. However, even in the latter, key design choices are still predefined or determined by expert knowledge and intuition. Two examples of such key design decisions are *inter-CE arrangement* and *intra-CE configurations*.



(a) Normalized throughput values of 200 randomly generated multi-engine accelerators.



(b) Normalized energy-per-inference values of 200 randomly generated multi-engine accelerators.

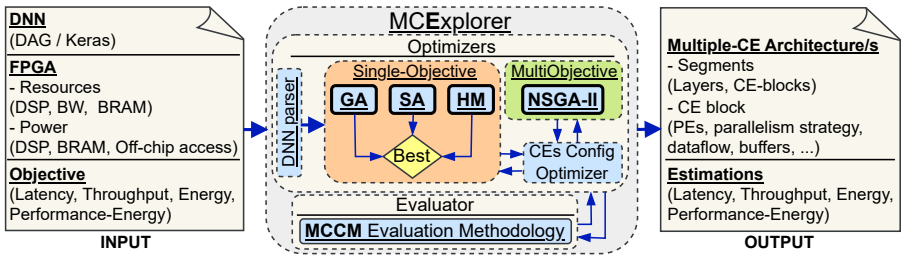
Figure 1.8: Throughput and energy of 200 randomly generated multi-engine accelerators. The values are obtained using MCCM [130].

Regarding **CE arrangements**, most existing model-specific multi-engine accelerators are designed using one type of engine block [9,149,150,168,174,197]. A less common option is hybrid, which uses two types of blocks, like FiBHA (Section 1.3.1). However, in both cases, the choice of the engine arrangement, *i.e.*, the types of engine blocks, is not based on rigorous quantitative analysis of the design alternatives. While in some cases it is based on empirical analysis and reasoning about a specific class of DL models, in other cases it is simply predefined with no explicit justification provided[5]. However, the chosen engine arrangement has a considerable impact on the multi-engine accelerator performance and efficiency. Figure 1.8 depicts some aspects of such an impact.

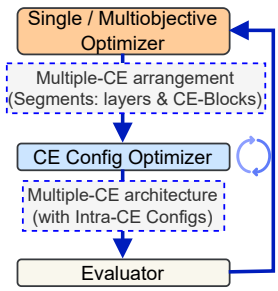
Similarly, predefined or intuition-based design choices are taken when it comes to **intra-CE configurations**. Deciding an optimal engine architecture

[5]Previous works usually do implicitly justify their design choices by comparing the end results against state-of-the-art and showing performance or efficiency improvements.

is in itself an open research problem with a large design space [72, 74, 111, 122]. Existing model-specific multi-engine accelerators use a uniform architecture. What changes across engines is only the degrees of parallelism [9, 150, 174, 197]. To give some examples: SPA [14] and TGPA [174] implement their engines as Systolic Arrays (SA) with different sizes; fpgaConvNet [166] adopts a fixed parallelism pattern across all engines (parallelism across output feature maps and inside a convolution kernel); all CLPs [150] engines parallelize across input channels and Filters. However, different layers of a DL model have varying computational characteristics; hence, using a single CE with the same parallelization strategy and dataflow to process these varying layers results in suboptimal performance and efficiency [10, 14, 174].



(a) MCExplorer core modules.



(b) MCExplorer main loop.

Figure 1.9: A high-level overview of MCExplorer components and its main optimization loop.

1.3.3.3 MCExplorer: Exploring the Design Space of Model-Specific Multi-CE DL Accelerators

To address the limitations of existing work, we propose MCExplorer, a design space exploration (DSE) framework for model-specific multi-engine accelerators. Given a DL model, FPGA resource budget, and user-defined optimization objectives, MCExplorer efficiently searches for optimized multi-engine accelerator configurations. Unlike existing approaches, MCExplorer explores a broader design space, including key design choices that are typically determined ad hoc. To the best of our knowledge, MCExplorer is the first framework that imposes no restrictions on inter-CE arrangement while simultaneously supporting diverse per-CE architectural choices (or intra-CE configurations). A

high-level overview of MCExplorer is presented in Figure 1.9. By exploring a broader space, MCExplorer consistently discovers multi-engine accelerator designs that surpass state-of-the-art. Furthermore, MCExplorer optimizes for different metrics, including throughput, latency, energy efficiency, and the tradeoffs among these metrics. MCExplorer integrates a collection of heuristics with different design philosophies, enabling fast and scalable exploration while reducing susceptibility to local optima and improving overall search quality. The contributions of this work are as follows:

- It proposes MCExplorer, a framework that explores the design space of model-specific multi-engine accelerators with flexible inter-CE arrangements and per-CE configurations. MCExplorer identifies an optimized multi-engine accelerator given a DNN model, hardware resource budget, and custom optimization objectives.
- It formulates a set of exploration heuristics that adopt different design philosophies, offering speed and scalability, and reducing the chances of converging to local minima. These include Genetic Algorithm (GA), Simulated Annealing (SA), and a novel, model-aware, search heuristic named Hierarchical Mapping (HM).
- It formulates a multiobjective optimizer based on Non-Dominated Sorting Genetic Algorithm (NSGA-II) to optimize multi-engine accelerators' performance and efficiency tradeoffs. We use the optimizer to identify multi-engine accelerators that simultaneously optimize performance and energy efficiency.
- It demonstrates the effectiveness of MCExplorer through an extensive evaluation using representative DNNs and hardware resource budgets. The evaluation shows that MCExplorer identifies optimized multi-engine accelerators that outperform the state of the art, considering different optimization objectives.

Our evaluation of MCExplorer with various DL models and FPGA boards shows that exploring the design space of multi-engine accelerators without restricting the inter-CE arrangements and per-CE configurations results in accelerators that outperform state of the art in all targeted optimization objectives, achieving up to 2.8× throughput improvement, 2.1× speedup, and 45% energy reduction. Moreover, the evaluation demonstrates that this comprehensive space exploration is crucial to identify multi-CE accelerators with the best performance-efficiency tradeoffs.

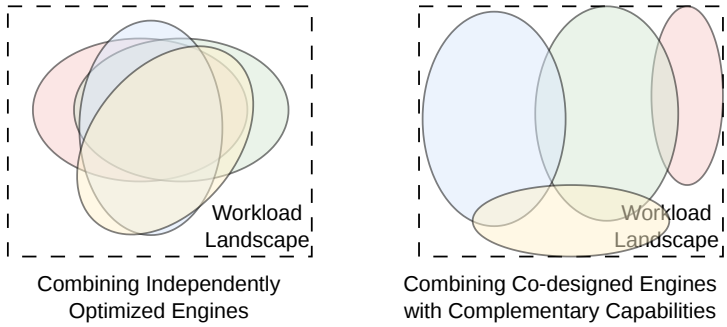


Figure 1.10: Workload landscape coverage using predefined independently optimized engines vs. co-designed engines. The area of an ellipse is an abstract indicator of how optimized an engine is. For example, it could correspond to the aggregate computations in workload layers where that engine achieves the "best" performance.

1.3.4 Flexible Multi-Engine DL Accelerator Design Methodology

The final contribution of the thesis is the Multi-Engine DL accelerator Design Methodology (MEDEM). MEDEM is a design methodology of flexible multi-engine DL accelerators with the goal of improving overall workload coverage. Hence, MEDEM addresses the third research question (**RQ3**). MEDEM is a two-stage design methodology where a pool of engines is co-designed, and then a combination among the co-designed engines that maximizes the coverage of diverse workloads is selected.

1.3.4.1 Background

The goal of flexible multi-engine DL accelerators is to support diverse DL workloads. To efficiently support a diverse range of workloads, the engines within a multi-engine accelerator must offer complementary capabilities, ensuring that layers with varying computational characteristics are effectively supported. We refer to this as maximizing the coverage of the workload set. As illustrated in Figure 1.10, maximizing coverage cannot be guaranteed when the engines are selected by combining existing and independently optimized ones. This is because independently optimized engines may perform best on the same subset or a largely overlapping subset of workloads, and thus their combination does not necessarily improve overall workload coverage. As the figure shows, a good combination containing "less optimized" engines, but that have lower overlap, can achieve the best workload coverage.

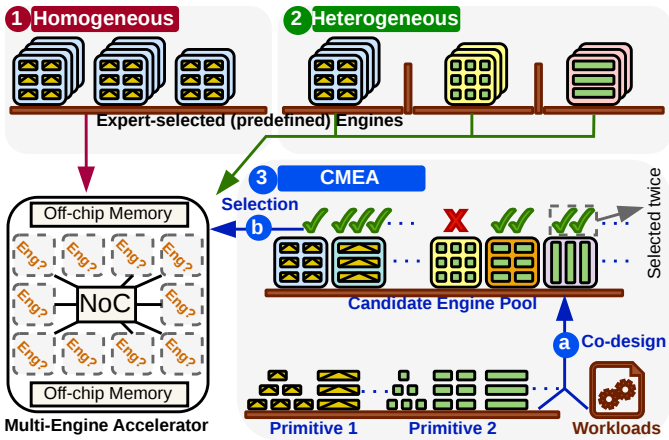


Figure 1.11: Engine design alternatives in multi-engine DL accelerators: 1 homogeneous engines 2 heterogeneous engines and 3 co-designed engines (CMEA). The design space expands moving from 1 to 2 to 3.

1.3.4.2 Engine Selection in Existing Flexible Multi-Engine DL Accelerators

In this work, the key design choice we focus on is the co-design and selection of engine microarchitectures in a flexible multi-engine DL accelerator. Existing flexible multi-engine DL accelerators can be classified, based on engine heterogeneity, into two categories. These categories are illustrated in Figure 1.11 (1 and 2). The first category of accelerators comprises homogeneous multi-engine accelerators. In these accelerators, all engines have identical hardware configurations [13, 144]. The second category consists of heterogeneous multi-engine accelerators. In these accelerators, the engines differ at either the architectural or microarchitectural level [29, 86, 120]. Current multi-engine accelerators typically rely on combinations of *independently optimized, expert-selected* engines. Examples of commonly used engine architectures are Systolic Arrays (SA), Eyeriss [18, 19], ShiDianNao [37], or NVDLA [118].

Adopting highly optimized architectures as a starting point to construct multi-engine accelerators is another example of the drawbacks of intuitive design choices (Section 1.1.4.1). It implicitly assumes that the best combination of engines is a combination of the best engines. However, as discussed in the previous section, this is not necessarily always the case. Ideally, the best coverage of diverse workloads can be achieved by co-designing an engine for each layer or kernel with distinct computational characteristics. However, this is not a practical setup within realistic resource constraints.

1.3.4.3 MEDEM: Multi-Engine DL Accelerator Design Methodology

To systematically search for engine combinations that achieve good coverage of a diverse set of workloads, we propose a Multi-Engine DL Accelerator Design Methodology (MEDEM). MEDEM is a two-stage design methodology, as highlighted in Figure 1.11 (a and b). The core idea is to first co-design a

pool of engines that would achieve an ideal coverage; each engine is optimized for a unique kernel occurring throughout the targeted set of workloads. Then *select* a combination (multiset) of these engines that is estimated to achieve the best coverage to build a multi-engine accelerator, given a practical resource budget. We call accelerators designed using this two-stage methodology Co-designed Multi-Engine Accelerators (CMEAs). Designing a CMEA requires tackling *two subproblems*: *co-designing candidate engines*, and *selecting a combination* with the best coverage of the workload set. Solving each of these subproblems requires exploring exponentially large design spaces.

MEDEM comprises a set of techniques that address the two subproblems of a CMEA design. Regarding the engine co-design, we propose fine-grained prioritization of selected axes of the hardware-mapping design space to make it easier to navigate. We *prioritize important axes* of the hardware-mapping space - like the engine PE array shape, or the tiling - based on their impact on the execution cost [76], regardless of whether they are in the hardware or the mapping space. As the prioritized axes are the most impactful, we aim to search them exhaustively. To enable this, we propose pruning techniques that leverage *domain knowledge* to prune redundant regions of the space and those that are guaranteed to be sub-optimal. Lower-priority axes are explored heuristically. We refer to this design strategy as Hierarchical Co-designer (HCo). To address the *second subproblem*, we propose Balanced Average and Specialization Search (BASS). The idea behind BASS is to balance two intuitive design principles: (1) designing for the common case by replicating the engine with the best average performance, and (2) maximizing specialization through engine heterogeneity. Rather than committing to either extreme, BASS explores multiple engine combinations, drawn from the pool generated by MEDEM's first stage, that span the spectrum from one extreme to the other. Each combination is then incrementally refined, and the combination with the lowest aggregate execution cost is finally selected. The contributions of this work are as follows:

- We propose MEDEM, a systematic design methodology for flexible multi-engine DL accelerators. MEDEM is a two-stage methodology that decouples engine co-design and selection.
- We propose HCo, a co-design strategy that efficiently navigates an engine hardware–mapping space by exhaustively exploring high-impact axes and heuristically exploring the rest.
- We propose BASS, a strategy for selecting engine combinations that maximizes workload-set coverage by exploring combinations ranging from ones designed for the common case to highly specialized ones.
- We comprehensively evaluate MEDEM and its derived CMEAs against state-of-the-art homogeneous and heterogeneous multi-engine accelerator designs, demonstrating MEDEM's scalability and generalizability.

Comprehensive evaluation of MEDEM using 51 single- and multi-model workloads shows that it identifies CMEAs that outperform state-of-the-art homogeneous and heterogeneous multi-engine accelerator baselines modeled after Simba [144] and Maelstrom [86], achieving geometric mean improvements up to 4.84× in energy-delay product (EDP) and 1.59× in throughput. These

improvements hold across various resource budgets ranging from those typical of edge to cloud systems.

1.4 Summary

This thesis proposes designs and systematic design methodologies of model-specific and flexible multi-engine DL accelerators. First, it presents a hybrid model-specific accelerator architecture, named **FiBHA**. FiBHA adopts a predominantly *empirically-driven* design methodology to identify opportunities for specialization and parallelism within a given DL model, enabling the derivation of a model-specific accelerator that optimizes performance under a fixed resource budget. The thesis then presents a cost model (**MCCM**) and a DSE framework (**MCExplorer**). MCCM and MCExplorer expand the scope by *analytically* modeling and *systematically* exploring a broader space of model-specific multi-engine DL accelerators. This broad exploration more fully exploits the potential of these accelerators. Finally, it presents a design methodology, named **MEDEM**, which covers flexible multi-engine DL accelerators. MEDEM co-designs and identifies engine combinations with complementary capabilities that maximize workload coverage across diverse DL models. Collectively, these contributions enable the identification of architectures that better exploit the performance and efficiency potential of both model-specific and flexible multi-engine accelerators.

The remainder of the thesis is organized as follows: Chapter 2 presents FiBHA, Chapter 3 presents MCCM, Chapter 4 presents MCExplorer, Chapter 5 presents MEDEM, and Chapter 6 concludes the thesis by summarizing the main contributions, discussing the limitations of the presented work, and outlining directions for future research.

Chapter 2

Hybrid Model-Specific Accelerator for Compact and Heterogeneous CNNs

This chapter presents FiBHA, a fixed-budget hybrid model-specific accelerator for compact and heterogeneous CNNs. FiBHA is a multi-engine accelerator that uses both single-CE and pipelined-CEs blocks described in Section 1.3.2.2. The goal of FiBHA is to exploit the specialization opportunities in compact CNNs by capturing both intra- and inter-layer heterogeneity under a certain resource budget. The chapter also presents (*SplitCNN*), a design heuristic that derives a FiBHA instance under a fixed resource budget and splits the CNN layers and the resources between the instance engines.

The FiBHA design flow is the first step of transitioning toward systematic design methodologies. On the one hand, the proposed design flow improves upon most existing model-specific multi-engine accelerators, where engine arrangements and architectures are predefined without an explicit design rationale. FiBHA engine arrangements and architectures are justified based on empirical analysis of compact CNNs. On the other hand, analytical modeling with more rigorous exploitation would be needed to co-design accelerators for models beyond compact CNNs.

2.1 Introduction

During the early years following the AlexNet breakthrough [81], researchers have mainly focused on designing Convolutional Neural Networks (CNNs) with higher accuracy [50, 151, 156, 192]. The accuracy improvements usually come at

This chapter is based on the following publications:
F. Qararyah, M. W. Azhar, and P. Trancoso, "Fibha: Fixed Budget Hybrid CNN Accelerator," in 2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). IEEE, 2022, pp. 180–190.
F. Qararyah, M. W. Azhar, and P. Trancoso, "An Efficient Hybrid Deep Learning Accelerator for Compact and Heterogeneous CNNs," ACM Trans- actions on Architecture and Code Optimization, vol. 21, no. 2, pp. 1–26, 2024.

the cost of higher memory consumption and computational complexity [160]. However, as CNNs have become dominant in computer vision, there has been a growing interest in improving their resource efficiency both to perform cheaper training and inference and to facilitate their deployment in resource-constrained environments [21, 57, 63, 79, 142, 154, 196]. At the same time, domain-specific accelerators that process Deep learning algorithms, including CNNs, more efficiently are being proposed [10, 18, 42, 100, 152, 154, 170, 176].

CNNs' resource efficiency is improved by reducing the model's computational complexity using modules - building blocks or layer combinations - with reduced memory and computational requirements [21, 57, 63, 142, 160, 196]. These modules require fewer weights and perform fewer computations. Hence, they enable building compact, computationally inexpensive, yet accurate CNNs [11, 57, 63, 156, 160, 196]. However, these modules introduce new layer types that increase a CNN's inter-layer-type heterogeneity. We use the term heterogeneity to describe the diversity of the layers in a CNN, especially the computationally intensive convolutional layers. For example, we say that ProxelessNAS [11] is more heterogeneous than VGG [151] because it contains more types of convolutional layers. And because its convolutional layers that are of the same type have more variations of kernel sizes, input sizes, shapes, and reuse patterns.

An in-depth analysis of monolithic accelerators, e.g., Tensor Processing Unit (TPU) [71], reveals that the "one-size-fits-all" heterogeneity-oblivious accelerators are very inefficient when it comes to processing heterogeneous CNNs [10,184]. Therefore, there has been an effort to design custom accelerators for these heterogeneous CNNs. FPGAs have been heavily used to develop such custom and model-specific accelerators due to their reconfigurability, that enabled supporting the rapidly changing CNN models [5, 41, 98, 100, 152, 176, 179, 186].

Most of the proposed custom and model-specific accelerators are composed of reusable engines, where a *single engine* computes *multiple layers* of the same type. We refer to such accelerators as single-engine multiple-layer accelerators (SEML). For instance, in [96, 98, 152, 176] a *single engine* processes all the Depthwise (DW) convolutional layers [21] and another *single engine* processes all the Pointwise (PW) convolutional layers. Designing dedicated Depthwise and Pointwise engines, each optimized for a layer type, improves efficiency compared to a monolithic accelerator that processes layers of both types using the same engine. Hence, we say that the SEML accelerators capture inter-layer-type heterogeneity. However, layers of the same type have various arithmetic intensities, input and output shapes, reuse patterns, workloads, and filter sizes [10, 170]. Consequently, even when there is a dedicated engine per layer type, this engine has to be optimized for the average case within that layer type. Hence, we say that single-engine multiple-layer accelerators do not capture intra-layer-type heterogeneity.

To capture the intra-layer-type heterogeneity, accelerators with a *dedicated engine per layer* are proposed [9, 97, 163, 166]. We refer to such accelerators as single-engine single-layer accelerators (SESL). They are known as streaming, or synchronous dataflow (SDF), accelerators in the literature. These accelerator engines are pipelined and run simultaneously to achieve high throughput. Implementing a pure SESL design where each layer has its dedicated engine

and where resources are well balanced among these engines is challenging and does not scale for deep models [43, 107]. Moreover, the simultaneously running engines need to be fed with data at the same time, requiring high memory bandwidth [97].

To summarize, neither pure SEML nor pure SESL accelerator architectures offer the best performance-resource trade-off. While SEML accelerators ignore the intra-layer-type heterogeneity, SESL accelerators are resource-hungry and unscalable. Hence, we propose *Fixed Budget Hybrid CNN Accelerator (FiBHA)* that combines both SESL and SEML architectures. We use the term hybrid to describe an accelerator that uses more than one mapping between CNN layers and the hardware, e.g., a combination of SESL and SEML mappings. FiBHA captures more heterogeneity than pure SEML accelerators, and is more resource-efficient than pure SESL. To derive a FiBHA instance given a fixed resource budget, we propose a heuristic "Split a CNN" (SplitCNN). SplitCNN identifies a splitting point; it splits the CNN layers and the resources between that instance's SESL and SEML parts.

To design the SESL part of FiBHA, one option is to implement a traditional pipeline, where a tile of each pipelined layer's feature maps (FMs) is stored in a double-buffered or FIFO fashion [97, 137, 166]. However, this introduces a non-negligible memory overhead. This overhead scales with the pipeline length since more buffers must be added between the engines. Hence, to design a memory-efficient single-engine single-layer accelerator part of FiBHA, we propose Fused-Inverted-Residual-Bottleneck (FIRB), a fine-grained and memory-light single-engine single-layer architecture building block. This block reduces memory and energy overheads compared to traditional pipelining. The core idea of the FIRB-based design is building a two-level pipeline composed of a chain of *fused engines*, or engine groups, rather than engines. Within each *fused engine*, the communication among its engine group is done at a fine granularity, which enables processing the intermediate results in a more timely manner and eliminates the need for double buffering.

The contributions of this work are as follows:

- We propose FiBHA (Fixed budget hybrid accelerator), a hybrid architecture composed of a single-engine single-layer (SESL) part and a single-engine multiple-layer (SEML) part, each computing a subset of CNN model layers.
- We propose SplitCNN, a heuristic that derives a FiBHA instance given a fixed resource budget, splits a CNN and the computational resources between its SESL and SEML parts in a way that maximizes the throughput.
- To implement the SESL part of FiBHA efficiently, we propose FIRB, a fine-grained and memory-light SESL building block, and evaluate its impact on implementing a pipelined architecture in terms of improving energy and memory efficiency.
- We analyze the impact of FiBHA on the memory requirements of a CNN. FiBHA reduces the intermediate results memory requirements considerably, allowing better scaling and avoiding off-chip communication, especially on memory-constrained hardware.

- We evaluate fibha by comparing its performance to a pure SEML accelerator using three different boards representing various resource budgets in the compute continuum, namely ZC706, KCU105, and ZCU102. Moreover, we evaluate FiBHA, comparing its performance to a pure SESL accelerator and demonstrating that it offers better performance and resource efficiency. We also evaluate FiBHA's energy efficiency, comparing it to a pure SEML accelerator and two GPUs using a set of heterogeneous CNNs.

In this work, we focus on accelerating inference [154]. We evaluated FiBHA using different FPGAs with varying resource budgets representing different points in the compute continuum, and state-of-the-art compact CNNs [11,57,142,159]. FiBHA achieves 2.5x and 4x of the throughput achieved by a state-of-the-art model-specific accelerators, under the same hardware budget. It also allows better scaling in cases of memory-constrained hardware compared to SEML. FIRB modules reduce the required memory by up to 54%, and energy requirements by up to 35% compared to traditional inter-layer pipelining.

2.2 Background

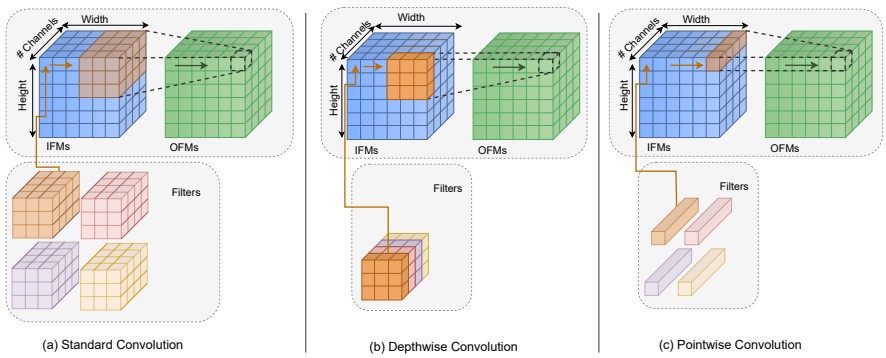


Figure 2.1: (a) Standard, (b) Depthwise, and (c) Pointwise convolution.

2.2.1 Convolutional Neural Networks (CNNs)

Convolutional neural networks (CNNs) are feedforward Deep learning models designed to capture features in 1D signals like language, 2D signals like images. or 3D like video or volumetric images [91]. A CNN consists of a set of stacked layers performing feature extraction and classification [92]. The primary layers in a CNN, both in terms of functionality and computational intensity, are the convolutional layers [18]. Figure 2.1 shows different forms of convolution that are found in CNNs. As the figure shows, a convolutional layer has a set of **filters**, which are composed of trainable **weights**. The filters are applied to input feature maps (**IFMs**) to extract embedded features from them, generating

output feature maps (**OFMs**). We use the term Feature Maps (**FMs**), or activations, to refer to both IFMs and OFMs.

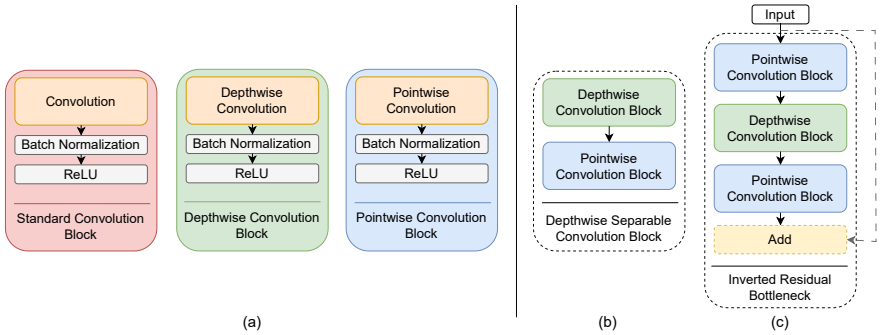


Figure 2.2: (a) Convolution blocks, (b) Depthwise Separable Convolution, and (c) Inverted Residual Bottlenecks.

2.2.2 Resource efficient CNNs

Resource-efficient CNNs are also referred to as heterogeneous, compact, or edge CNNs in the literature [10, 184]; hence, we use these terms interchangeably throughout this paper. These CNNs are designed to balance the accuracy-efficiency trade-off. They either improve the accuracy without increasing the model weights and computations [21] or reduce the model weights and computations considerably at the cost of a negligible loss in accuracy [11, 57, 142, 160, 196]. The Depthwise separable convolution, inverted residual, and linear bottlenecks are two predominant building blocks of resource-efficient CNNs [11, 21, 57, 142, 160, 196].

The Depthwise Separable Convolution (DSC) is a form of convolution that is based on decoupling the spatial and the cross-channel correlations in the feature maps [21]. This is achieved by replacing the standard convolution with two operations: (a) Depthwise convolution (**DW**) and (b) Pointwise convolution (**PW**). As Figure 2.2(b) shows, a Depthwise Separable Convolution is composed of Depthwise followed by Pointwise convolution. Figure 2.1(b) shows that the Depthwise convolution applies a filter to each feature map and sums each feature map's results individually (no across-FMs summation). This is followed by a Pointwise convolution with 1×1 filters. Pointwise convolution, as shown in 2.1(c), applies a filter and sums the results across different feature maps but not within a single feature map. Depthwise Separable Convolution is a more efficient way to use the model parameters compared to the standard convolution [21, 179].

The inverted residual with linear bottlenecks is a module designed to significantly reduce the CNN model weights and operations while maintaining the accuracy [142]. As shown in Figure 2.2(c), this module combines three convolutional layers. The first layer is a 1×1 Pointwise convolution that *expands* the IFMs by increasing their depth. The second is a Depthwise convolution that processes the expanded FMs. The third is another 1×1 Pointwise convolution

that *squeezes* the FMs again. There could be a shortcut connection in the module that forwards the input of the module to be added to the outputs of the last convolutional layer of the module. This shortcut is represented by the dashed arrow connecting the module input to an *Add* layer in Figure 2.2. The reduction in weights and computations that the inverted residual with bottlenecks module permits is a result of performing heavy operations like the Pointwise convolution on a relatively low-dimensional representation, that is, the unexpanded or the squeezed FMs. And a light operation like Depthwise convolution on the expanded representation.

Since the modules used to design resource-efficient CNNs have different types of convolutional layers compared to the conventional CNNs that have only one type of convolution, we say that the resource-efficient CNNs have more heterogeneity. Note that even conventional CNNs have one form of heterogeneity, which is intra-layer-type. This is because even convolutional layers of the same type have variations in inputs, outputs, and filter shapes.

2.2.3 Model-aware CNN accelerators

A plethora of custom, model-specific accelerators have been proposed, aiming to improve the performance and efficiency of processing compact and heterogeneous CNNs. We categorize the custom accelerators in the literature into four categories:

- **Monolithic accelerators:** accelerators in which all the core layers are executed using the same engine [5, 18].
- **Single-Engine Multiple-Layer (SEML):** accelerators in which all the layers of a certain type are executed using the same engine [5, 98, 100, 152, 176, 179, 186].
- **Single-Engine Single-Layer (SESL):** accelerators in which each layer is mapped to a distinct engine. In these accelerators, the chain of engines is pipelined to work concurrently and maintain a high throughput [9, 97, 163, 166].
- **Multiple-Engine Multiple-Layer (MEML):** accelerators in which a single layer is processed by multiple engines, where each engine processes a tile of that single layer, and these multiple engines are reused across multiple layers [42, 107].

In the rest of this paper, we mainly focus on the second and third categories, i.e., SEML and SESL. This is because the MEML is more tailored for resource-demanding scenarios like having large DNNs or the training phase. Note that in this context, the term *single-layer* refers to a combination of convolution, batch normalization, and activation layers. We assume that a convolutional layer is fused with the batch normalization (BN) and the activation layer, e.g., rectified linear unit (ReLU), following it. Note also that our taxonomy is based on the nature of the mapping between the model layers and the engines rather than the engine count. For example, considering a SEML, there could be multiple engines, but the point is that each *single-engine* computes *multiple layers*. Figure 2.3 shows examples of both SEML and SESL mapping approaches. The

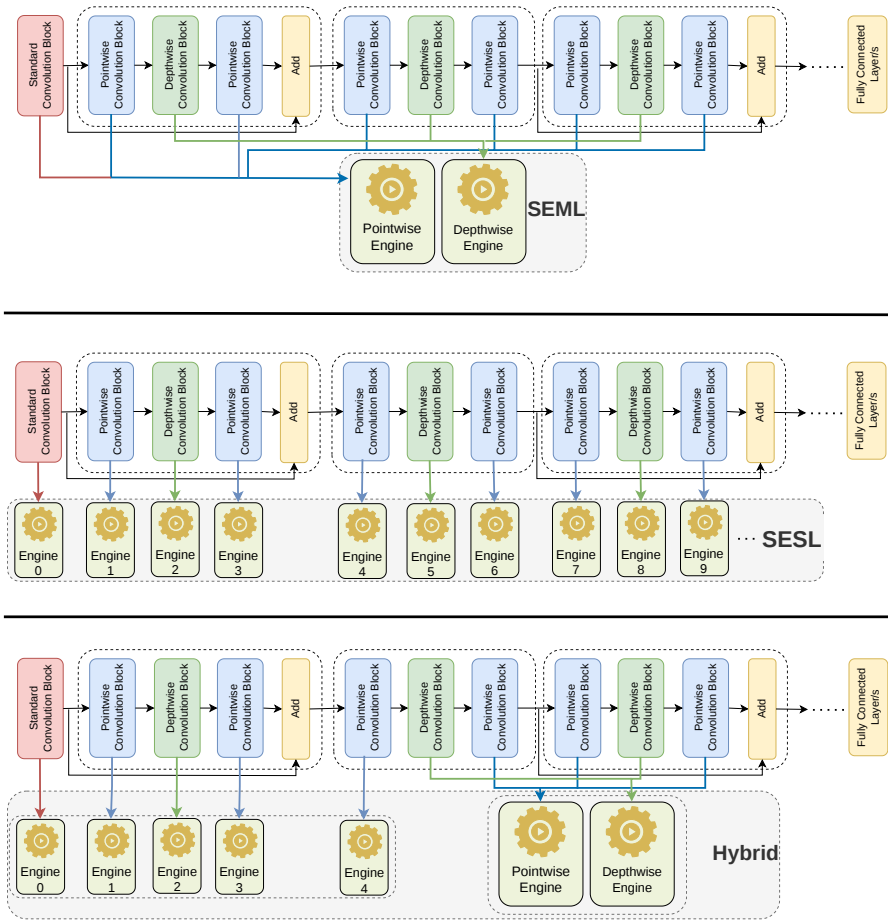


Figure 2.3: High-level overview of the SEML, SESL, and Hybrid accelerators. The figure focuses on the mapping from the convolutional layers of a CNN to the compute engines of the accelerators; the internals of the engines, their connectivity, and the memory system are omitted for simplicity.

top part of the figure shows an example of an SEML where all the Depthwise layers are mapped to a depthwise engine, and all the Pointwise layers and the first layer are mapped to a Pointwise engine. This is a common mapping in custom accelerators targeting heterogeneous CNNs. In this mapping, Pointwise and standard convolutions share the same engine. The middle part of the figure shows an SESL mapping where each layer is mapped to its own dedicated engine. Note that our taxonomy is independent of the internal design of these engines. These engines could either be custom convolution engines, each highly optimized for its layer. Or a set of Systolic Arrays with the same architecture, but with different dimensions to suit their varying layer workloads. The bottom part of the figure depicts what we refer to as *Hybrid Accelerator*. We use the term hybrid to describe an accelerator that uses more than one mapping technique between CNN layers and the compute engines. The figure shows an example that maps each of the first 5 convolutional layers to its engines and uses a SEML mapping technique for the rest of the layers.

We refer to a SESL where all the engines have the same execution time as *perfectly balanced*. In a perfectly balanced SESL, theoretically, all engines are active all the time, eliminating any resource idleness or underutilization. We use the term processing element (**PE**) as an abstraction. It refers to hardware capable of doing a MAC operation regardless of the components that are actually used in a particular implementation (e.g., in an FPGA, a PE could be implemented using Lookup Tables (LUT) or a Digital Signal Processor (DSP)).

2.3 Motivation

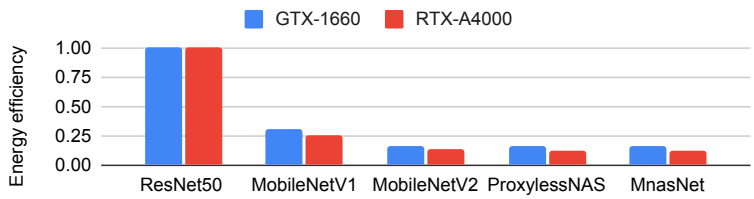
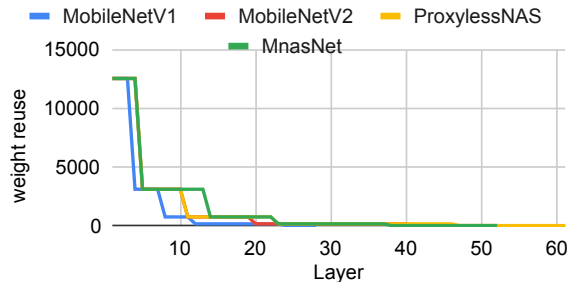


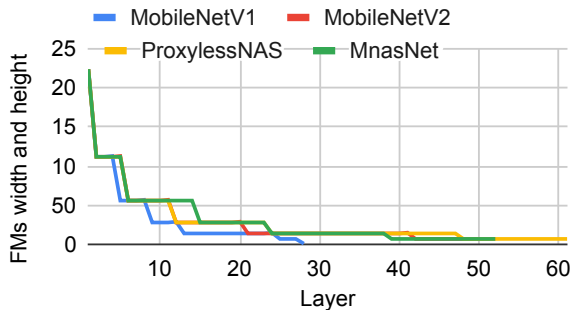
Figure 2.4: Energy efficiency of a set of heterogeneous CNNs normalized to the energy efficiency of ResNet50.

The Depthwise Separable Convolution (DSC) and the inverted bottlenecks (Section 2.2) reduce CNN's storage and computational requirements. However, they increase the CNN model heterogeneity as these modules include layers of varying arithmetic intensities, reuse opportunities, FMs, weight shapes, and memory requirements. This makes the generic accelerators unable to fully achieve the desired performance gains out of this reduction. Figure 2.4 shows an example of that; it depicts the energy efficiency of four compact and heterogeneous CNNs normalized to that of ResNet50 [50]. In contrast to ResNet50, these heterogeneous CNNs have much lower energy efficiency.

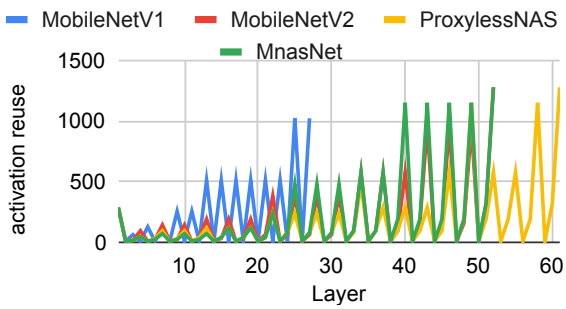
Figure 2.5a, and 2.5c depict some aspects of the heterogeneity in a set of compact CNNs, namely MobileNetV1 [57], MobileNetV2 [142], Proxyless-



(a)



(b)



(c)

Figure 2.5: **(a)** Weight reuse. **(b)** FMs spatial dimensions - width and height - (note that FMs width = FMs height). **(c)** Activation (FM element) reuse .

NAS [11], and MnasNet [159]. Figure 2.5a depicts the weight reuse of the models on the y-axis and layers on the x-axis. We use the term *reuse* to indicate the number of times, or operations, where a single element (weight/input/output) is used. To give an example, in Figure 2.1(c), each of the Pointwise filters slides over the width and height of the IFMs, and is used to produce one output feature map. Since the number of elements in one output feature map equals *OFMs width * OFMs height*, a single weight or filter could be loaded from memory once and used to do that many computations. The reuse indicates the compute-to-memory access ratio of an element. Capturing reuse is important as it reduces data movement, which in turn reduces the latency [18, 105] and energy. Figure 2.5a shows that the initial layers have high and dynamic weight reuse; the reuse is low and stable in the rest. Figure 2.5b shows the width and height of the FMs in the targeted CNNs. The figure shows that the FMs of the initial layers have relatively large widths and heights. As a result, the weight reuse in these layers is the highest. The figure also suggests that in the first layers, the width and height of the FMs change, more specifically decrease, frequently, and after that, they change less frequently. As a result of the frequent changes in the FM's width and height, the weight reuse is more dynamic in these first layers. As a result of the less frequent change in the rest of the layers, the reuse does not vary as much.

Figure 2.5c depicts the activation reuse. For DW layers, activation reuse is calculated as *filter width* $\times$ *filter height*, except for the activation on IFM borders and in case of strides of greater than one. For PW layers, activation reuse is equivalent to the *number of filters*. Activation reuse follows an opposite trend compared to weight reuse, low in the initial layers and high in the rest. Moreover, unlike weight reuse, activation reuse fluctuates. This is a result of the inverted bottleneck module and the DW convolutions. The peaks of the activation reuse represent the reuse values of all the PW layers in the MobileNetV1 case and the expansion layers (Section 2.2) in the rest. The valleys represent the DW layers. Note that activation reuse is negligible in the case of DW layers. Hence, regarding activation reuse, we focus on the higher and more dynamic reuse of the PW layers.

Reuse heterogeneity is one of many forms of heterogeneity in the studied CNNs. SEML accelerators fail to capture the intra-layer-type form of heterogeneity. SESL accelerators could address the two forms, but they are impractical for deep CNNs and do not scale [107, 135]. Researchers proposed several techniques to alleviate SESL bottlenecks [6, 9, 166], but each of these techniques has its own shortcomings.

Motivated by the observation that neither SEML nor SESL accelerators address the trade-off between addressing heterogeneity and maintaining resource efficiency well enough, we propose our hybrid architecture (FiBHA). Based on Figure 2.5a and 2.5c, we can generalize by saying that the layers in the targeted CNNs can be split into two parts. The first part has high weight reuse and low input reuse, while the second has low weight reuse and high input reuse (considering input reuse only in the PW layers). This trend motivates designing a hybrid accelerator that uses two forms of architectural design to handle these two parts.

2.4 FiBHA

2.4.1 Design Choice: First SESL Then SEML

The primary design goal of FiBHA is to improve efficiency by capturing the two forms of heterogeneity present in resource-efficient CNNs as much as the available resource budget permits. The first form of heterogeneity is the inter-layer-types heterogeneity, which is a result of having different types of layers, including Depthwise, Pointwise, and Standard convolution. The second is heterogeneity among layers of the same type, represented in variations in input and output shapes, sizes, filter shapes, and reuse patterns as discussed in Section 2.3. When designing FiBHA, we mainly consider the PEs budget. However, we also target minimizing memory consumption, which leads to energy savings, and we avoid the off-chip memory bandwidth bottleneck. In the rest of this section, we discuss why using SESL to process the initial layers of a CNN and SEML to process the rest is the option that fulfills these design goals. It is important to note that while this arrangement is the best for CNNs, as we show in the rest of this section, our design is not restricted by it, meaning that in other workloads, it could be beneficial to use other arrangements.

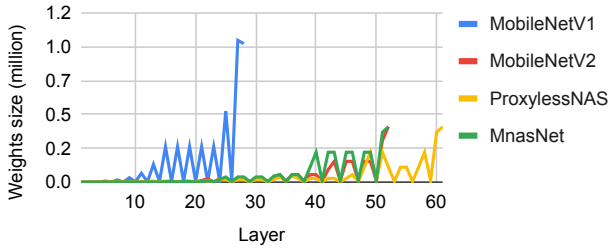
2.4.1.1 Maximizing the captured heterogeneity

SESL captures the two levels of heterogeneity; hence, it is preferred over SEML when there is more heterogeneity. The initial layers exhibit more heterogeneity than the rest. Three forms of heterogeneity are much more visible in the initial layers:

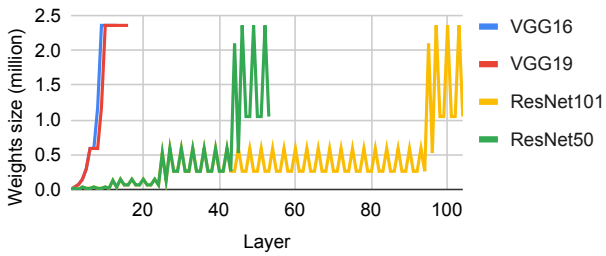
- **Weight reuse heterogeneity:** Figure 2.5a shows that the weight reuse in the initial layers is changing dramatically. However, in the rest of the layers, the weight reuse is either stable or changes negligibly.
- **Parallelism patterns heterogeneity:** The initial layers, especially the first layer, have shallower inputs compared to the rest. This means they exhibit different parallelism patterns. As a result, these layers have low resource utilization when computed using an engine that is suitable for the majority of the layers. Previous work suggested computing the first layer on the CPU [17, 113].
- **Inter-layer-type heterogeneity:** Another form of heterogeneity that the first layer introduces, considering all the compact CNN models we have evaluated, is inter-layer-type heterogeneity. This layer is the only standard convolution, making its filters unique.

2.4.1.2 Minimizing FMs memory requirements

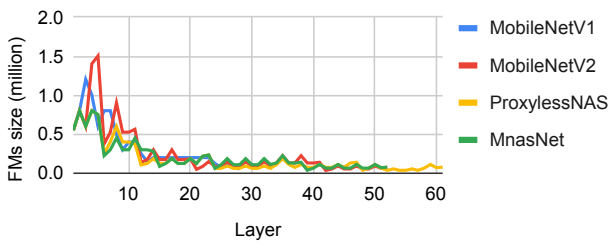
The memory consumption of Deep Learning algorithms represents a major bottleneck both in training and inference [42, 132]. In some state-of-the-art accelerators, on-chip buffers consume up to 70% to 87% of the chip area [18,42]. As a result, reducing memory requirements is a crucial part of designing an efficient accelerator.



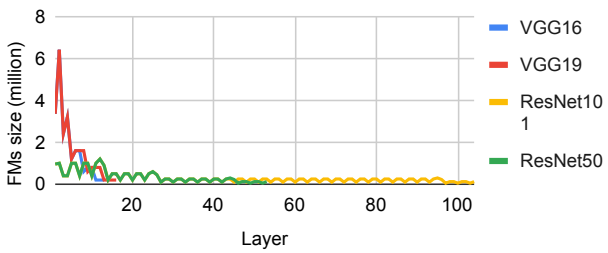
(a)



(b)



(c)

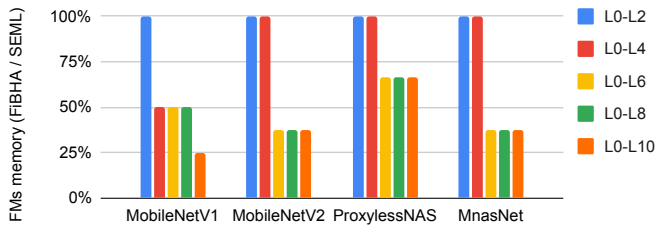


(d)

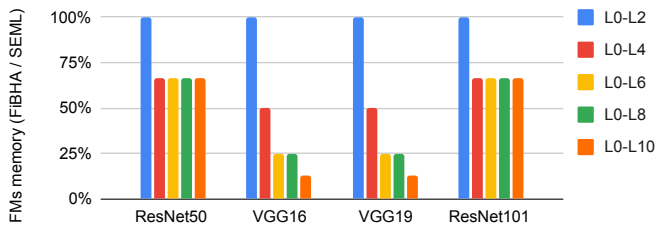
Figure 2.6: Weight size per layer of (a) heterogeneous and (b) less-heterogeneous CNNs. FMs size per layer of (c) heterogeneous and (d) less-heterogeneous CNNs, respectively.

Figure 2.6c, which depicts layers on the x-axis and the size of FMs on the y-axis, shows that the initial layers have much larger FMs than the rest. CNNs' initial layers extract many simple features from an input, e.g., edges and curves in an image. As we go deeper, the layers combine the many simple features into fewer, more abstract features. That is why FMs' size shrinks as we go deeper. The shrinking is done in practice by sub-sampling through pooling layers or skipping through strided convolutions.

Unlike SEML, where the outputs need to be fully stored in memory and then loaded when the next layer is computed, in SESL, data flows between the pipelined engines and is processed almost immediately. As a result, if SESL is used to process the initial layers, smaller on-chip buffers are needed to store the FMs. On the other hand, the rest of the layers have small FMs, meaning that storing them fully is relatively cheap.



(a)



(b)

Figure 2.7: Maximum FMs memory requirement of FiBHA normalized to SEML or monolithic accelerator considering various SESL part lengths of **(a)** heterogeneous and **(b)** less-heterogeneous CNNs. L0-Lx means that the SESL part of FiBHA spans the layers starting from layer 0 up to layer x - 1.

Figure 2.7a shows the FMs memory requirement of FiBHA normalized to SEML or a monolithic accelerator. The figure shows that up to 75% of the required FMs memory could be saved. A SEML accelerator requires an FMs buffer size that can accommodate the maximum IFMs plus the OFMs of a layer. This assumes that two buffers are allocated to hold the IFMs and OFMs of a layer and are reused in an alternating fashion throughout an inference. FiBHA requires an FMs buffer size that can accommodate the maximum IFMs plus the OFMs of a layer in the SEML part (starting from the last SESL layer + 1) plus the SESL part overhead. The SESL part overhead depends on the granularity of the pipeline and is negligible for all the considered SESL lengths. Note that in the case of having skip connections, as in the case of inverted

bottlenecks (Section 2.2), a layer has more than one input or output. For models that have a linear structure with no skip connections, like MobileNetV1 or VGG, one could make a rough estimate of the FMs buffer size reduction by looking at Figure 2.6c or 2.6d. The size of the required buffer in the case of a pure SEML accelerator is the sum of the two highest peaks. In the case of FiBHA, the required buffer size is roughly the sum of the two highest peaks starting from FiBHA's SEML part's first layer. Equation 2.1 gives the memory reduction for any CNN, ignoring the SESL part's small buffering overheads, where *sl* is the splitting layer, which is the first layer processed by the second part of FiBHA, and *ll* is the last layer in the CNN.

$$
\begin{aligned}
FMs'\ memory\ reduction = \max_{l_x \in [0, ll]} \Bigg(& \sum_{IFMs \in l_x\ inputs} size(IFMs) \\
& + \sum_{OFMs \in l_x\ outputs} size(OFMs) \Bigg) \\
- \max_{l_y \in [sl, ll]} \Bigg(& \sum_{IFMs \in l_y\ inputs} size(IFMs) \\
& + \sum_{OFMs \in l_y\ outputs} size(OFMs) \Bigg)
\end{aligned}
\tag{2.1}
$$

This reduction in FMs' memory requirements has two benefits. First, when using on-chip memory to store FMs, relying on large buffers is expensive both in terms of latency, area, and power [42]. Second, embedded accelerators may not have sufficient on-chip memory; in this case, FiBHA helps to fit the FMs on a smaller on-chip memory and avoid relying solely on the costly-to-access DRAM. In section 2.6.1, we demonstrate the benefits of FMs memory reduction. Note that the results of the analysis *in this section* are generalizable and apply to other more homogeneous CNNs, like ResNet [50] and VGG [151]. Figure 2.6b and 2.6d show that these CNNs' FMs and weights follow similar trends; while the weights grow bigger, FMs shrink as we go deeper. Hence, the advantage of FiBHA is not limited to compact CNNs; it applies to other CNNs as shown in Figure 2.7b.

2.4.1.3 Avoiding off-chip memory bandwidth bottleneck

A SESL architecture suffers from the off-chip memory bandwidth bottleneck since all the pipelined engines must be supplied with weights concurrently. As Figure 2.6a shows, the initial layers have small weights that can be stored on-chip locally. As a result, these layers could be processed with an SESL architecture while avoiding off-chip memory bandwidth being a bottleneck. On the other hand, the layers towards the end of a model have much larger weights that usually have to be stored off-chip. Note that the increase in the size of the weights as we move from the first towards the last layers happens because the depth of the FM channels increases, requiring more and larger filters.

Based on this analysis, to capture more heterogeneity, minimize memory requirements, and avoid bandwidth bottlenecks, the SESL part is used to

process the initial layers. The SEML is used to compute the rest.

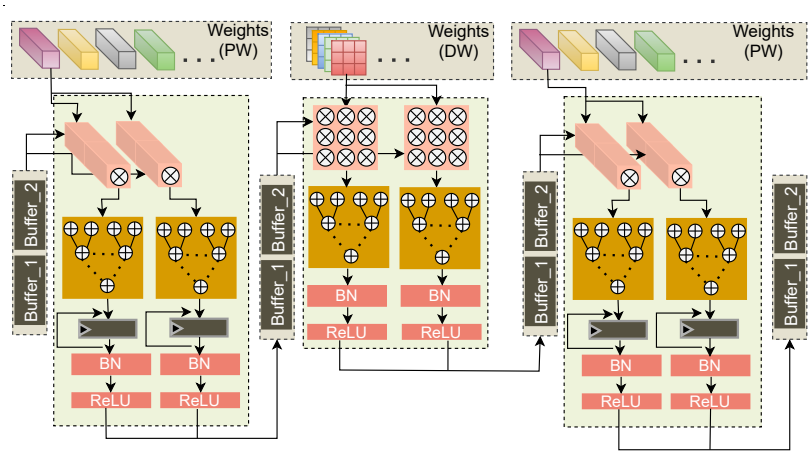
2.4.2 Fused Inverted Residual Bottlenecks(FIRB)

As discussed in section 2.4.1.1, the motivation behind using SESL to process the first layers is minimizing memory requirements. However, in a traditional implementation of the pipeline, a tile of the FMs of each layer needs to be stored on-chip and double-buffered for concurrent access, introducing a non-negligible overhead. This overhead scales with the pipeline length as more double buffers must be added.

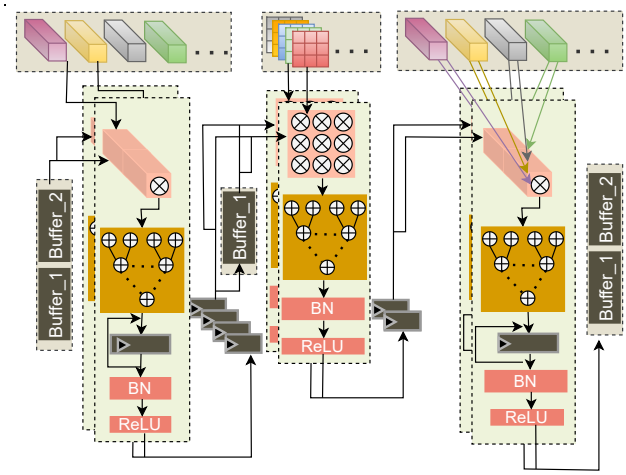
To reduce the buffering required by a pipelined architecture, *Gao et al.* have proposed a novel dataflow named Alternate Layer Loop Ordering (ALLO) [43]. However, it is only possible to apply ALLO to alternate pairs of adjacent layers. Hence, it requires fully delaying and completely buffering the FMs of half of the pipelined layers. While ALLO improves over naive coarse-grained pipelining, where the whole FMs need to be entirely stored between each pair of layers, it is still very expensive when considering a limited on-chip memory. Hence, we introduce the Fused Inverted Residual Bottlenecks (FIRB) module. FIRB applies a fine-grained pipelining within the inverted residual bottleneck modules (Section 2.2), resulting in a considerable reduction in the required buffers.

Figure 2.8a and 2.8b show two implementations of an Inverted Residual Bottleneck module. The implementation in Figure 2.8a is a variant of traditional inter-layer pipelining. This implementation maximizes weight reuse across the three layers of the module. Figure 2.8b shows an implementation of FIRB; this implementation maximizes input reuse in the first layer (the expansion layer), sacrifices the reuse in the second (the DW layer), and maximizes the output reuse in the third (the projection layer). The main advantage of FIRB is eliminating the need for three out of the four buffers connecting the expansion to the DW and the DW to the projection layer (Figure 2.8b). Note that the four intermediate buffers are three to six times bigger than the four outer buffers because they hold *expanded* FMs (Section 2.2), and the usually used expansion ratios are three and six times. One of the four intermediate buffers has to be kept to hold the FMs' values that are shared between two consecutive passes of the DW layer filters.

The advantage of reducing buffers allowed by FIRB comes at the cost of sacrificing the reuse in the DW layer and imposing an extra constraint on the parallelism patterns. Sacrificing the reuse in the initial DW layers, as the SESL part targets the initial layers of a CNN, is not costly, as they hold hundreds of weights, which are negligible compared to the millions of weights in the whole model. The parallelism patterns constraint requires that the three layers target the three dimensions of the FMs in a way that permits having a common blocking factor across them (the three layers). Common blocking is important to balance the computational loads and avoid leaving any of the engines idle. One challenge that this constraint introduces is the need to gather the weights from multiple filters in the projection layer, as shown in Figure 2.8. This is required in case two or more levels of memory are used. However, this challenge can be resolved by simply rearranging and storing the layer weights in a round-robin fashion offline. Note that the weights do not change during



(a)



(b)

Figure 2.8: **(a)**Traditional inter-layer pipelining-based implementation of an Inverted Residual Bottleneck module. **(b)** FIRB-based implementation of an Inverted Residual Bottleneck module.

inference, so the rearranging is done only once.

We implement a relatively coarse-grained pipelining across FIRB's modules. This is because the fine-grained pipelining cannot be extended to cover multiple modules, as the expansion and the projection layers have to work in an alternating fashion to maintain the common blocking factor. This means that a FIRB module produces a set of *d_squeezed* elements each *d_expanded* cycles, where *d_squeezed* is the depth of the squeezed FMs and *d_expanded* is the depth of the expanded FMs, rather than a single element per cycle. However, compared to ALLO [43], our coarse-grained pipelining has finer granularity and does not require stalls or storing the whole FMs.

FIRB supports models that use the Inverted Residual Bottleneck as a building block; these represent the majority of the compact CNNs [11, 142, 159, 196]. However, the older families of compact models, like MobileNetV1, do not have this module. Hence, to support both, we construct the SESL part of FiBHA using FIRB or traditional pipelining based on the targeted model as shown in Figure 2.9.

2.4.3 Architecture Overview

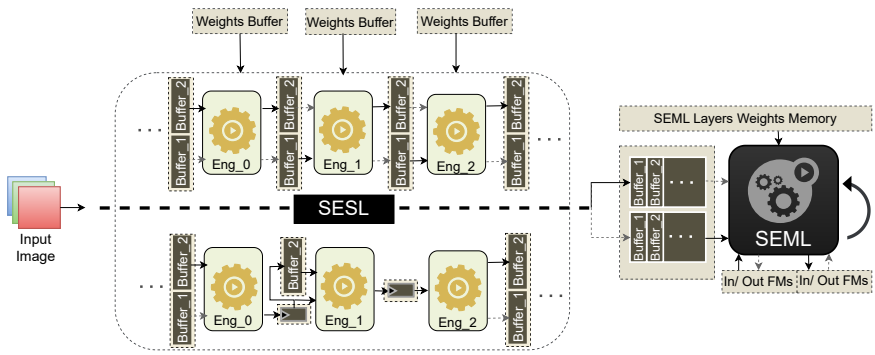


Figure 2.9: FiBHA architecture: a high-level overview. The SESL part implementation is FIRB-based or traditional tiled-pipelining-based depending on the CNN.

Figure 2.9 shows a high-level overview of FiBHA architecture. A FiBHA instance is composed of two main parts: a SESL and a SEML. Double buffering is used to bridge these two parts such that both can work concurrently. For instance, while the SEML part is processing the input n, the SESL could process the next input $(n + 1)$. The SESL part is composed of a chain of FIRB modules if they support the targeted CNN; otherwise, it is composed of a tiled traditional inter-layer pipelining. The SEML part is represented as a black box to indicate that any SEML design, e.g., the two-engines design presented in Figure 2.3, could be used as a part of FiBHA. This is explained in detail in the next section (Section 2.4.4).

In the SESL part, the engines are pipelined, and double buffering is used between the engines when necessary. Note that these buffers are much smaller than SEML buffers. In addition to the buffers between the engines, each engine

has its own weight buffer. The input image is processed in chunks. Assuming we have l engines in the SESL part, while Engine_0 is processing chunk $l+n$, Engine_1 handles chunk $l+n-1$, and so forth. Once Engine_$l-1$ completes a chunk, it pushes it to one of the two buffers connecting the SESL part to the SEML part. Each of these engines is assigned a number of PEs such that the execution times of the engines are the same, which makes the SESL part perfectly balanced.

Layers $l+1$ up to the last layer are executed using the SEML part. This part could be implemented using any SEML architecture [5,98,100,152,176,179,186]. This part processes its layers one by one and has two buffers used in an alternating fashion to load and store the inputs and the outputs.

The implementation details of the SEML part engines are not a part of the core contribution of this work. However, in Section 2.5, we provide the details of an FPGA-based implementation example. To be as generic as possible, FiBHA works at a higher level of abstraction, making the following assumptions:

I Any SEML accelerator, e.g. $SEML_x$, is not capable of capturing the intra-layer-type heterogeneity. This results in inefficiencies like execution time/throughput overhead.

II Each engine in a SESL accelerator should have a negligible overhead. This is because it is designed and optimized for a specific layer. If the workloads are perfectly balanced among all the SESL engines, we could generalize by saying that the SESL part has a negligible overhead as well.

From 1 and 2, given $SEML_x$ and an available PEs budget and a CNN (CNN_x) we argue that a hybrid architecture that is composed of a SESL part and $SEML_x$-like part outperforms the pure $SEML_x$ if: (a) CNN_x can be split, and PEs can be redistributed such that the ratio of $SEML_x$ PEs to its workload remains the same or increases, (b) the SESL part execution time, is upper-bounded by the SEML execution time. Note that we apply the second assumption regardless of the implementation of the SESL part, which could be FIRB-based or traditional pipelining.

2.4.4 SplitCNN: A Heuristic to Derive FiBHA Instance

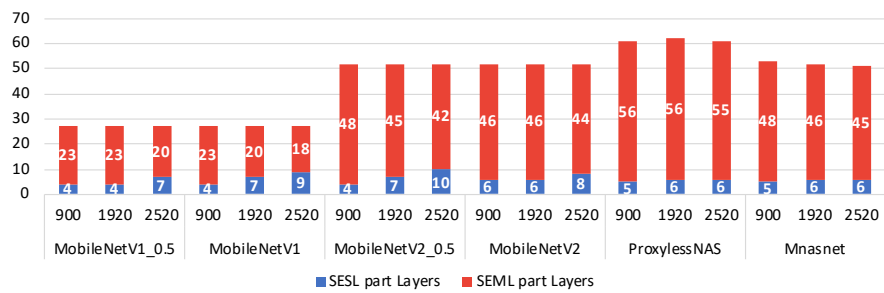


Figure 2.10: Number of layers assigned to each of FiBHA parts. The numbers on the x-axis represent the PEs' budget.

Algorithm 1 SplitCNN

```
1: Input: number_of_PEs, CNN_model, SEML_overhead
2: Output: FiBHA_throughput, FiBHA_PEs_distribution
3: split_layer ← 2
4: SESL_PEs ← []
5: FiBHA_PEs_distribution ← [[], num_of_PEs]
6: FiBHA_throughput ← GetThroughput(number_of_PEs, 0, CNN_model,
                                   SEML_overhead)
7: while split_layer < number_of_layers do
8:     SESL_PEs ← GetBalancedSeslPes(split_layer, CNN_model)
9:     SESL_max_PEs, _ ← GetPeRatios(split_layer, CNN_model)
10:    while SESL_PEs ≤ SESL_max_PEs do
11:        tmp_FiBHA_throughput ← GetThroughput(number_of_PEs,
                                  SESL_PEs, CNN_model, SEML_overhead)
12:        FiBHA_throughput, FiBHA_PEs_distribution ← Maximize(
                                  FiBHA_throughput, tmp_FiBHA_throughput)
13:        SESL_PEs ← AugmentSeslPes(SESL_PEs)
14:    end while
15: end while
```

To optimize FiBHA's performance, a CNN and the PEs budget should be split between the two parts of FiBHA such that: (a) the execution times of the SESL engines are balanced to avoid any within-pipeline idleness (to guarantee assumption 2 in 2.4.3), (b) as the slower of the two parts forms a bottleneck, the maximum of the SESL part execution time and the SEML part execution times should be minimized to maximize the overall throughput[1]. We have designed a heuristic named SplitCNN to split a CNN and the PEs budget in a way that meets these two objectives. The high-level pseudocode of the main part of SplitCNN is shown in Algorithm 1.

$$SESL\_PEs = \frac{1}{GCD(comp(l_x))} \times \sum_{l=0}^{sl} comp(l), l_x \in [0, sl) \tag{2.2}$$

$$SESL\_throughput \propto \frac{1}{max(comp(l)/PEs(l))}, l \in [0, sl) \tag{2.3}$$

$$SEML\_throughput \propto \frac{1}{\sum_{l=sl}^{ll}(1 + Overhead(l)) \times \frac{comp(l)}{PEs(l)}} \tag{2.4}$$

The starting point or the baseline throughput that SplitCNN aims to improve is that of a certain SEML, or a monolithic architecture. This starting point could be any of the SOTA compact model-specific accelerators, or even an implementation of generic DL accelerators like Eyeriss [18] or Systolic Array [175]. SplitCNN takes as input the number of PEs, the CNN model, and the baseline SEML overhead. The CNN model is a data structure that contains, for each CNN layer, the IFMs and OFMs shape, weights shape, strides,

[1]To minimize latency, the sum rather than the maximum of the SESL and SEML part execution times should be minimized.

and layer type (DW/PW/standard Convolution). The SEML overhead is the difference between the theoretical and the practical SEML execution time. The theoretical execution time is the result of equation 2.4 when substituting 0 for the *Overhead*. SplitCNN produces a FiBHA instance description of the best-performing instance given the PEs' budget. The instance description contains the splitting layers and the PEs' distribution on each of the SESL part engines and the SEML part. It also provides the estimated throughput of that instance.

SplitCNN explores the available splitting layer options and tries to find a hybrid architecture that improves over the baseline SEML throughput. Initially, it is assumed that all the PEs are assigned to the SEML part, and the throughput is obtained accordingly using equation 2.4 (line 6). The main loop, starting at line 7, explores the viable splitting options (*split_layer*) starting from the second layer up to the last. Note that "splitting at the last layers" is actually not splitting; it is rather a pure SESL architecture. This means that if the number of PEs is sufficient to implement a pure and perfectly balanced SESL architecture, it could be suggested by SplitCNN.

In each iteration of the main loop (for each *split_layer*), there could be multiple viable PE distributions. Initially, the SESL part is assigned the minimum number of PEs that is needed to form a balanced pipeline of depth *split_layer*; this is obtained at line 8 using equation 2.2 (where *comp(l)* is the operation count in layer l and sl is the *split_layer*). The SEML part is assigned the rest of the PEs. Then the maximum number of PEs that could be assigned to the SESL part is obtained (line 9), which depends on the ratio between the workloads of the two splits. For each PEs distribution between the SESL minimum and maximum, in the inner loop (lines 10-14), the throughput of FiBHA is estimated, and the combination that maximizes it is kept as the best option of *split_layer*. Throughput estimation is described in equations 2.3 and 2.4, where ll is the last layer in the CNN. Figure 2.10 shows examples of the splittings suggested by splitCNN under different PE budgets; these budgets correspond to the number of DSP slices on the used evaluation boards (Table 4.1). Table 2.1 shows the PE distribution for a FiBHA instance example.

Table 2.1: SplitCNN suggested FiBHA architecture for MobileNetsV1_0.5 and 2048 PEs

Split layer	7						
SEML PEs	1693						
SESL PEs	355						
SESL Engines	Engine_0	Engine_1	Engine_2	Engine_3	Engine_4	Engine_5	Engine_6
PEs/Engine	54	18	64	9	64	18	128
MAC/Engine	5419008	1806336	6422528	903168	6422528	1806336	12845056

2.4.5 FiBHA instance generation

To facilitate generating a FiBHA instance given a CNN model and a PEs budget, we developed a set of utilities and an HLS engine repository. Currently, the engine's repository contains HLS implementations of the standard, PW, and DW convolution, fused ReLU-BN-quantization, and pooling layers. These

utilities map a model description written in a high-level framework using the engine's repository to an HLS-based hardware description. The mapping steps are described in Figure 2.11. Our utilities take a CNN model description written in a high-level framework - we support TensorFlow Lite [65] - and extract the weights and metadata. The metadata is fed to SplitCNN, which in turn generates a recommended PE distribution and splitting point. The data generated by SplitCNN and the CNN weights are used to generate the HLS code of the FiBHA instance. This HLS code is fed to the Vitis tool flow to generate the accelerator back-end. All the evaluated FiBHAaccelerators' HLS implementations are generated using this flow. This set of utilities is being developed and extended, aiming to support more CNNs and provide a fully automated end-to-end mapping flow.

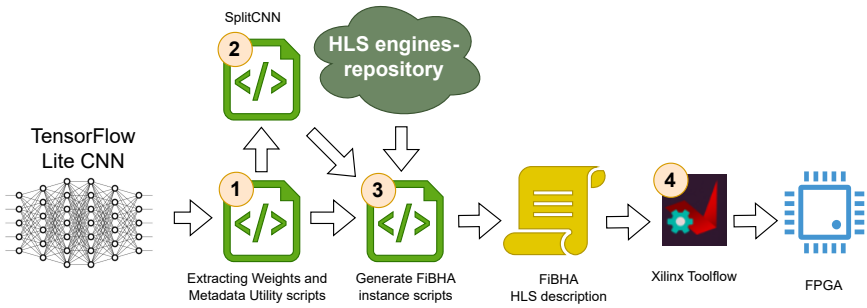


Figure 2.11: FiBHA instance generation flow.

2.5 Experimental Setup

2.5.1 Synthesis Tools & Hardware Platform

We used Vitis-IDE, Vitis-HLS, and Vivado (2021.2) to implement and evaluate our accelerator instances. The evaluation is done using three FPGA boards representing different points of resource budgets. These boards are ZCU102, ZC706, and KCU105, their details are listed in Table 2.2. Moreover, we deployed FiBHA on a physical board, Zynq UltraScale+ MPSoC ZCU102 with an XCZU9EG FPGA and quad-core ARM Cortex-A53. The clock frequency used for FiBHA instances is 180 MHz, as it is the highest supported by most of the designs.

A resource budget determines the combined degrees of parallelism (mainly expressed as loop unrolling in HLS) of all the engines of the accelerator under evaluation. For example, in the FiBHA instance that uses a budget of 1024 PEs, the parallelism in the SESL part engines and the SEML part engines combined is less than or equal to 1024.

Our designs' power is measured with Vivado Power Analyzer using the post-route netlist. This approach is commonly used to produce accurate power dissipation estimations [38, 83]. We have compared the energy efficiency of our designs with two generic NVIDIA GPUs, namely GTX-1660 and RTX-A4000.

Table 2.2: The used evaluation boards. Note that the memory size reported is in megabits.

Resource	ZC706	KCU105	ZCU102
LUT available (K)	218.6	242.4	274.1
LUT max utilization	82%	72%	86%
DSP slices available	900	1920	2520
DSP slices max utilization	89%	68%	83%
Block RAM available (Mb)	19.1	21.1	32.1
Block RAM max utilization	92%	80%	68%

To measure the power consumption of these GPUs, we use the *nvidia-smi* monitoring utility. The frequency is set to the maximum supported for both GPUs.

2.5.2 Baseline Accelerators

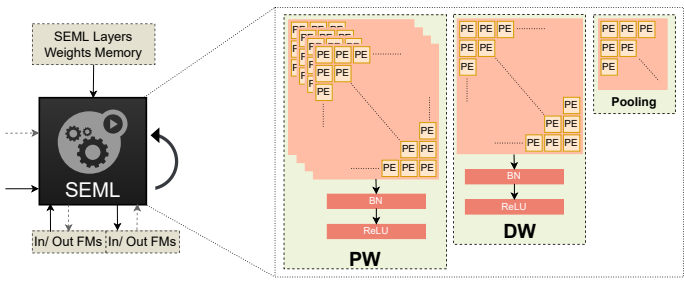


Figure 2.12: B_SEML : The Baseline SEML.

2.5.2.1 B_SEML.

Figure 2.12 shows the main engines of the baseline SEML we used in our implementation and evaluations. It has two separate convolution engines, one for PW convolution and another for DW convolution. It also has a pooling engine, as all the models we evaluate contain a pooling layer. This design is similar to many SOTA compact CNN model-specific accelerators [98, 152, 176]; we refer to this accelerator as *B_SEML*. The accelerator designs that we based B_SEML on were mainly proposed to accelerate MobileNets variants, as they are the most heavily targeted compact CNNs. To support different models and resource budgets, B_SEML supports:

- different DW filter sizes (e.g., 3×3 filters in MobileNets and 3×3 and 5×5 in Mnasnet and ProxylessNAS).
- The PEs are distributed among the engines depending on their workloads. For each engine, the used parallelism on each dimension is set to a factor of the available parallelism (input/output dimensions) to fully utilize the

PEs [105]. When scaling the PEs' budget, the number of PEs in each engine is scaled accordingly as long as the rest of the resources suffice.

B_SEML is used to implement the SEML part of FiBHA (Figure 2.9). Note that the SEML part of FiBHA has the architecture of B_SEML, but usually with a smaller PEs budget since some of the PEs are used to implement the SESL part. We argue that replacing B_SEML by another SEML accelerator will not change the trends of the evaluation. This is because the performance difference between B_SEML and FiBHA comes from capturing more heterogeneity rather than how optimized each of the designs is (Section 2.6.1).

2.5.2.2 FINN.

FINN is an open-source framework that generates a model-specific streaming-dataflow accelerator. In the generated accelerator, each layer has its own dedicated engine [9]. FINN gives the ability to specify the desired throughput and generates an accelerator design that guarantees that throughput as long as the available resources can support it. In our experiments, we report the maximum supported throughput by FINN and the corresponding resource utilization. This means that the PEs' budget, in this case, is the maximum available on the FPGA, and that FINN cannot produce an accelerator with higher throughput using more resources. We used only MobileNetV1 variants in our comparison with FINN as they are the only models in our set with an available FINN end-to-end open-source implementation. It was not possible to generate FINN accelerators for the other CNN models in our experiments. For these models, FINN fails at the streamlining stage, which is one of its transformation stages. More specifically, it fails to convert the addition and scaling operations in these CNNs to *thresholding operations* [9].

2.5.3 Evaluated CNNs

Our evaluation used a representative set of compact, heterogeneous, and hardware-efficient CNNs. They are MobileNetV1, MobileNetV2, ProxylessNAS and MnasNet [11, 57, 142, 159]. CNN descriptions are listed in Table 2.3. We also used MobileNetV1_0.5 and MobileNetV2_0.5, which are lighter variants having a *width multiplier* of 0.5 [57], especially in the cases where the original variants do not fit due to resource constraints. We picked MobileNets as they are the most targeted resource-efficient CNNs in the literature (Section 2.7). We picked ProxylessNAS and MnasNet models as they represent a class of compact CNNs that are designed using NAS algorithms. Inputs of size $224 \times 224 \times 3$ were used (ImageNet input size [33]). We used 8-bit quantization (INT8) for weights and FMs and 32-bit quantization for partial sums. We implemented the quantization scheme used in TensorFlow Lite [65]. Hence, the accuracy is proportional to that of the quantized model provided by TensorFlow Lite. When comparing to FINN, we used 8-bit for the first layer and 4-bit for the rest to match the quantization offered in the FINN-supported open-source models [121].

Table 2.3: Evaluated CNNs' convolutional layers and operation counts.

CNN	Std. Conv Layers	DW Layers	PW Layers	Weights (M)	MAC (B)
MobileNetV1	1	13	13	4.2	0.57
MobileNetV1_0.5	1	13	13	1.3	0.15
MobileNetV2	1	17	34	3.4	0.3
MobileNetV2_0.5	1	17	34	1.9	0.07
ProxylessNAS	1	20	40	4.1	0.27
MnasNet	1	17	34	4.4	0.26

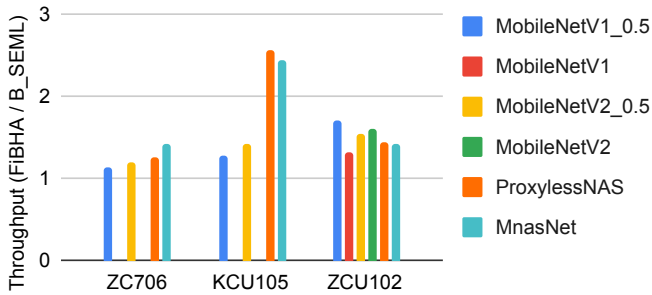
2.6 Evaluation

2.6.1 FiBHA vs. SEML (B_SEML).

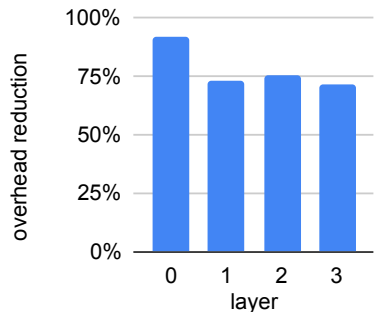
Figure 2.13a shows the throughput of FiBHA compared to B_SEML using various boards and CNN models. For each evaluation board and CNN, we report the throughput of FiBHA normalized to that of B_SEML. The figure shows that there is always an instance of FiBHA that outperforms B_SEML. This demonstrates that the hybrid architecture has an advantage under different resource budgets. The figure depicts three scenarios; we discuss these scenarios in detail in the following paragraphs. The first scenario is where the throughput improvement is between $1x$ and $2x$. The second scenario includes cases where there are missing bars. The third scenario is where the improvements are greater than $2x$.

In the first scenario, the throughput improvements are a result of FiBHA capturing more heterogeneity than B_SEML. Unlike B_SEML, FiBHA processes the initial layers that are heterogeneity-rich (Section 2.4.1) using their own specifically-optimized engines (Sections 2.4.3). Figure 2.13b shows the overhead reduction in the first four layers of MobileNetV2 when implemented, each using its own dedicated engine (FiBHA's SESL part). Where the overhead is the difference between the measured and the ideal execution time of a layer. Similar trends are found in the rest of the evaluated CNNs. We report the results for the first four layers as they are common in all SESL parts of FiBHA in our experiments (Figure 2.10). This reduction in overhead is the highest for the first layer, which is a standard convolution layer. This is because this layer suffers the lowest resource utilization with B_SEML mainly due to its shallow IFMs (parallelism patterns heterogeneity). The first layer forms a bottleneck in other state-of-the-art accelerators as well [17, 113]. The overhead reduction in the rest is a result of, in part, capturing heterogeneity in activation and weight reuse (Section 2.3). Another form of heterogeneity is the variation in the computational load, or the number of MAC operations, across the DW layers. This variation can be categorized as intra-layer-type heterogeneity. Figure 2.13c shows the number of operations in a DW layer for all the DW layers of the evaluated CNNs. The first few DW layers have much larger operation counts than the rest. Since B_SEML design uses a single DW engine to execute all the DW layers, the PEs assigned to that engine must be proportional to the average DW layer operation count. This results in a mismatch between the workload and the engine resources in the initial layers, making them a bottleneck.

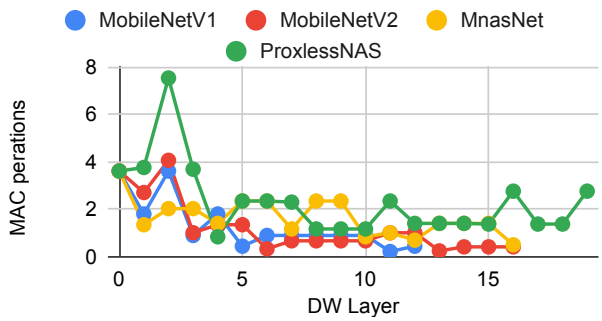
In summary, the considerable heterogeneity among the initial layers explains



(a)



(b)



(c)

Figure 2.13: **(a)** FiBHA throughput normalized to B_SEML throughput, the missing bars represent cases where B_SEML FMs did not fit on-chip. **(b)** Overhead reduction in the first four layers of MobileNetV2 using FiBHA compared to B_SEML on ZCU102. **(c)** Number of MAC operations per DW layer.

why processing them using dedicated engines is crucial. As FiBHA processes these layers using dedicated engines that suit their parallelism patterns and their workloads (e.g., Table 2.1), the inefficiencies are minimized. Note that these dedicated engines are resource-inexpensive (e.g., Table 2.1). This is because their execution time could be relaxed to fill all the time needed by the SEML part to execute the rest of the layers. The double buffering between the two parts of FiBHA (Section 2.4.3) allows the few layers of the SESL to take as much time as the majority of the layers executed by the SEML part take without forming a bottleneck.

In the second scenario, that is MobileNetV1 and MobileNetV2 on ZC706 and KCU105 (figure 2.13a). Only FiBHA instances worked successfully, but there are no results for B_SEML instances because their FMs of a single layer did not fit in the on-chip BRAM (FiBHA reduces the FMs buffer requirements considerably, figure 2.7). This gives FiBHA a qualitative advantage over an SEML-based accelerator. Note that in such scenarios, an SEML-based design would work if the FMs are stored off-chip. However, we do not consider such an implementation since it results in a huge amount of access to the external memory, which would lead to a considerable energy overhead. To give an example, a SEML-based implementation of MobileNetV2 that stores FMs off-chip requires roughly 13.4 million DRAM accesses compared to *zero* when using FiBHA in these scenarios.

In the third scenario, B_SEML , unlike FiBHA, did not scale because the available on-chip BRAM imposed a scaling bottleneck. Hence, FiBHA provides $> 2x$ throughput improvement in this scenario. KCU015 has roughly twice the number of PEs compared to ZC706, but both of them have roughly similar on-chip memory (Table 4.1). When scaling an engine by adding more PEs, more elements need to be accessed in one cycle to keep up with the parallelism and not form a bottleneck. To enable accessing more elements in one cycle, the data has to be partitioned into a sufficient number of independently accessible banks or blocks. This results in some form of fragmentation, where even though the FMs are slightly smaller than the available BRAM, they cannot be stored in a way that satisfies the PEs' scaling requirements. Since FiBHA reduces the required FMs buffering, it increases the free space, which enables partitioning the data without being bottlenecked by the on-chip memory. As a result, FiBHA scaled up to utilize the additional PEs available on KCU015 while B_SEML did not.

2.6.2 FiBHA vs. SESL (FINN)

Figure 2.14a shows the throughput of FiBHA, B_SEML, and FINN measured in frames per second (FPS). Both FiBHA and B_SEML outperform FINN under the same resource budget. FiBHA provides up to $4x$ throughput improvement compared to FINN. For all three approaches, the resource utilization is maximized for the given budget. As such, neither of the three approaches can scale further to achieve higher throughput given this budget. FINN's relatively low throughput is caused by the general SESL scalability issue and by FINN's pipeline balancing technique. We explain these issues in detail in the following paragraphs.

SESL-based accelerators do not scale well in general. Figure 2.14b shows the

maximum compute resource utilization relative to the overall resources available on the used FPGA for both FINN and FiBHA. Note that when comparing with FINN, LUTs are used as the computing resources rather than DSPs; this is because all layers but the first use 4-bit quantization, for which FINN offers a LUT-based optimized implementation. FINN utilizes less than half of the LUTs, which are used as compute resources or PEs. This is because, as with any pipelined SESL accelerator, scaling the performance requires scaling all the engines (Figure 2.3). If only one is not scaled, it forms a bottleneck, forcing the rest to stay idle until it finishes its layer. Moreover, to maintain the utilization and improve the throughput, the engines should be scaled by a common factor of the available parallelism in the layers [9]. Note that in MobileNetV1_0.5 case (Figure 2.14b), the resource utilization is 35%. So ideally, FINN should scale by a factor of 2× and utilize roughly 70% of the resources, but it did not. This is because the on-chip memory forms a bottleneck. Figure 2.14c shows the BRAM efficiency reported by FINN, which is the ratio of the actually used memory to the allocated BRAM blocks. BRAM efficiency drops when scaling up as a result of fragmentation similar to SEML (Section 2.6.1). This is because when an engine is scaled up, it needs to access more weights and FMs elements per cycle to keep its PEs busy. Since each BRAM has a limited number of ports, accessing more elements per cycle requires partitioning these elements across more BRAMs to scale the bandwidth. This partitioning reduces BRAM efficiency. With more scaling, the result is using more BRAM blocks with fewer elements in each block, which at some point becomes a scaling bottleneck. But unlike SEML, where the reason for fragmentation is storing the large FMs of the initial layers on-chip, in FINN, the reason is storing the large weights of the last layers on-chip.

The FINN scaling algorithm starts by setting the parallelism to 1 for all engines; they refer to this step as a minimal dataflow compute implementation. As the name suggests, this is the minimum number of PEs that could be used. After that initial assignment, and based on the compute requirements of all the layers, the FINN scaling algorithm systematically gives more resources to the most pressing bottlenecks. When giving more resources to these bottlenecks, which are the engines with the highest latencies, other engines become the new bottlenecks. Hence, this process is repeated by identifying the new bottleneck engines and giving them more resources until the available resources are exhausted. Even though this gives the best balance for the available PEs budget, the achieved balance could be far from optimal. This is because an optimal balance requires that all the engines have equivalent ratios between their layer's computational workloads and the processing elements that they are assigned. Figure 2.14d depicts the impact of FINN assignment on idleness, which is a direct result of not having an optimal balance among the engines. The x-axis shows time, and the y-axis shows weighted idleness, which is the product of the idleness of an engine (the difference between its execution time and the slowest engine execution time), and the ratio of that engine's resources to the overall used resources. The figure shows that up to $\approx 60\% - 75\%$ of the resources are idle for more than 10% of the time. Note that this ratio is out of FINN's utilized resources, not the overall resources.

FiBHA does not suffer from any of these issues; FiBHA's pipeline is much shorter than a pure SESL (e.g., FINN), and this has two advantages. First,

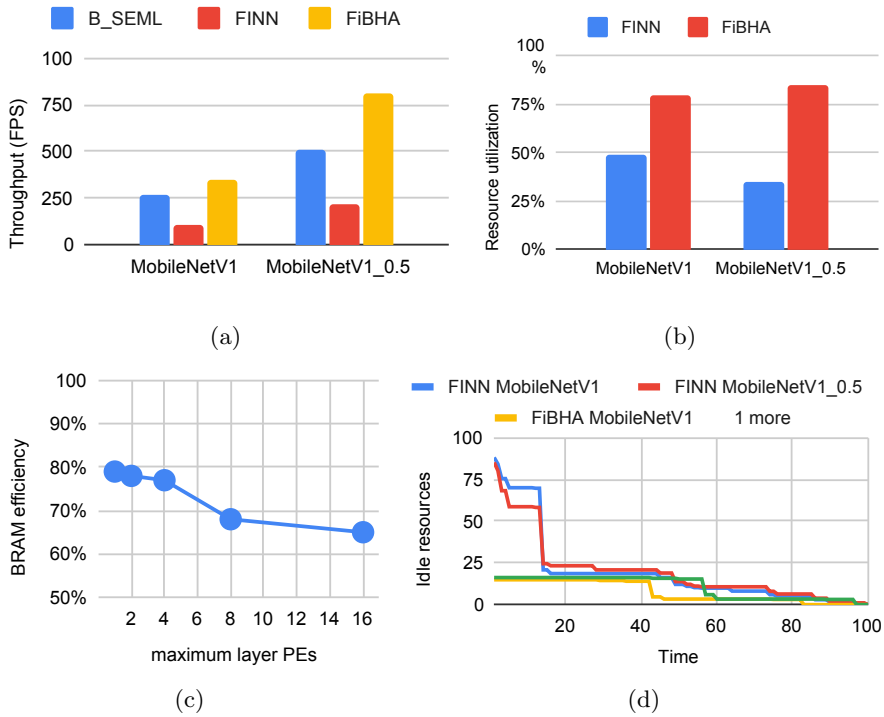


Figure 2.14: **(a)** B_SEML, FINN and FiBHA throughput normalized to B_SEML throughput. **(b)** Maximum LUT utilization achieved by FiBHA and FINN on ZCU102. **(c)** FINN BRAM efficiency, the ratio of the actually used to the allocated BRAM blocks. **(d)** FINN vs. FiBHA weighted compute resource idleness

scaling FiBHA's SESL part is much less expensive. Secondly, FiBHA's SESL engines are easier to balance, hence it experiences lower resource idleness than a pure SESL.

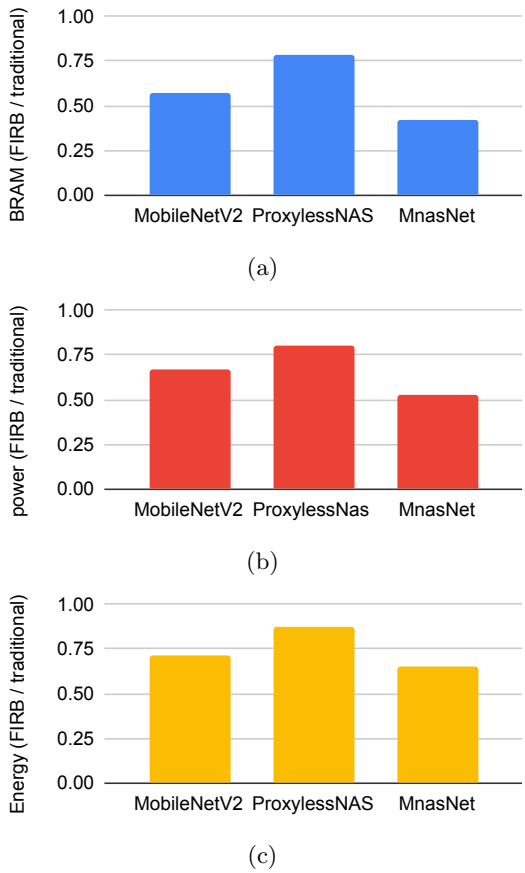


Figure 2.15: **(a)** Required on-chip buffers of FIRB normalized to a traditional tiled inter-layer pipeline. **(b)** Power consumption of FIRB buffers normalized to a traditional tiled inter-layer pipeline. **(c)** Energy consumption per inference of FIRB normalized to a traditional tiled inter-layer pipeline.

2.6.3 FIRB v.s. traditional layer-layer pipelining

Figure 2.15 quantifies the advantage of using FIRB pipelining compared to traditional inter-layer pipelining to implement the SESL part of FiBHA. Figure 2.15a shows the amount of on-chip buffers (BRAM) required by FIRB compared to the traditional pipelining. FIRB saves up to 55% of the SESL buffering. This reduction is a result of the fine-grained pipelining inside the inverted bottlenecks (Section 2.4.2). Figure 2.15a shows the power consumption of the BRAM required by FIRB compared to the traditional pipelining. It can be seen that the power reduction is almost proportional to the reduction in the needed BRAM count. ProxylessNas has the lowest memory and power reduction; this is due to the fact that its initial inverted bottlenecks have a smaller expansion

Table 2.4: Energy efficiency of prior art accelerators and FiBHA targeting heterogeneous CNNs (lower is better). M_v1 stands for MobileNetV1, M_v2 for MobileNetV2, and Prox for ProxylessNAS.

Paper	Platform	quantization (bit-width, type)	energy efficiency (joules / inference)			
			M_v1	M_v2	Prox	MnasNet
[186]	XC7V690t	8, integer	-	**0.113**	-	-
[4]	XCZU7EG	channel-wise, integer	-	0.125	-	-
[173]	Arria10 GX1150	-	0.79	-	-	-
[98]	XC7Z045	16, fixed-point	18.4	-	-	-
[147]	XCZU7EV	8, integer	-	0.514	-	-
[181]	XC7Z020	16, fixed-point	-	0.198	-	-
[124]	XCZU7EV	dynamic, fixed-point	-	4.85	-	-
[133]	XC7V690T	16-bit fixed-point	1.23	-	-	-
[200]	XC7VX690T	-	-	-	-	0.73
FiBHA	XCZU7EG	8, integer	**0.179**	0.177	**0.168**	**0.16**

factor (Sections 2.2, 2.4.2) than the others. This means that the buffers inside the bottlenecks, which are eliminated by FIRB, are small compared to the other models. Figure 2.15c shows the energy consumption of FIRB compared to traditional pipelining. FIRB constantly improves energy consumption due to power savings.

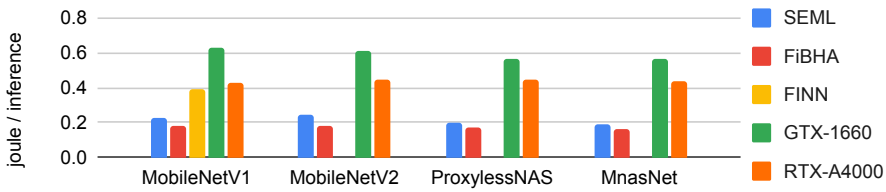


Figure 2.16: Joules per inference of FiBHA , B_SEML , GTX-1660 and RTX-A4000. FiBHA and B_SEML measurement are of ZCU012.

2.6.4 Energy efficiency

Figure 2.16 shows the energy consumption of FiBHA, B_SEML, FINN, and two GPUs (NVIDIA GTX-1660 and RTX-A4000) using four heterogeneous CNNs. Note that in our evaluation, we use a batch size of 1, as is usually done when evaluating inference. For latency-critical applications, e.g., inference in vision applications for autonomous driving, a batch size of 1 could be the only practical option. However, for applications that are not latency-critical, GPUs could exploit larger batch sizes to achieve better performance.

FiBHA constantly improves the energy efficiency over B_SEML. This reduction is caused by capturing heterogeneity, which is translated to better resource utilization (Section 2.6.1). In addition to that, FiBHA reduces the FMs buffer sizes considerably. Smaller buffers contain fewer BRAM blocks, leading to power savings (Section 2.6.3). Both FiBHA and B_SEML outperform FINN; FINN consumes more energy per inference due to its relatively high execution time (Section 2.6.2). The three dedicated accelerators have better energy efficiency than the GPUs. The RTX-A4000 is more energy-efficient

than the GTX-1660. This can be explained by the fact that RTX-A4000 has third-generation tensor cores while GTX-1660 does not have any [119].

MobileNetV2, ProxylessNAS, and MnasNet have similar energy consumption levels. This is expected as they have similar operation counts (Table 2.3), and the same building block (the Inverted Residual Bottleneck). Even though MobileNetV1 has roughly double the number of operations compared to the rest, it has similar energy consumption. However, note that MobileNetV2, ProxylessNAS, and Mnasnet have $2\% - 5\%$ higher accuracy compared with MobileNetV1 [11, 142, 159]. Meaning that they offer higher accuracy without increasing energy consumption.

Table 2.4 shows the energy efficiency of several accelerators targeting compact and heterogeneous CNNs. Many of these accelerators are highly dedicated and target one model, which is often a variant of MobileNets. Different approaches use different optimizations, different quantization levels, and are deployed on different boards. Hence, this table is meant to show efficiency in the context of prior art rather than presenting a direct comparison. Note that FiBHA , as we discuss in sections 2.4.3, 2.4.4, is not a specific implementation, and that any state-of-the-art SEML accelerator could be used as the SEML part of FiBHA. For example, [186] is likely to outperform our FiBHA implementation of MobileNetV2 using the same board, as it is designed specifically and highly optimized only for MobileNetV2. But [186] could then be used instead of B_SEML to generate a new and more optimized FiBHA instance targeting MobileNetV2 following the steps described in Section 2.4.4. [4] also outperforms FiBHA using the same board, since it uses channel-wise quantization with extremely low bit-widths. Note that [4] could be fed to SplitCNN 2.4.4 and be used to derive a more optimized FiBHA instance.

2.7 Related work

To improve the resource efficiency of DNN processing, at both training and inference phases, researchers have been proposing model-specific accelerators. FPGAs are heavily used in developing such custom accelerators owing to their reconfigurability [9, 48, 97, 107, 134, 135, 152, 163].

MobileNets are the most commonly targeted resource-efficient CNNs by model-specific SEML accelerators. This is because the modules or the building blocks of MobileNets have been used to design most of the state-of-the-art efficient models [11, 159, 160, 196]. Bai *et al.* [5] proposed an accelerator that processes all the CNN layers using a single engine. But they designed a configurable adder tree that supports both DW and PW convolutions. In [98, 100, 152, 173, 176], the authors proposed to use two separate convolution engines, one for PW and another for DW. These two engines are pipelined to maintain high throughput. Su *et al.* [152] also utilized two separate engines to process the convolutional layers. Moreover, they optimized the model using structured kernel-level pruning and quantization to further reduce the model parameter count, producing Redundancy-Reduced MobileNet (RR-MobileNet). In RR-MobileNet, the model parameters and intermediate results are stored on-chip. Wu *et al.* [176] proposed a channel augmentation technique that improves the efficiency of the first layer of MobileNets. Yan *et al.* [186] optimized the

bottleneck module, and they proposed dedicated hardware to reduce the module data transmission latency.

SESL accelerators were proposed to capture CNN heterogeneity to the fullest [6, 9, 35, 97, 137, 163, 166]. In an SESL architecture, all the model layers are mapped to the hardware, each layer having its own dedicated engine. These engines are pipelined and work simultaneously. Synchronous Dataflow (SDF) model of computation [93] is usually used to map CNNs to the hardware [163, 166]. To reduce the latency and buffering overheads among the pipelined engines, Gao *et al.* [43] have proposed an alternate layer loop ordering (ALLO) dataflow that enables forwarding the intermediate results in a relatively timely manner between pairs of adjacent layers.

Pure SESL accelerators, where all the layers are mapped to dedicated engines, are resource-demanding and unscalable [107, 135]. One FPGA-based workaround is based on partitioning the CNN model along its depth and mapping each partition to a bitstream [166]. This incurs a considerable overhead, which grows as the models get deeper. Another proposal is to use aggressively quantized, even binarized CNNs [9,163], where fewer bits are needed to represent the weights, the activations, and the partial sums. As a result, computations are performed using simpler hardware, and the bandwidth requirements are reduced. However, the accuracy drop caused by aggressive quantization may not be acceptable. Another bottleneck of a pure SESL accelerator is the need to feed weights and inputs to all the simultaneously running engines, which increases the bandwidth requirements. Li *et al.* [97] proposed a mechanism to alleviate the bandwidth bottleneck that helps when a CNN is shallow, e.g., AlexNet [80].

Unlike the related work, FiBHA's hybrid design captures as much heterogeneity as the PEs budget allows without sacrificing resource efficiency [127]. Hence, it achieves better scalability and higher throughput compared to the alternatives. In this extension, we propose, implement, and evaluate FIRB, a fine-grained and memory-light SESL building block. We analyze and demonstrate the memory efficiency and scalability of FiBHA using multiple FPGA boards. Finally, we demonstrate FiBHA's energy efficiency compared to both custom accelerators and GPUs.

2.8 Conclusion and future work

In this chapter, we presented FiBHA, a hybrid architecture that captures CNN heterogeneity within a fixed resource budget. We presented FIRB, a memory-light and fine-grained module that improves the efficiency of pipelining the Inverted Residual Bottlenecks that are common in recent compact CNNs. We implemented instances of FiBHA and demonstrated its advantages over the two commonly used alternative custom and model-aware accelerator families (SEML and SESL) using different FPGAs representing different resource budgets. Our experiments demonstrated that capturing CNN heterogeneity translates to better resource utilization and higher throughput. For a fixed hardware budget, FiBHA achieved up to $2.5x$ and $4x$ of the throughput achieved by the state-of-the-art accelerators of the two categories. Moreover, FIRB modules reduce the required memory by up to 54%, and energy requirements by up to 35%

compared to traditional inter-layer pipelining. Finally, we compare and analyze the energy efficiency when processing such CNNs using FiBHA , a SEML-based accelerator, and generic GPUs.

We plan to extend our cost model and heuristics by designing multi-objective optimizations that address the trade-offs between performance, memory requirements, and energy efficiency. In our future work, we plan to extend the targeted models from compact to general CNNs.

Chapter 3

An Analytical Cost Model for Fast Evaluation of Model-Specific Multi-Engine CNN Accelerators

This chapter presents MCCM, an analytical cost model and evaluation methodology for multi-engine (or multiple-CE) CNN accelerators. The proposed methodology can be extended to evaluate any model-specific FPGA-based DL accelerator. Compared to traditional synthesis-based evaluation, MCCM provides significantly faster evaluation while maintaining high accuracy. As a result, it can serve as a basis for efficient exploration of the multi-engine accelerator design space. This chapter focuses on the modeling aspect, presenting a simple DSE case, and the next chapter (Chapter 4) presents a design methodology that uses MCCM as a basis for DSE.

3.1 Introduction

The use of Convolutional Neural Networks (CNNs) in various application domains and across a spectrum of environments, ranging from edge to cloud, brings a wide range of performance requirements and optimization goals [22,168]. While some applications are latency-critical, others target high throughput. Moreover, resource efficiency is crucial when having limited resources [23, 178], and when resources are shared by multiple applications [169]. FPGAs' reconfigurability enables the design of accelerators tailored to such diverse

This chapter is based on the following publications:
F. Qararyah, M. A. Maleki, and P. Trancoso, "An Analytical Cost Model for Fast Evaluation of Multiple Compute-Engine CNN Accelerators," in 2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2025, pp. 1–13.

requirements, which makes FPGA-based CNN accelerators popular [9, 23, 40, 176, 178, 197].

Accelerators that organize FPGA resources into generic reusable Compute Engines (CEs) have limited adaptability to the varying characteristics of CNN layers, leading to dynamic resource underutilization [14, 127, 128, 174, 199]. This limitation is often described as one size does not fit all. Multiple Compute-Engine (*multiple-CE*) accelerators overcome this limitation. They organize FPGA resources into a number of dedicated CEs tailored to the structure of the CNN model and the available resources [2, 14, 117, 128, 149, 150, 167, 174, 199]. However, the existing multiple-CE accelerators differ in how they arrange their CEs and distribute the FPGA resources and CNN layers among the CEs. Identifying CE arrangements that achieve the best performance and efficiency requires a systematic approach to multiple-CE accelerator design.

A systematic accelerator design approach requires analyzing the *bottlenecks* and inefficiencies and *exploring the design space* to identify optimizations that mitigate them. The literature reveals that the inefficiencies of Deep Learning (DL) accelerators, in general, have three root causes, **(1)** *Processing Element (PE) underutilization* [10, 106, 195], **(2)** *large on-chip buffers* [10, 125, 191], **(3)** and the time and energy costly *off-chip access* [106, 195]. Considering multiple-CE accelerators, the second and third causes are still open challenges. However, multiple-CE accelerators have less PE underutilization than generic ones. Multiple-CE accelerators mitigate the PE underutilization bottleneck but in a way that creates *throughput and latency* trade-off [14, 128, 174]. Hence, a systematic multiple-CE accelerator design approach requires analyzing PE underutilization with latency-throughput trade-offs, on-chip buffer requirements, and off-chip access.

The large design space of CE arrangement possibilities and the lack of a fast methodology that models and evaluates their impact on multiple-CE accelerators' performance and efficiency make a systematic design approach impractical. While High-Level Synthesis (HLS) shortens the design time, synthesizing a single accelerator instance can take hours. To overcome this challenge, this work proposes a Multiple-CE accelerator analytical Cost Model (MCCM), and an evaluation methodology built around it. The proposed model and methodology enable the expression of any multiple-CE accelerator using simple notation and provide a fast and accurate evaluation of its throughput, latency, on-chip buffer requirements, and off-chip accesses. This work makes the following contributions:

- It proposes a multiple-CE accelerator analytical cost model (MCCM). MCCM applies bottom-up modeling of the impact of CE arrangement possibilities on the performance and efficiency of multiple-CE accelerators.

- It proposes an MCCM-based evaluation methodology, which enables expressing any multiple-CE accelerator and evaluates its latency, throughput, on-chip buffer requirements, and off-chip access orders of magnitude faster than the traditional approach.

- It presents three use cases of MCCM. First, an end-to-end evaluation of state-of-the-art multiple-CE accelerators considering different metrics, CNNs, and resource budgets. Second, fine-grained evaluation of

multiple-CE accelerators' performance bottlenecks. Third, a design space exploration of multiple-CE accelerators enabled by MCCM fast evaluation.

MCCM is in the order of 100000× faster than synthesis-based evaluation and has an average accuracy of > 90%. The presented MCCM use cases using three state-of-the-art multiple-CE accelerators, five CNNs, and four FPGA boards show that no single multiple-CE architecture is the best, given different metrics, CNN models, and resource budgets. Moreover, the evaluation shows the impact of engine arrangements on multiple-CE accelerators' performance bottlenecks. Finally, using MCCM fast evaluation as a basis for multiple-CE accelerators' design space exploration enables identifying designs that outperform the state-of-the-art.

3.2 Background and Motivation

3.2.1 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are feed-forward Deep learning (DL) algorithms [91]. A CNN comprises a sequence of *layers* that perform feature extraction and classification [92]. Convolutional layers are the primary layers in a CNN. A convolutional layer consists of a collection of trainable parameters, known as *weights*, which are organized into *filters*. These filters extract features from multi-dimensional inputs or intermediate results. The inputs of a layer are referred to as Input Feature Maps (*IFMs*), and the outputs as Output Feature Maps (*OFMs*). The term Feature Maps (*FMs*), or activations, refers to both IFMs and OFMs. FMs consist of 2D slices known as *channels*.

3.2.2 CNN Compute-Engines

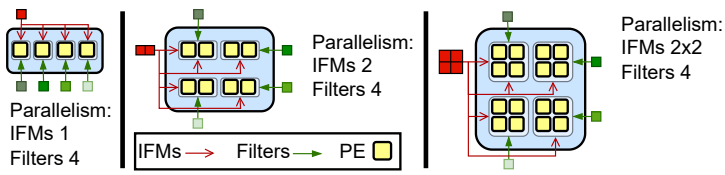


Figure 3.1: Compute-Engine (CE) parallelism examples

As convolutions represent more than 90% of modern CNN operations [106], CNN Compute-Engine (CE) optimizations focus on convolutional layers. A CE comprises a grid of Processing Elements (PEs) where each PE performs a MAC operation, e.g., a DSP in an FPGA. Two main design decisions that affect the performance and efficiency of a CE are *parallelism strategy* and *dataflow*. A convolution operation comprises a loop nest of six loops without batching [74]. A CE parallelism strategy describes which among these six loops are partially or fully parallelized. The CE parallelism strategy affects data reuse and memory access patterns [19,106]. Figure 3.1 depicts examples of parallelism strategies in 3-D, 2-D, and 1-D. An exhaustive analysis of parallelism strategies on FPGAs

shows that parallelizing on three dimensions, namely across filters and within an IFM channel width and height, yields the best average reuse and CE utilization across the layers of common CNNs [106]. However, 2-D and 1-D parallelism are also common when each CE has a limited PE budget or when 1-D or 2-D options are more compatible with the dimensions of the CNN layers processed by that CE [2, 107, 150, 166, 194, 197, 199]. In the context of convolution CEs, the term dataflow refers to both convolution loop transformations, *i.e.* the scheduling of the loop computations, and the mapping of computations across the CE PEs. The most common dataflow types are *weight-stationary*, *output-stationary*, and *input-stationary*. Each dataflow indicates which of the IFMs, OFMs, or weights is scheduled to move least frequently [84].

3.2.3 Multiple compute-engine CNN accelerators

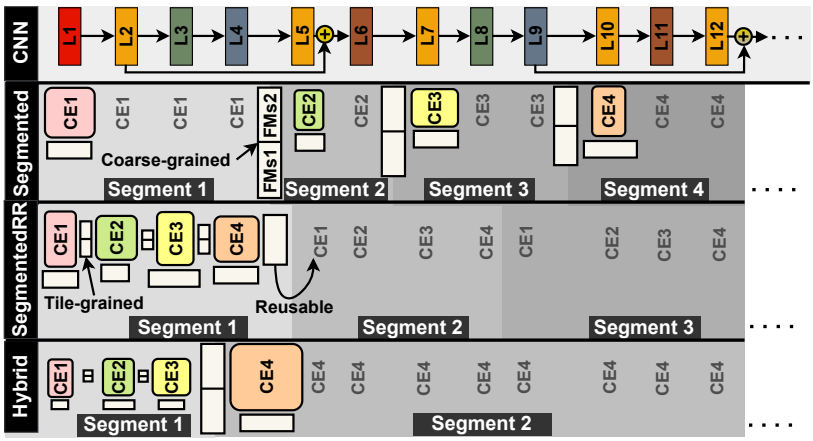


Figure 3.2: CNN to multiple-CE architecture mapping examples. For example, in Segmented, CE1 processes layers L1-L4, CE2 processes layers L5 and L6, and so on. In practice, CE and buffer sizes are proportional to the segment layers' compute and memory requirements, respectively.

We use the term multiple Compute-Engine (*multiple-CE*) accelerators to refer to accelerators that organize the FPGA resources into more than one convolution CE. The extreme case of multiple-CE architecture is to have a CE per CNN layer tailored to that layer's characteristics. This approach is resource-demanding and not scalable [14, 128, 199]. To be both CNN model-aware and scalable, state-of-the-art multiple-CE accelerators adjust the number of CEs based on the CNN characteristics and the available FPGA resources. Exploring the literature shows that, apart from the mentioned extreme case, multiple-CE accelerators follow one of three architectural patterns. Figure 3.2 shows a high-level representation of these architectures using four CEs. In practice, the number of CEs varies depending on the CNN structure and the available resources. Each of these CEs may have a unique number of PEs, parallelism strategy, and dataflow (Section 3.2.2).

We refer to the first architecture in Figure 3.2 as **Segmented** [149, 150]. In this architecture, each CE processes one or more *segments*. Each segment

Table 3.1: Comparison of multiple-CE accelerators using ResNet50 on AMD Zynq 7000 SoC ZCU102. The values for a metric are normalized to the best in that metric.

	latency	on-chip buffers	off-chip accesses
SegmentedRR	1.0	2.64	1.79
Segmented	4.7	1.0	1.99
Hybrid (b)	1.11	1.74	1.0

consists of a set of consecutive layers. Each CE has its own on-chip buffer that stores the weights and FMs partially or fully, depending on the available on-chip memory. *Coarse-grained pipelining*, where different CEs process different inputs (images) at a certain time step, is applied between the CEs. CE inputs are double-buffered to enable pipelining. We call the second architecture **SegmentedRR** (round-robin) as CEs in such an architecture usually process CNN layers circularly according to their topological order [9, 167, 174]. While some apply coarse-grained pipelining between CEs, Wei *et. al* [174] have shown that, for this architecture, *tile-grained pipelining* is more efficient. In addition to the double-buffering between the CEs, each CE has a separate buffer for the weights. We call the third architecture **Hybrid** [2, 117, 128, 199]. It comprises two parts. The first contains a set of tile-grained pipelined CEs, each processing a single layer. The second has a larger CE that processes the rest of the layers. If a CNN has two types of convolutional layers, the second part could have two sub-CEs [128]. Coarse-grained pipelining is applied between the Hybrid's first and second parts.

3.2.4 Multiple-CE accelerator modeling and evaluation

State-of-the-art multiple-CE accelerators adopt one variant of the three mentioned architectures and focus on optimizing over two extremes, namely reusable-CEs-based accelerators [5, 106, 190] and accelerators with as many CEs as CNN layers [163, 197]. Adopting fixed architectures limits the potential of existing multiple-CE accelerators. No single architecture is optimal when considering different metrics. Table 3.1, which reports normalized synthesis results of instances of the three architectures, exemplifies that. For example, the SegmentedRR at the first row has the best latency but requires 2.64× the on-chip buffer and 1.79× the off-chip accesses of the best accelerator in each metric. Note that buffer requirements and off-chip memory accesses are treated here as optimization goals. This is because minimizing off-chip accesses, consequently bandwidth requirements, and on-chip buffer requirements are crucial, especially when resources are shared by multiple applications [169], even when neither of them forms a bottleneck. The fact that no single architecture gives the best results considering different metrics, even for the same CNN and FPGA as the table shows, suggests that a fixed architecture is far from optimal. Hence, a more systematic design approach that explores different CE arrangement possibilities is needed to identify the best designs given an application performance metric, CNN model, and resource budget. However, a traditional evaluation of one CE arrangement can take hours. This makes exploring the numerous CE arrangement possibilities of multiple-CE accelerators, and consequently, a

systematic design approach, impractical. This motivates the proposal of a fast evaluation methodology for multiple-CE accelerators.

3.3 Multiple-CE accelerator evaluation methodology

3.3.1 Overview of the evaluation methodology

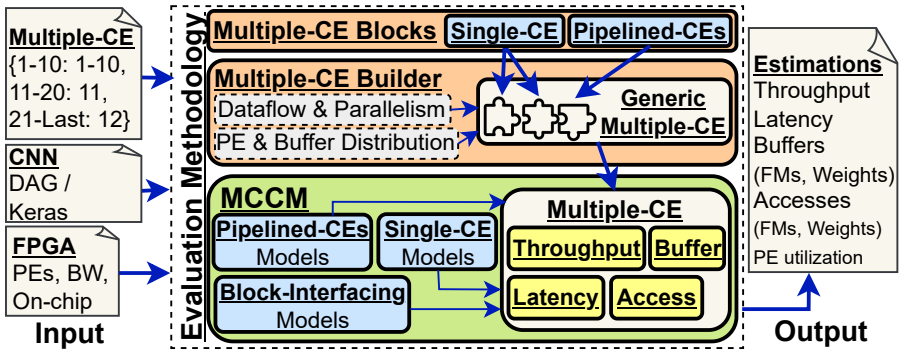


Figure 3.3: Overview of multiple-CE evaluation methodology.

Figure 3.3 presents an overview of the proposed multiple-CE accelerator evaluation methodology. The methodology takes as **inputs**: **(1)** a multiple-CE accelerator description, discussed in Section 3.3.2, **(2)** a CNN representation, **(3)** the FPGA's number of PEs (DSPs), off-chip memory bandwidth, and on-chip memory capacity. The main **outputs** are throughput, latency, on-chip buffer requirements, and off-chip accesses. Other outputs include a fine-grained analysis of PE utilization and a breakdown of the results on the level of weights and FMs. The evaluation methodology comprises three modules. The **Multiple-CE Blocks** module contains codified descriptions of two building blocks that could be used to construct any multiple-CE accelerator, namely *single-CE* and *pipelined-CEs*. A building block comprises one or more CEs. The **Multiple-CE Builder** module constructs a *generic multiple-CE accelerator* backbone by combining the building blocks, considering the three methodology inputs. It then decides the implementation details, including the CEs buffer and PE distribution, dataflows, and parallelism strategies based on a set of heuristics inspired by the prior art [9, 106, 128, 150, 174]. The **MCCM** module evaluates a generic multiple-CE accelerator bottom-up by modeling its building blocks and their interfaces. MCCM estimates multiple-CE accelerator throughput, latency, on-chip buffers, and memory access requirements. MCCM is the main contribution of this work; hence, we use the terms MCCM and evaluation methodology interchangeably. MCCM components are discussed in detail throughout Section 3.4. As DL accelerator design is a hot topic where new accelerators are frequently proposed, we designed MCCM to be modular and extensible. MCCM interface allows easy addition of models of newly proposed

specialized designs.[1]

3.3.2 Expressing a generic multiple-CE accelerator

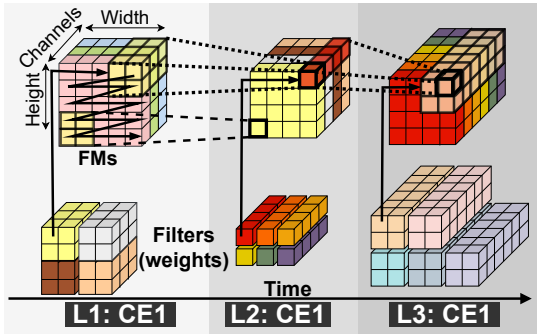
Examining existing multiple-CE accelerators, shown in Figure 3.2, shows that they comprise two basic building blocks. The first is a *single-CE processing a range of layers* one by one. For example, CE_1 processes layers $L_1 - L_4$ in the Segmented, and CE_4 processes L_4 to the last layer in the Hybrid. The second building block is a set of *pipelined-CEs, each processing a layer*. For example, $CE_1 - CE_4$ process layers $L_1 - L_4$ in SegmentedRR, and $CE_1 - CE_3$ process layers $L_1 - L_3$ in Hybrid. These two blocks could be used to express any multiple-CE architectures. We propose the following notation to express a multiple-CE architecture:

- CE_x: denotes a single-CE block, and $CE_x - CEy$: denotes $(y - x) + 1$ pipelined-CEs block.
- $\{L_x - L_y : CE_z\}$: denotes that layers x to y are processed using a single-CE block (CE_z) sequentially. A special case is having one layer only, namely $\{L_x : CE_z\}$.
- $\{L_x - L_y : CE_z - CE_w\}$: denotes that layers x to y are processed using pipelined-CEs block. If the number of layers exceeds the number of CEs, the pipelined-CEs block is assumed to process $(w - z) + 1$ layers at a time.

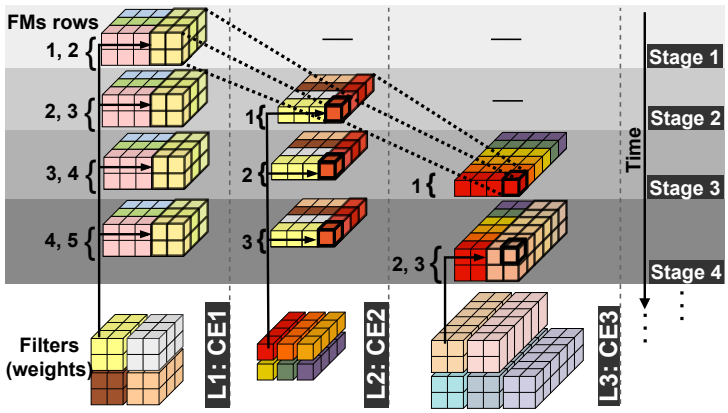
For example, the Segmented accelerator in Figure 3.2 is expressed as $\{L1 - L4 : CE1,\ L5 - L6 : CE2,\ L7 - L9 : CE3,\ L10 - L12 : CE4, ...\}$, and the SegmentedRR can be expressed as $\{L1 - Last : CE1 - CE4\}$. Multiple-CE Builder transforms this notation, given the CNN and FPGA descriptions, into a generic multiple-CE accelerator representation that is fed into the analytical cost model.

3.4 Multiple-CE accelerator analytical cost model

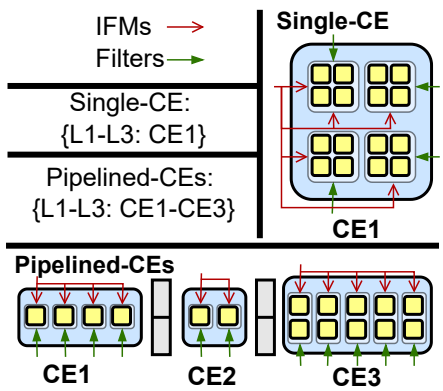
The Multiple-CE accelerator analytical Cost Model (MCCM) models a generic multiple-CE accelerator using a bottom-up approach. It evaluates the accelerator using the models of the single-CE and pipelined-CEs blocks and the interfaces between them. The block modeling is discussed in Section 3.4.1. Modeling a whole accelerator depends on **(1)** whether each of its blocks processes multiple segments (like the pipelined-CEs block in the SegmentedRR in Figure 3.2) or one segment (like single-CEs blocks in the Segmented in Figure 3.2) and **(2)** whether there is inter-segment pipelining (like in the Segmented in Figure 3.2) or not. The details of such dependencies are discussed in Section 3.4.2.



(a) Single-CE sequential processing



(b) Pipelined-CEs concurrent processing



(c) 16 PEs distribution examples

Figure 3.4: (a) Sequential (layer by layer) processing, (b) pipelined processing, and (c) Single-CE and Pipelined-CEs blocks.

3.4.1 Modeling multiple-CE accelerator building blocks

This section presents the modeling of the single and pipelined-CEs blocks. Figure 3.4a and 3.4b illustrate the differences between processing three convolutional layers using the two blocks. Figure 3.4a depicts the single-CE case. In this case, each layer is processed to completion before moving to the next. Figure 3.4b depicts the pipelined-CEs case. In this case, the layers are processed concurrently, stage by stage, at tile granularity. The figure shows four out of the six stages.

3.4.1.1 Latency and throughput

In the *single-CE* case, the latency of processing a set of layers is the sum of their latencies, and the throughput is its inverse.

For simplicity, we assume that all layers are compute-bound and that the memory access time is completely hidden by computation time throughout the discussion. In practice, however, we do consider memory access time. Equation 3.1 describes latency estimation considering PE underutilization, where $layers$ is the number of layers processed by the single-CE, DD is the disjoint dimensions of the filters and FMs, *i.e.* the six dimensions corresponding to the six convolution loops (Section 3.2.2), and $Par(CE_j, d)$ is the parallelism on a dimension d. The parallelism across all dimensions is upper-bounded by the number of PEs of a CE. To explain the PE underutilization, consider the single-CE shown in Figure 3.4c with the parallelism of 16 ($4 \times 2 \times 2$). This CE processes weights from 4 filters and a 2×2 IFM tile in a cycle. When this CE is used to process L2 from Figure 3.4a, which has 6 filters, its PEs are fully utilized when processing the first 4 filters and half utilized when processing the remaining 2. The parallelism dimensions could be configured differently to avoid underutilization in L2, but such arrangements would result in underutilization in the other layers. Generally, the more diverse the layers processed by a single-CE are, the harder it is to avoid PE underutilization.

$$
\begin{aligned}
Latency &= \sum_{i=1}^{layers} Lat(L_i, CE_j) \\
Lat(L_i, CE_j) &= \prod_{d}^{DD_i} \lceil |d| / Par(CE_j, d) \rceil \\
where &\prod_{d}^{DD_i} Par(CE_j, d) \leq PEs(CE_j)
\end{aligned}
\tag{3.1}
$$

The literature defines the latency of *pipelined-CEs* in two ways. The first is the time to process a single input, e.g., an image. The second is the time to process a batch of inputs divided by the number of inputs. Here, we adopt the first definition because batching is not always an option and because the first definition is the one that matters in latency-critical applications. Equation 3.2 shows latency estimation of pipelined-CEs, where $PipeStages$ is the number of stages to complete processing an input, CEs is the number of CEs in the pipeline,

[1]The details on how to extend MCCM are provided in the repository: https://github.com/fqararyah/MCCM.

$FMsTile_{ij}$ is the FMs tile processed at stage i by CE_j, and $activeCEs(stage_i)$ are CEs active at stage i. Figure 3.4b visualizes stages and CE activity. The interpretation of the equation is that the latency of processing a set of layers is the summation of the pipeline stage latencies, and the stage latency depends on the slowest active CE in that stage.

$$\begin{aligned} Latency &= \sum_{i=1}^{PipeStages} Lat(stage_i) \\ Lat(stage_i) &= \max_{j=1}^{CEs} \Big(Lat(FMsTile_{ij}, CE_j) \Big) \\ & \quad where\ CE_j \in activeCEs(stage_i) \end{aligned} \tag{3.2}$$

Equation 3.3 depicts pipelined-CEs throughput estimation where CE_Stages_i denotes the stages in which CE_i is active. Pipeline throughput is the inverse of the slowest CE latency [197]. To maximize the throughput of pipelined-CEs, the slowest CE's latency must be minimized. This is done by balancing the pipeline stages, *i.e.* assigning PEs to each CE proportional to its relative workload [9, 197].

$$\begin{aligned} Throughput &= \frac{1}{\max_{i=1}^{CEs} \Big(Lat(L_i, CE_i) \Big)} \\ Lat(L_i, CE_i) &= \sum_{j=1}^{CE\_Stages_i} Lat(FMsTile_{ij}, CE_i) \end{aligned} \tag{3.3}$$

The pipelined-CEs example in Figure 3.4c demonstrates how distributing the PEs on multiple CEs could solve the PE underutilization. Each CE in the figure has a parallelism that perfectly divides the corresponding layer filters and FM dimensions. As a result, the pipelined design has an overall higher throughput than the single-CE. However, this comes at the cost of often higher latency. While improving PE utilization results in lower average latency per operation when all pipeline CEs are active, all CEs are not always active from a single-input (image) perspective. This idleness of certain CEs at certain stages (Figure 3.4b) increases the overall latency; this effect scales with the number of CEs [14, 128]. In practice, there is a trade-off between underutilization and latency. CNNs have tens to hundreds of layers; hence, solving PE underutilization completely may require large numbers of dedicated CEs, resulting in higher latency.

3.4.1.2 On-chip buffer requirements

There is a trade-off between on-chip buffer sizes and off-chip memory accesses. This section describes the buffer sizes needed to guarantee minimum off-chip accesses. For generality, we assume weights are stored off-chip at the beginning of a CNN inference. This is because CNNs may have several million weights that cannot always be stored fully on-chip. Consequently, *minimum off-chip accesses* in this context refer to one access per weight and zero access per FM

element (apart from the mandatory loads and stores of CNN first and last layers FMs). In the *single-CE* case, layers are processed one at a time. Hence, the OFMs are produced completely at the end of each layer. But only a portion of layer weights must be on-chip at a time (See Figure 3.4a). Equation 3.4 depicts the estimation of buffer size requirements, where Sz stands for size. The same buffers are reused across layers because layers are processed one at a time. Consequently, the buffering required to achieve the target minimum accesses must accommodate the largest layer FMs, both IFMs and OFMs, plus the largest weights tile. Note that FMs_i must account for multiple copies of the FMs in case a layer has residual connections [50] (e.g., L_2 in Figure 3.2).

$$BufferSz = \max_{i=1}^{layers} (FMsSz_i) + \max_{i=1}^{layers} (weightsTileSz_i) \tag{3.4}$$

In the *pipelined-CEs* case, achieving the defined minimum off-chip accesses requires that all the weights of the pipelined layers be kept on-chip after their first load. Otherwise, as can be inferred from Figure 3.4b, each weight needs to be accessed as many times as the pipeline stages in which its CE is active. Regarding the FMs, double buffering is required between the CEs to work concurrently. Equation 3.5 describes the on-chip buffer requirements of the pipelined-CEs to guarantee minimum off-chip accesses. Multiplication by 2 is because of double buffering. *FMsBufferSz* is determined per layer using Multiple-CE builder heuristics. Smaller buffers mean smaller on-chip memory, but less reuse and more weight movement [128, 197].

$$BufferSz = \sum_{i=1}^{layers} \Big(weightsSz_i + 2 \times FMsBufferSz_i \Big) \tag{3.5}$$

This section discusses buffer sizes that guarantee the defined minimum off-chip accesses, assuming unlimited on-chip memory. The following section covers the case where these buffers exceed the available on-chip memory.

3.4.1.3 Off-chip access

Equation 3.6 depicts an example of calculating off-chip accesses of a *single-CE* assuming an OS dataflow (Section 3.2), where *offCh(x)* evaluates to 1 if x is stored off-chip and 0 otherwise. As discussed, we assume that the weights are stored off-chip at the beginning of an inference for generality. Hence, the ideal scenario is when both IFMs and OFMs of a layer fit on-chip, resulting in accesses equal to the size of the weights. The equation also describes two options when the available on-chip memory does not guarantee the ideal scenario. The first option is an output stationary, local input stationary. In this option, an IFM element is loaded once from off-chip memory and used to produce all the dependent OFM elements. In this option, the weights need to be loaded multiple times. The second option is output stationary local weight stationary. This option is the opposite of the first, meaning that a weight is loaded once, but an IFM element is loaded multiple times. The adopted option is the one that has fewer memory accesses. Multiple-CE Builder heuristics identify the buffer sizes that minimize accesses in each option.

$$
\begin{aligned}
Accesses &= \sum_{i=1}^{layers} Acc(L_i, CE_j) \\
Acc(L_i, CE_j) &= argmin\Big(OFMsSz_i \times offCh(OFMs_i) + \\
&\quad min\Big(weightsSz_i \times \lceil \frac{IFMsSz_i}{IFMsBufferSz_j} \rceil + \\
&\quad IFMsSz_i, IFMsSz_i \times \lceil \frac{weightsSz_i}{weightsBufferSz_j} \rceil + \\
&\quad weightsSz_i \Big) \times offCh(IFMs_i) + \\
&\quad \Big(1 - offCh(IFMs_i)\Big) \times weightsSz_i \Big) \\
s.t.\ & weightsBufferSz_j + IFMsBufferSz_j + \max_{i=1}^{layers} \big(OFMsSz_i \times \\
&\quad (1 - offCh(OFMs_i))\big) \leq CE\_BufferSz_j
\end{aligned}
\tag{3.6}
$$

Equation 3.7 gives off-chip accesses in the *pipelined-CEs* case where $offCh(weights_i, j)$ evaluates to 1 if the weights of the layer are off-chip at the beginning of stage j. As the weights are initially off-chip, $offCh(weights_i, 1)$ is always 1. In other stages, the weights of a layer are stored on-chip if the space allows. Each weight that is not kept on-chip after the first load must be accessed as many times as the pipeline stages in which its layer is active (see Figure 3.4b). In the pipelined-CEs case, the FMs are kept on-chip. This is practically the case when doing tile-grained pipelining, as the buffer sizes are tailored to the available on-chip memory.

$$
Accesses = \sum_{i=1}^{CEs} \sum_{j=1}^{CE\_Stages_i} weightsSz_i \times offCh(weights_i, j) \tag{3.7}
$$

3.4.2 Bottom-up modeling: From blocks to full accelerator

Modeling a multiple-CE accelerator using the models of its building blocks and the interfacing between them is achieved using generalized versions of the equations from the previous section (Section 3.4.1) or composite equations formed by substituting some into others. The nature of the modifications to the modeling in the previous section depends on **(1)** whether the building block processes *one or multiple segments* of a CNN, and **(2)** whether there is an *inter-segment pipelining* across segments (coarse-grained). This dependency is described in the rest of this section at a high level and clarified using *Segmented* architecture as a concrete example.

3.4.2.1 Latency and throughput

Latency and throughput are estimated using the same equations, whether the building block processes one or multiple segments. However, the final values

differ because when a block processes multiple segments, its PEs, parallelism, and hence utilization are optimized for the average case rather than for a unique segment. The latency and throughput of segments with inter-segment pipelining are estimated similarly to a pipelined-CEs block. However, there is a difference in how the latencies of the pipeline stages are estimated. This is because, in inter-segment pipelining, each stage could comprise a set of CEs rather than one CE and because the granularity is a whole input rather than a tile. The difference between fine and coarse-grained pipelining can be inferred by comparing the Segmented with the Hybrid and SegmentedRR architectures in Figure 3.2. To account for these differences, equations 3.2 and 3.3 are modified in two ways depending on the types of the coarsely-pipelined blocks. For example, in the Segmented case where the coarse-grained blocks are of single-CE type (Figure 3.2): **(1)** the stage latency estimation part in Equation 3.2 is substituted by Equation 3.1 because each stage is a single-CE processing a set of layers, **(2)** in Segmented coarse-grained pipeline, each segment processes a distinct image concurrently. Hence, the condition that keeps track of active CEs in Equation 3.2 must have an additional parameter to identify which CEs are processing the input for which throughput and latency are being estimated. When there is no inter-segment pipelining, the total latency is simply the sum of the segment latencies, and the throughput is the inverse of the latency. In both cases, there is an extra inter-segment communication latency.

3.4.2.2 On-chip buffer requirements

When a building block processes multiple segments, the on-chip buffer sizes are estimated considering the worst case, meaning that the buffers must accommodate the largest weight and FM tiles of the layers across these multiple segments. When there is inter-segment pipelining, the interfacing between the segments must apply double-buffering at input granularity between every two segments. Equation 3.8, which estimates Segmented architecture buffer sizes, clarifies these points. In a Segmented architecture, in principle, a CE could process multiple segments. Hence, $CE\_BufferSz_i$ estimation in Equation 3.8 is simply a generalization of Equation 3.4, applied across CE segments to account for the worst case. Because Segmented has inter-segment pipelining, there is double-buffering between every two segments ($2 \times interSegBufferSz_i$).

$$
\begin{aligned}
BufferSz &= \sum_{i=1}^{Segments} (CE\_BufferSz_i + 2 \times interSegBufferSz_i) \\
CE\_BufferSz_i &= \max_{j=1}^{CE\_Segments_i} \left(\max_{k=1}^{layers_j} (FMsSz_k) \right) + \\
&\quad \max_{j=1}^{CE\_Segments_i} \left(\max_{k=1}^{layers_j} (weightsTileSz_k) \right)
\end{aligned}
\tag{3.8}
$$

When there is no inter-segment pipelining, only a single buffer is reused across segments, but to guarantee the defined minimum off-chip accesses, that buffer must accommodate the largest inter-segment intermediate results.

3.4.2.3 Off-chip access

Off-chip accesses of a set of segments are the sum of intra-segment accesses plus the possible off-chip accesses on the interface between them. While the intra-segment off-chip accesses are computed using the same equations whether the same block processes multiple segments or not, there is an indirect dependency. The accesses depend on the on-chip buffer sizes (see Equations 3.6, 3.7), which in turn depend on whether multiple segments are processed using the same block as discussed in Section 3.4.2.2. The off-chip accesses on the interface between the segments depend on the *inter-segment pipelining*. Inter-segment pipelining double-buffering requires more space; when not available on-chip, the data communicated between the segments must be stored and loaded from off-chip. For example, Equation 3.9 calculates the Segmented architecture off-chip accesses, where $Acc(L_j, CE_i)$ is calculated using Equation 3.6. Segmented architecture has inter-segment pipelining, and the inter-segment FMs are stored off-chip if the on-chip cannot accommodate them.

$$Accesses = \sum_{i=1}^{Segments} \left(\left(\sum_{j=1}^{Layers_i} Acc(L_j, CE_i) \right) + 2 \times interSegBufferSz_i \times offCh(interSegBuffer_i) \right) \tag{3.9}$$

This section used the Segmented architecture as a concrete example. SegmentedRR, Hybrid, or any multiple-CE architecture modeling builds on top of the models of the single-CE and pipelined-CEs similarly, taking into account the impact of single-block processing multiple segments and the existence or absence of inter-segment pipelining.

3.5 Evaluation

3.5.1 Experimental Setup

3.5.1.1 Platforms and systems

Table 3.2 shows the FPGA boards used in our evaluation, focusing on PEs (DSPs), on-chip memory (Block RAM), and off-chip bandwidth. The boards represent various resource budgets with PEs ranging from hundreds to thousands and on-chip memory ranging from 2.4 to 16 MiB[2]. To evaluate the accuracy of MCCM, we use Vitis-IDE (2021.2) to implement and evaluate a set of multiple-CE accelerators using high-level synthesis (HLS). Vitis synthesis results are commonly used both in the literature and as a part of the design process in the industry. MCCM and the evaluation methodology are implemented in Python and C++. The experiments are conducted on a machine with 16 logical cores, Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz.

[2]The on-chip memory values are reported in MiB rather than Mb, which is commonly used by AMD

Table 3.2: Evaluation FPGA boards

	ZC706	VCU108	VCU110	ZCU102
DSPs	900	768	1800	2520
Block RAM(MiB)	2.4	7.6	4	16.6
Off-chip memory BW (GB/s)	3.2	19.2	19.2	19.2

Table 3.3: Evaluated CNN models

	ResNet152	ResNet50	XCeption	DenseNet121	MobileNetV2
Abbreviation	Res152	Res50	XCp	Dns121	MobV2
Weights (M)	60.4	25.6	22.9	8.1	3.5
Conv layers	155	53	74	120	52

3.5.1.2 Workloads

We use a representative set of CNNs with different structures, depths, and parameter (weight) counts. These are ResNet152 and ResNet50 [50], XCeption [21], MobileNetV2 [142], and DenseNet [58]. Table 3.3 shows the relevant characteristics of these CNNs. The results presented can be generalized as the evaluated CNNs share core blocks with many recent CNNs. For example, MBConv, the core block of MobileNetV2, is used in EfficientNet [160] and MnasNet [159]. The same applies to the Residual blocks of ResNets, and the Extreme Inceptions of XCeption are also used in recent CNNs.

3.5.1.3 Baseline architectures

To validate MCCM and present a set of use cases, we implement representative accelerators of the three multiple-CE architectures (Section 3.2.3). The Segmented implementation is based on Shen *et al.* [150]. The SegmentedRR tiling and buffer designs and granularity are based on Wei *et al.* [174], and the engine design is based on Ma *et al.* [106]. The implementation of Hybrid architectures is based on Qararyah *et al.* [128]. The experiments use 10 different CE counts in each architecture, starting from 2 CEs (the smallest possible CE count in a multiple-CE accelerator) to 11 CEs. These CE counts cover all possibilities in the baseline architectures. However, MCCM does not impose a limitation on the number of CEs. The number of PEs in a CE depends on the PE budget and is proportional to the CE workload. For example, in a multiple-CE accelerator with 2 CEs on VCU108, the 768 DSPs are distributed on the 2 CEs, and in an accelerator with 11 CEs, the same 768 DSPs are distributed on the 11 CEs.

3.5.2 MCCM validation

Table 3.4 summarizes 150 experiments validating MCCM estimation accuracy. The accuracy of the estimation is calculated using Equation 3.10. MCCM off-chip access calculations are exact since the accesses are deterministic and independent of the optimizations of the synthesis. MCCM accuracy ranges from 80.7% to 100%. The average accuracy values are $> 90\%$ for the three architectures. Moreover, the average accuracies of the three architectures are close to each other; the differences among them are $< 5\%$. There are differences between latency and throughput accuracies in most cases because in multiple-CE accelerators, throughput is not necessarily the inverse of end-to-end latency

Table 3.4: MCCM evaluation accuracy on VCU108: summary of 150 experiments (3 architectures × 10 CE counts × 5 CNNs)

	Max			Min			Average		
	Segmented	SegmentedRR	Hybrid	Segmented	SegmentedRR	Hybrid	Segmented	SegmentedRR	Hybrid
On-chip buffers	99.4%	99.7%	99.8%	84.2%	91.2%	84.6%	93.1%	97.4%	95.4%
Latency	98.8%	99.0%	99.6%	87.2%	80.7%	85.0%	92.8%	93.3%	92.5%
Throughput	99.6%	100%	99.6%	86.4%	83.4%	85.0%	93.9%	95.1%	92.5%
Off-chip accesses	100%	100%	100%	100%	100%	100%	100%	100%	100%

(Section 3.4.1.1). MCCM correctly predicted the best architecture among the three in 139 of the 150 experiments when considering on-chip buffer sizes, and in all experiments when considering latency, throughput, and accesses. Performance differences between architectures vary greatly and are sensitive to CE count. For example, considering ResNet50, SegmentedRR has the lowest latency of 73% normalized to the latency of Hybrid and 58% normalized to the latency of Segmented when using 2 CEs. In contrast, Hybrid has the lowest latency of 20% normalized to the latency of Segmented and 96% normalized to the latency of SegmentedRR when using 11 CEs.

$$Accuracy = 100 \times \left(1 - \frac{|synthesis\ result - estimated|}{synthesis\ result}\right)\% \tag{3.10}$$

3.5.3 Use Case 1: End-to-end evaluation

Table 3.5: Multiple-CE accelerators that achieve best results (■ Segmented ■ SegmentedRR ■ Hybrid) and their CE-counts. Multiple colored cells in the same column for the same metric indicate a tie. We consider results within a 10% difference as a tie to account for estimation errors.

	zc706					vcu108					vcu110					zcu102				
	Res152	Res50	XCp	Dns121	MobV2	Res152	Res50	XCp	Dns121	MobV2	Res152	Res50	XCp	Dns121	MobV2	Res152	Res50	XCp	Dns121	MobV2
Latency		2	2					2							2					
	2			3	3	2	2		3	3	2	3		3	3	2	2	2	4	
			4		2			4		3			4		7			4		7
Throughput	3	2	2		8	3	2	2			3	6			9	4		2	5	11
		10		11	2		6		10	3		11	2	11			8		3	7
Access						2	2		2	2	2	2	2	2	2				2	
				2	2	2	2	2	2	2	2	2	2	2	2				2	2
	3	3	7	3	6	3	3	7	3	5	2	2	6	2	5	4	6	7	3	5
Buffers	2	2	2				2				2					2	2	2		
				2	3				2	3				2	3				2	3
		2	6		11	2	2	7		11	3	3	7		11		2	6		11

This use case discusses MCCM end-to-end evaluation of the baseline multiple-CE architectures. Table 3.5 depicts a summary of this evaluation.

It shows MCCM selection of the accelerators that achieve the best results considering different metrics, CNNs, and FPGA boards. Four insights can be identified by examining Table 3.5. *First*, in 80% of the cases, no single architecture provides the best results in the four metrics, even for the same CNN model and board. This is shown in the table as 16 out of 20 columns do not have four cells of the same color. The 4 cases where Hybrid is best in all metrics belong to MobV2 and XCp; these CNNs have various convolution types. This is expected as the Hybrid architecture was proposed to handle such CNNs [128]. *Second*, even when a single architecture yields the best results in all metrics, no single instance of that architecture is always the best. For example, for MobV2 on ZC706 (fifth column), the Hybrid accelerator of 2 CEs has the best latency and throughput, the one with 6 CEs has the minimum off-chip accesses, and the one with 11 CEs has the minimum buffer requirements. *Third*, some metrics are uniquely dominated by one of the architectures. For example, SegmentedRR achieves the best latency in 15 out of 20 cases. This is because SegmentedRR architectures were proposed to minimize latency [167, 174]. Hybrid has the lowest buffer requirements in 14 out of 20 cases. *Fourth*, the Hybrid always achieves the minimum off-chip accesses as this was one of its design objectives [128]. The other two architectures catch up on VCU108 and VCU110, as these boards have relatively large on-chip memories that accommodate big enough buffers required to achieve the minimum off-chip access (Table 3.2).
In summary, multiple-CE accelerators' different CE arrangements affect their performance and efficiency, as discussed in detail in sections 3.4.1 and 3.4.2. This highlights the importance of exploring different CE arrangement possibilities and identifying the best ones considering the CNN models, hardware resources, and the metric of interest.

3.5.4 Use Case 2: Fine-grained evaluation

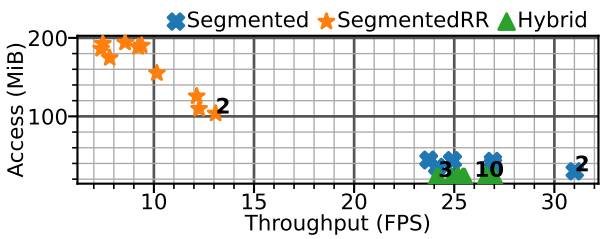
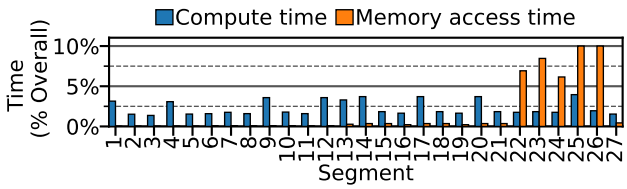


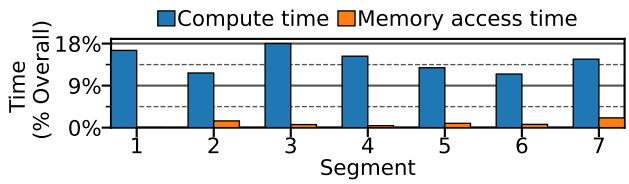
Figure 3.5: Throughput vs. off-chip memory accesses of ResNet50 on ZC706 using 10 accelerator instances per architecture with 2-11 CEs. The numbers indicate the CE counts of the accelerators with the highest throughput or minimum accesses of each architecture.

This use case demonstrates MCCM *fine-grained* evaluation to identify the performance bottlenecks of multiple-CE architectures and guide optimizations that alleviate these bottlenecks. Figure 3.5 depicts MCCM estimations of throughput and off-chip accesses of the three baseline architectures. As can be seen, both Hybrid and Segmented have relatively good results in both metrics.

SegmentedRR instances, however, have considerably more off-chip memory accesses that form a bottleneck. Figure 3.6a shows the MCCM breakdown of compute and memory access time of SegmentedRR with 2 CEs, which has the highest throughput among SegmentedRR in Figure 3.5. In segments $22 - 26$, the memory access time is the bottleneck. In 29% of the overall execution time, CEs are idle, waiting for data. The Segmented and Hybrid, by contrast, have no such bottlenecks. For example, the breakdown of compute and memory access time of Segmented with 7 CEs, which has the highest throughput among Segmented, is shown in Figure 3.6b.



(a)



(b)

Figure 3.6: Segments compute and memory access time normalized to the overall execution time of **(a)** SegmentedRR with 2 CEs, **(b)** Segmented with 7 CEs using ResNet50 on ZC706

When considering alleviating the off-chip access bottleneck, data compression is a candidate optimization [19]. However, compression has its overhead. MCCM fine-grained evaluation helps identify the segments that form bottlenecks and apply compression only to these segments' layers to ensure minimum overheads. For example, Figure 3.6a suggests that compression would be beneficial only for the layers of segments $22 - 26$. Moreover, it is crucial to identify which data dominates the accesses. For example, Figure 3.7 shows the MCCM breakdown of the accesses of the accelerator instances with the highest throughput in each of the three architectures shown in Figure 3.5. Figure 3.7 suggests that while in SegmentedRR and Hybrid cases, compressing the weights would have a considerable impact on the accesses, compressing FMs would be a pure overhead.

3.5.5 Use Case 3: Guiding design space exploration

This use case demonstrates how MCCM fast evaluation enables a systematic multiple-CE accelerator design approach based on identifying the performance bottlenecks and exploring the space of design points that alleviate these bottlenecks. This use case aims to identify the architecture of a multiple-CE accelerator that maximizes throughput while minimizing on-chip memory us-

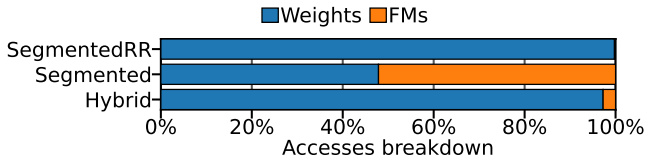


Figure 3.7: Off-chip memory access breakdown of SegmentedRR with 2 CEs, Segmented with 7 CEs, and Hybrid with 9 CEs using ResNet50 on ZC706.

age. The CNN and board used are XCp and VCU110. Figure 3.8 shows the trade-off between throughput and on-chip buffer requirements using XCp on VCU110.

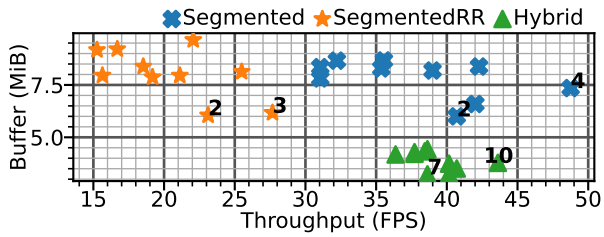
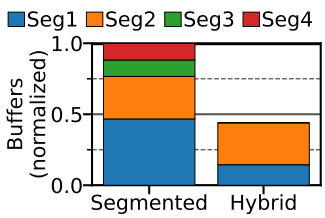
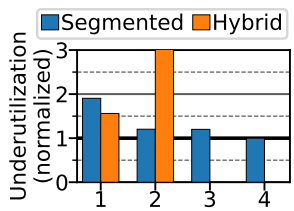


Figure 3.8: Throughput vs. on-chip buffer using XCp on VCU110 using 10 accelerator instances per architecture with 2-11 CEs. The numbers indicate the CE counts of the accelerators with the highest throughput or minimum buffer requirements of each architecture.

One way to derive multiple-CE architectures with better throughput-buffer trade-offs is to take the most promising architectures in each metric as starting points, identify their bottlenecks using MCCM fine-grained evaluation, and explore architectures that mitigate these bottlenecks. The most promising architectures in Figure 3.8 are those located in the bottom right region. For example, the Segmented accelerator with 4 CEs has the highest throughput, and the Hybrid with 7 CEs has the lowest buffer requirements.



(a) Per-segment buffers normalized to the Segmented total buffer size



(b) Per-segment PE underutilization normalized to the minimum underutilization

Figure 3.9: Buffer and PE underutilization of Hybrid with 7 CEs (2 segments) and Segmented with 4 CEs (4 segments)

Figure 3.9 depicts MCCM evaluation of the bottlenecks of the two mentioned instances of Segmented and Hybrid architectures. As Figure 3.9a shows, the buffers of the first segments of the Segmented form a bottleneck. The opposite is the case for Hybrid. Regarding throughput, both Hybrid and Segmented have coarse-grained pipelining (Section 3.4.2). Hence, their throughput is determined by the slowest segment execution time. MCCM breakdown shows that the first block is the bottleneck in the case of Segmented, but the last block is the bottleneck in the case of Hybrid. The main reason behind that is the PE underutilization shown in Figure 3.9b. The higher the PE underutilization is, the slower the block.

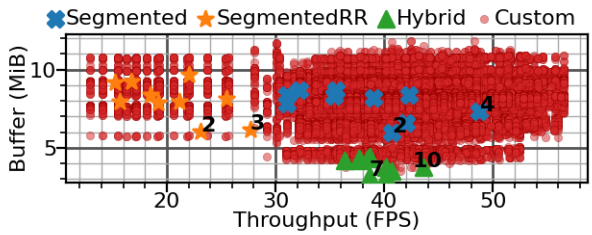


Figure 3.10: Throughput vs. on-chip buffer using XCp on VCU110 of a sample of 100000 custom accelerators

The observation that the bottlenecks are in the Segmented first block and Hybrid second block hints that a custom architecture that comprises a Hybrid-like first block followed by Segmented-like blocks may improve efficiency. Assuming 10 CE possibilities (2-11 CEs) and given the XCp model structure, the design space of such a custom architecture has roughly *97.1 billion* designs. Figure 3.10 shows an evaluation of a random sample of 100000 out of them. MCCM evaluation of this sample took 10.5 minutes with an average of *6.3 ms* per design. The average synthesis time of a single design on the same machine is roughly an hour, meaning that MCCM is in the order of *100000*× faster. Traditional synthesis-based evaluation of the sample of 100000 designs would take *years*. Exploring the selected sample resulted in identifying custom accelerators that outperform the state-of-the-art. Some custom accelerators achieve the throughput of Segmented with 4 CEs while reducing the buffer requirements by up to 48%. Custom accelerators with the highest throughput improve throughput by up to 17% compared to Segmented with 4 CEs while reducing buffer requirements by up to 39%.

3.6 Related work

The literature on DL accelerator design suggests that no fixed architecture is optimal given the diverse workloads and application requirements, and that model-aware CNN and DNN in general, to hardware mappings achieve the best performance and efficiency [9, 10, 14, 84, 122]. The prior art on optimizing CNN-to-hardware mapping takes two main forms. The first form focuses on intra-layer optimizations, including dataflow and reuse, parallelism strategies, tiling, and exploring custom algorithms like GEMM and FFT [18, 20, 74, 84,

90, 104, 106, 122]. Timeloop [122] and MAESTRO [84] provide frameworks for evaluating and exploring the intra-layer architecture design space. The second form focuses on inter-layer or model-level optimizations, including designing layer-dedicated compute engines (CEs), convolution layer-fusion, and inter-layer pipelining [9, 15, 43, 126, 166, 188, 197].

FPGAs are a common target for inter-layer optimizations due to their reconfigurability, which enables arranging the resources into a custom number of dedicated CEs. When designing a multiple-CE accelerator, there are numerous CE arrangement possibilities. An obvious arrangement is a set of layer-specific and fully pipelined CEs where the number of CEs is equal to the number of CNN layers [9, 166, 197]. More scalable, resource- and latency-aware arrangements have an adjustable number of CEs, which is determined by both the CNN structure and the hardware budget [2, 14, 117, 128, 149, 150, 167, 174, 199].
The prior art multiple-CE accelerators follow a set of fixed design templates and are not based on systematic design space exploration. This paper proposes a multiple-CE accelerator analytical cost model and a fast evaluation methodology based on this model. Fast evaluation of multiple-CE architectures is key to a systematic design approach.

3.7 Conclusion

Multiple-CE accelerators are more adaptable to various CNN model structures and application performance metrics than generic accelerators. However, existing multiple-CE accelerators have fixed architectures and do not explore the impact of CE arrangements on performance and efficiency. This chapter presented Multiple-CE accelerator analytical Cost Model (MCCM), and a fast MCCM-based evaluation methodology. MCCM is in the order of 100000× faster than traditional synthesis-based evaluation and has an average accuracy of $> 90\%$. MCCM helps to identify the best-performing among state-of-the-art multiple-CE accelerators given various metrics, CNNs, and resource budgets. Moreover, MCCM fast evaluation permits identifying performance bottlenecks and exploring the vast space of CE arrangements, opening the door to a more systematic approach to multiple-CE accelerator design.

Chapter 4

Model-Specific Multi-Engine DL Accelerator DSE Framework

This chapter presents a methodology for designing model-specific multi-engine DL accelerators. At the core of the methodology is MCExplorer, a framework for exploring the design space of these accelerators. MCExplorer uses the MCCM fast evaluation methodology presented in Chapter 3. MCExplorer employs single- and multiobjective optimization heuristics to identify accelerator architectures optimized for throughput, latency, energy efficiency, and trade-offs among these metrics given hardware resource budgets. Unlike existing approaches, MCExplorer explores a broader design space by considering unrestricted engine arrangements and distinct compute-engine architectural parameters.

4.1 Introduction

The increasing complexity of Deep Learning (DL) models and the heterogeneity of their operations (layers) hinder monolithic DL accelerators, with a single reusable Compute Engine (CE), from maintaining optimized performance, efficiency, and scalability [10, 12, 172, 199]. As a result, there is a trend towards modular accelerators composed of multiple Compute Engines (CEs) [14, 43, 128, 153]. Having multiple CEs enables tailoring different CEs to the varying computational characteristics of the model layers. Hence, multiple-CE accelerators usually outperform monolithic ones when processing complex DNNs with heterogeneous layers [14, 128, 199].

This chapter is based on the following publications:
F. Qararyah, M. A. Maleki, and P. Trancoso, "MCExplorer: Exploring the Design Space of Multiple Compute-Engine Deep Learning Accelerators," ACM Transactions on Architecture and Code Optimization, vol. 22, no. 4, pp. 1–27, 2025.

Many multiple-CE DL accelerators utilize reconfigurable platforms, such as Field-Programmable Gate Arrays (FPGAs), as configurability provides flexibility to design dedicated CEs tailored to the characteristics of the model layers [9,128,150,168,174,197,199]. These model-aware multiple-CE accelerators usually outperform generic ones, considering different optimization goals and under varying resource budgets [128, 168, 199]. However, the design space of multiple-CE accelerators has not been adequately explored. Hence, the potential of such accelerators has not been fully leveraged [130].

Because an exhaustive exploration of the design space of multiple-CE accelerators is impractical [130, 150, 199], previous work restricts the design space in two ways. *First*, the previous work adopts fixed CE arrangements and adjusts only the number of CEs depending on the DNN structure and the hardware resource budget [9, 150, 174, 197, 199]. *Second*, the previous work usually applies a single parallelism strategy and dataflow across all CEs, and tunes only the degrees of parallelism of these CEs [9, 150, 174, 197]. For example, Wei *et al.* [174] implement all CEs as systolic arrays with the same parallelism strategy and dataflow; these systolic arrays differ only in their shapes. Even in cases where CEs have heterogeneous parallelism strategies, deciding on these strategies is fixed rather than exploration-based [128]. Such restrictions to the design space result in the state-of-the-art multiple-CE accelerators being outperformed even by random design space exploration (DSE) [130]. In addition to restricting the design space, the previous work generally focuses on optimizing one performance metric and measuring improvements in other metrics as a by-product. While some mainly optimize latency [174], others target throughput [9, 150].

To overcome the limitations of previous work, we propose a DSE framework of FPGA-based multiple-CE accelerators (MC**E**xplorer). MC**E**xplorer streamlines the search for an optimized multiple-CE accelerator given a DNN, FPGA resources, and custom optimization objectives. MC**E**xplorer explores a space beyond that explored in the literature. To our knowledge, MC**E**xploreris the first framework that does not restrict accelerator inter-CE arrangements and considers multiple distinct per-CE configurations, including parallelism strategies and data flow options. As a result, MC**E**xplorer always identifies multiple-CE accelerators that outperform the state of the art. Moreover, MC**E**xplorer optimizes for different metrics, including throughput, latency, energy efficiency, and the trade-offs among these metrics. MC**E**xplorer comprises a set of heuristics with differing design philosophies, offering exploration speed, scalability, and better search quality by reducing the chance of convergence to local minima. Our contributions are as follows.

- We propose MC**E**xplorer, a framework that explores the design space of multiple-CE accelerators with flexible inter-CE arrangements and per-CE configurations. MC**E**xplorer identifies an optimized multiple-CE accelerator given a DNN model, hardware resource budget, and custom optimization objectives.

- We formulate a set of exploration heuristics that adopt different design philosophies, offering speed and scalability, and reducing the chances of converging to local minima. These include Genetic Algorithm (GA), Simulated Annealing (SA), and a novel, model-aware search heuristic

named Hierarchical Mapping (HM).

- We formulate a multiobjective optimizer based on Non-Dominated Sorting Genetic Algorithm (NSGA-II) to optimize multiple-CE accelerators' performance and efficiency trade-offs. We use the optimizer to identify multiple-CE accelerators that simultaneously optimize performance and energy efficiency.
- We demonstrate the effectiveness of MC**E**xplorer through an extensive evaluation using representative DNNs and hardware resource budgets. The evaluation shows that MC**E**xplorer identifies optimized multiple-CE accelerators that outperform the state of the art, considering different optimization objectives.

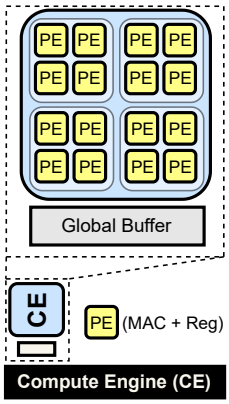
Our evaluation of MC**E**xplorer with various DL models and FPGA boards shows that exploring the design space of multiple-CE accelerators without restricting the inter-CE arrangements and per-CE configurations results in accelerators that outperform the state-of-the-art in all targeted optimization objectives, achieving up to 2.8× throughput improvement, 2.1× speedup, and 45% energy reduction. Moreover, the evaluation demonstrates that this comprehensive space exploration is crucial to identify multiple-CE accelerators with the best performance-efficiency trade-offs.

4.2 Background and Motivation

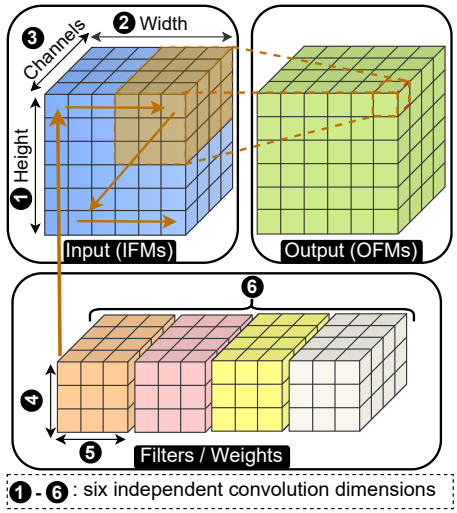
4.2.1 Compute Engines and their Parallelism Strategies and Dataflow

We use the term *Compute-Engine (CE)* to describe an array of Processing Elements (PEs) that cannot be independently controlled but work as a unit to compute a workload. Figure 4.1a shows a high-level overview of a CE. In this context, a PE is hardware that can perform a single MAC operation per cycle. A CE has two levels of memory; each PE has its local registers, and the CE has a buffer globally accessible by its PEs. In this work, we focus on computer vision DL models. As convolutions are the core operators in these models, usually representing more than 90% of their operations [106], our CEs mainly perform convolution operations. Figure 4.1b shows the inputs and outputs of a convolution operation (layer). The first input of a convolutional layer is a set of trainable parameters, known as weights. The weights are organized into filters. The second is Input Feature Maps (IFMs). The outputs are known as Output Feature Maps (OFMs). Feature Maps (FMs), or activations, refer to both IFMs and OFMs. FMs consist of 2D planes, known as channels.

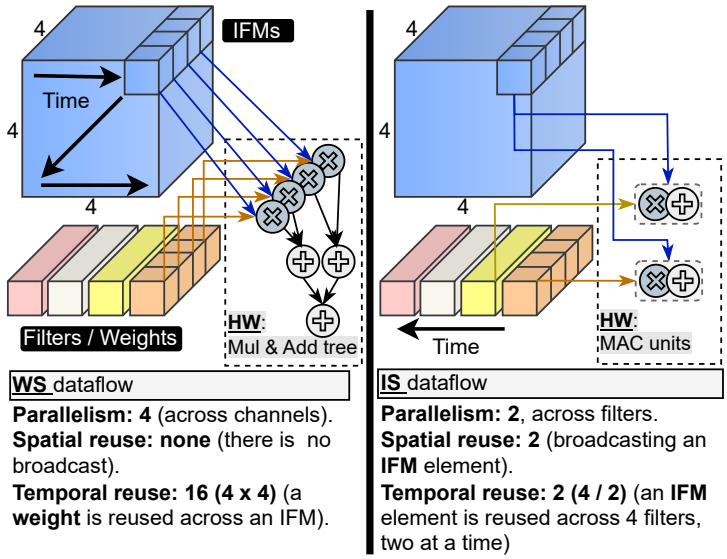
The performance and efficiency of a compute engine are largely determined by its *parallelism strategy* and *dataflow* [74]. The *parallelism strategy* describes the number of CE parallelism levels and the specific dimensions to parallelize or spatially unroll among the six independent convolution dimensions (❶–❻) depicted in Figure 4.1b. The CEs in most state-of-the-art accelerators apply parallelism on one to three of the six dimensions [2,14,106,107,115,150,168,194, 197,199]. Figure 4.1c shows two examples of simple one-dimensional parallelism. The parallelism strategy dictates the required hardware, PE utilization, and



(a) Compute Engine high-level overview.



(b) Convolution operation input, output, and six dimensions.



(c) Simple examples of convolution engines, parallelism, and dataflow.

Figure 4.1: (a) Compute Engine (CE) overview, (b) convolution operation, and (c) simplified convolution engines parallelism and dataflow examples.

data reuse. A mismatch between FMs and weight topologies of a layer and the parallelism dimensions of a CE results in PE underutilization [130]. PE underutilization negatively impacts performance as well as resource and energy efficiency. Regarding data reuse, the parallelism strategy affects reuse by different PEs in the same clock cycle, known as *spatial reuse*. For example, the parallelism strategy in the example on the left in Figure 4.1c does not have spatial reuse. In contrast, the parallelism on the right has a spatial reuse of 2 since a single FM element is used by two PEs in the same clock cycle. Note that for simplicity, the figure describes reuse only at the PE register level.

We use the term *dataflow* to describe both the scheduling and mapping of a convolution to the CE PEs. The most common dataflow types are weight-stationary (WS), output-stationary (OS), and input-stationary (IS). The name of a dataflow indicates which of the IFMs, OFMs, or weights change least frequently [84]. Dataflow choice has an impact on the data movement across different memory levels. An aspect of this is the impact of the dataflow on *temporal reuse*. Temporal reuse refers to data reuse by a single PE over consecutive clock cycles. Higher temporal reuse means less data movement to and from the memories farther from the PEs, reducing latency and energy consumption. Figure 4.1c shows the impact of the dataflow (WS compared to IS) on temporal reuse. For example, in the WS case, on the left side of the figure, each weight is loaded once and applied to all elements in an IFM channel (IFM width $\times$ IFM height).

4.2.2 From Single-CE to Multiple-CE DL Accelerators

Single-CE accelerators, where a single PE array is used to process all the core layers of a DNN, do not lead to efficient DL. Different layers of a DNN have varying input and output sizes and topologies and differ in their arithmetic intensities; hence, using a single CE with a monolithic parallelization strategy and dataflow to process these DNNs results in suboptimal performance and efficiency [10, 14, 174]. As a result, there is usually a considerable gap between the peak performance and efficiency of these accelerators and what is achievable. Moreover, the gap between the peak and the achieved performance of single-CE accelerators increases when scaling these accelerators [141, 172]. This increase can be mitigated by having large and high-bandwidth on-chip memory, but this results in area and energy overhead [10, 172]. Designing a flexible PE array that can be configured to suit the characteristics of different layers is one approach to mitigate the inefficiencies of single-CE accelerators [182, 190]. For example, Xie *et al.* [182] propose a Genetic Simulated Annealing (GSA) algorithm to explore the space of dynamic, per-layer, parallelism strategies that maximize the resource utilization and minimize the latency. However, this approach also has its overheads in terms of area and energy [26, 198]. To overcome these shortcomings and overheads and offer better performance and efficiency, many state-of-the-art DL accelerators are adopting modular designs with multiple CEs [12, 14, 43, 145, 150, 170, 174].

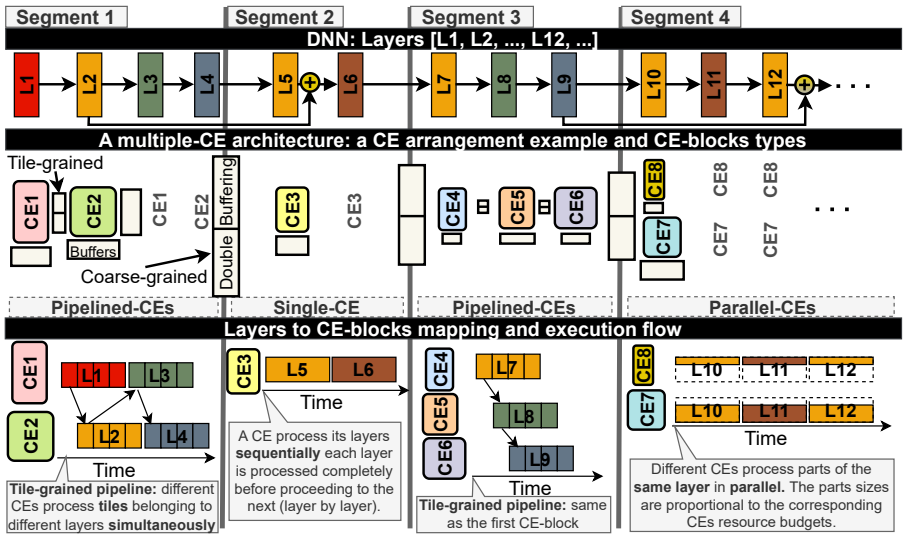


Figure 4.2: DNN to a multiple-CE architecture mapping example. The example shows 4 segments that divide the figure vertically, and three sections that divide the figure horizontally. The sections show DNN layers, the types of CE-blocks used, and the mapping and execution flow of the layers on their CE-blocks. CEs and buffers vary in size and shape, proportional to their segment layers' needs.

4.2.3 Multiple-CE Accelerators on FPGAs

Reconfigurable platforms, such as FPGAs, offer the flexibility to organize their resources into a model-aware accelerator with dedicated CEs. As a result, various FPGA-based multiple-CE accelerators have emerged [9, 117, 128, 150, 168, 174, 197, 199]. A recent survey shows that single-engine approaches are becoming a minority among FPGA acceleration toolflows [68]. Figure 4.2 shows an example of a DNN to multiple-CE accelerator mapping. A DNN is split into *segments*, each of which is processed using a *CE-block* of one or more CEs. Two basic CE-blocks, shown in the figure, are used across the existing multiple-CE accelerator. These are *single-CE* blocks and *pipelined-CEs* blocks [130]. The single-CE block processes its assigned DNN segments in a layer-by-layer (LBL) fashion, where a layer is processed completely before moving to the next. The pipelined-CEs block comprises a set of pipelined CEs, which process the layers in a layer-pipelined (LP) fashion, where multiple layers are processed simultaneously. Different CE-blocks, similar to different CEs within a pipelined-CEs block, work in pipelined fashion, where the output of a pipeline stage is the input to the next. In this context, double buffering is used to loosen the data dependencies by allowing different stages to work on different consecutive inputs or tiles simultaneously. Different CE-blocks usually process whole FMs belonging to different consecutive inputs simultaneously, and the pipelining among them is referred to as coarse-grained pipelining. By contrast, different CEs within a pipelined-CEs block usually process different tiles of the same input, and the pipelining among them is referred to as tile-grained pipelining.

In addition to these two blocks, we present the *parallel-CEs* block in this work. In the parallel-CEs block, the PEs are arranged into distinct CEs non-uniformly. These CEs process the same layer simultaneously as depicted in the bottom right section of Figure 4.2. A Parallel-CEs block improves the scalability and enables higher performance when further scaling of a single-CE block becomes ineffective. Figure 4.3 depicts this using a simplified convolution example. The figure shows how scaling a single-CE from 3 to 4 PEs is ineffective in reducing the execution time, and how using parallel-CEs solves the issue. There are different ways to partition a convolutional layer workload on these CEs along the 6 dimensions depicted in Figure 4.1b. The choice of dimensions to partition across impacts data sharing. To give an example, considering the simplified convolution in Figure 4.3, there are 2 dimensions to partition across. The adopted partitioning in the presented parallel-CEs example is across the IFM's width (or columns), where the CE with 1 PE processes the 6th column and the CE with 3 PEs processes the rest. This results in *splitting* the IFMs and *duplicating* the weights throughout the execution. In more complex examples, each of the IFMs and weights is either split or duplicated. Regarding OFMs, the choice of the splitting dimensions necessitates *synchronization* between the CEs within the block if different CEs share an OFM element. This is because the accesses to the OFMs are reads and writes. This work explores only splitting choices where the CEs in the block have no shared OFM elements; hence, it avoids the need to synchronize among the CEs within the block while processing a layer. However, the execution is synchronized across different layers. This means that the CEs start processing layer $x + 1$ only after the slowest finishes its portion of layer x. Each of the parallel CEs has local buffers to store its working set data. The internal architecture of a CE in the parallel-CEs block in terms of dataflow and parallelism strategies is no different than other CE-blocks.

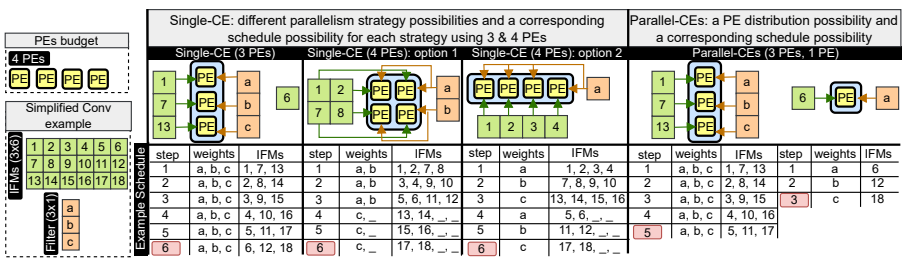
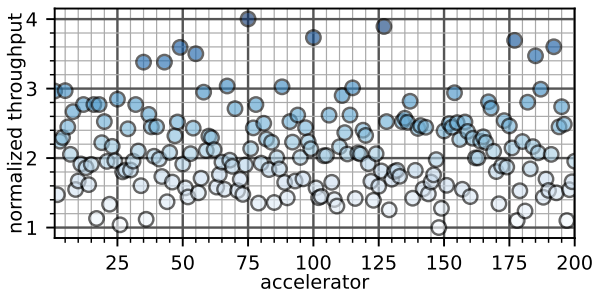


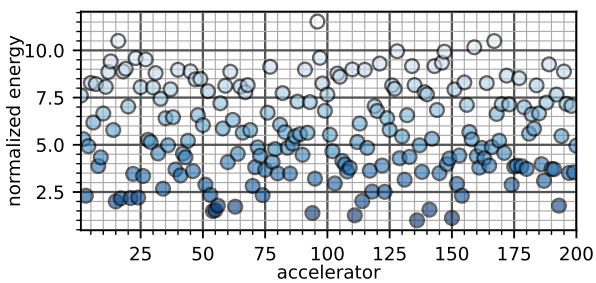
Figure 4.3: A simplified convolution example. PE distributions and example schedules of single-CE and parallel-CEs blocks using up to 4 PEs. The examples use a convolution with a single 3×1 filter and 3×6 IFM. A single-CE requires 6 steps using 3 PEs and does not scale effectively when using 4 PEs. Arranging the 4 PEs into two parallel CEs enables a shorter execution of 5 steps ($1.2x$ speedup).

4.2.4 Design Space Exploration of Multiple-CE Accelerators

The main limitation of state-of-the-art FPGA-based multiple-CE accelerators is that they explore a limited space of fixed architectural templates [130]. However, the arrangement of the CEs has a considerable impact on the accelerator performance and efficiency. Figure 4.4 depicts some aspects of such an impact. The figure shows the throughput and the energy per inference of 200 randomly generated CE arrangements. The figure demonstrates that the differences in throughput and energy efficiency are up to 4× and 11.5×, respectively. Exploring the entire design space of multiple-CE accelerators is impractical. To give an example, if we consider a simple case using only contiguous single-CE blocks, there are $\binom{\#Layers\ -\ 1}{\#CEs\ -\ 1}$ multiple-CE accelerator possibilities, which means combinatorial growth. For example, considering ResNet152 [50] and 10 CEs, there are roughly 10^{14} possible architectures. In this example, even if the modeling and evaluation of one architecture is optimized to take as little as $1\mu s$, an exhaustive sequential exploration would take years. This motivates efficient exploration of the design space of multiple-CE accelerators to identify architectures with promising performance and efficiency.



(a) Normalized throughput values of 200 randomly generated multiple-CE accelerators.



(b) Normalized energy-per-inference values of 200 randomly generated multiple-CE accelerators.

Figure 4.4: Throughput and energy of 200 randomly generated multiple-CE accelerators. The values are obtained using MCCM [130].

4.2.5 Scope of This Work

Figure 4.5 depicts the scope of this work in the broader context of FPGA-based DL accelerator design approaches. The focus is on *model-aware multiple-CE accelerators*. For brevity, we refer to such accelerators simply as multiple-CE accelerators throughout the paper. Note that the categorization in the figure is based on the scope rather than the potential of the work. For example, research work on model-aware accelerator design, including this work, can potentially be used to perform accelerator architecture search in Neural Accelerator Architecture Search (NAAS). However, we distinguish the current scope of this work from NAS since the terms NAS and NAAS are used only in works that involve actively modifying the architecture of a DL model, *i.e.*, its structure, parameters, and operations.

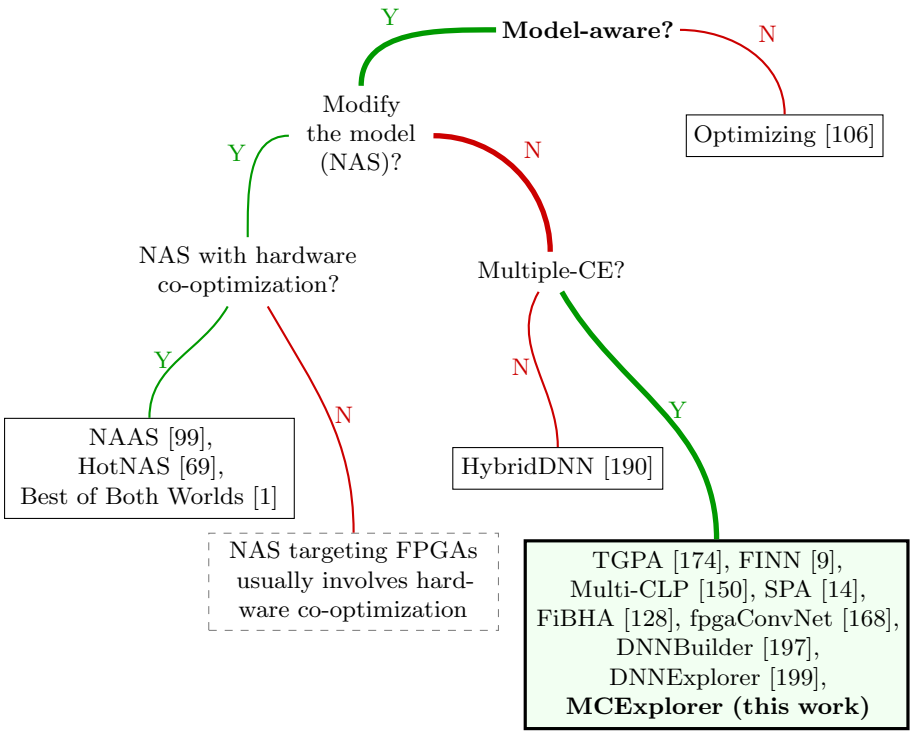


Figure 4.5: The scope of this work within the broader context of FPGA-based deep learning accelerator design approaches.

4.2.6 Design space exploration metaheuristics

4.2.6.1 Genetic Algorithm.

Genetic Algorithms (GA) are a population-based metaheuristic inspired by natural selection [55]. GA simulates evolution by maintaining a population of solutions (*individuals*) that evolve over generations through operators such as *crossover*, *mutation*, and *selection*. An individual is represented as a string or

a vector known as a *chromosome*. Each entry in the chromosome is known as a *gene*. The crossover operator combines parts of two chromosomes to produce new *offspring*. The crossover aims to produce better individuals by exploiting the good genes of existing ones, as defined by a fitness function. The mutation operator introduces random changes to a chromosome, maintaining diversity in the population, exploring new areas of the search space, and avoiding convergence to local optima. The selection operator chooses chromosomes from the population according to their fitness to serve as parents for the next generation. It biases the GA toward better solutions while maintaining genetic diversity. GA performs well in complex search spaces and has been adopted by state-of-the-art DNN-to-accelerator mapping [74, 114].

4.2.6.2 Simulated Annealing.

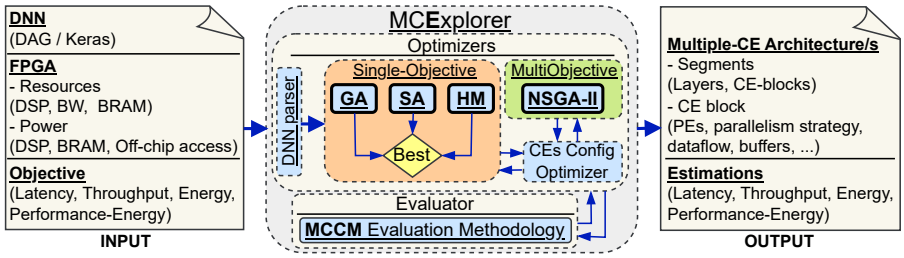
Simulated Annealing (SA) is a single-solution-based probabilistic metaheuristic inspired by the physical process of solid annealing [78]. SA starts with a usually randomly generated initial solution and a high *temperature*. The temperature in SA is a hyperparameter corresponding to the probability of accepting worse solutions during optimization. The initial high temperature encourages exploration of the solution space. SA explores the search space by *generating neighbors* through small changes to a solution. Throughout SA iterations, the temperature is gradually reduced according to a *cooling schedule*, and the algorithm focuses more on exploiting good solutions. SA has been used in related work for its simplicity and relatively fast convergence [12, 72, 201]

4.2.6.3 Non-Dominated Sorting Genetic Algorithm.

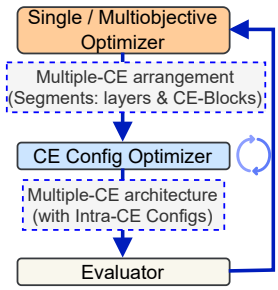
Non-Dominated Sorting Genetic Algorithm II (NSGA-II) is a type of genetic algorithm specifically designed for *multiobjective optimization*. NSGA-II uses representations and operators similar to those used in traditional GA. A key difference from traditional GA is that NSGA-II seeks a set of solutions with optimal trade-offs (a Pareto front), rather than a single solution. Hence, NSGA-II has a different selection mechanism than traditional GA. NSGA-II *selection* is based on techniques known as *non-dominated sorting* and *crowding distance* [32]. The non-dominated sorting technique arranges the solutions (individuals) into different *fronts* based on the number of others that dominate them. The crowding distance technique estimates how close an individual is to its neighbors in the search space. It is used to maintain a diverse Pareto front by favoring solutions that are located in less crowded regions of the search space. NSGA-II helps to avoid the hand-crafting of a cost function that comprises multiple criteria [114].

4.3 MCExplorer: a Framework for Multiple-CE Accelerators DSE

To satisfy the various DL applications' performance and efficiency objectives, the potential of multiple-CE accelerators needs to be fully leveraged. Fully leveraging the potential of multiple-CE accelerators requires exploring a comprehensive design space without restricting the CE arrangements or per-CE



(a) MCExplorer core modules.



(b) MCExplorer main loop.

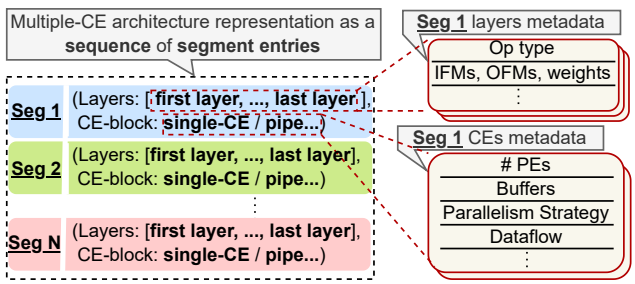
Figure 4.6: A high-level overview of MCExplorer components and its main optimization loop.

configurations. This section presents MCExplorer, a framework for conducting such a comprehensive DSE.

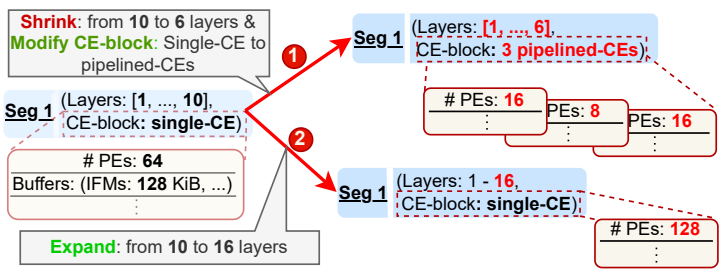
Figure 4.6a shows a high-level overview of MCExplorer. MCExplorer takes as *inputs*: **(1)** DNN representation. **(2)** FPGA resources, including the number of PEs (DSPs), the off-chip memory bandwidth, and the on-chip memory (BRAM) capacity; and the power consumption of these resources. **(3)** An optimization objective or objectives, *e.g.* throughput or throughput-energy trade-off. MCExplorer produces as *outputs*: **(1)** Multiple-CE architecture description when the optimization is single-objective, or a set of architecture descriptions representing a Pareto front when the optimization is multiobjective. The architecture description includes all the details required to implement a multiple-CE accelerator, including how the DNN and FPGA resources are partitioned among the CEs, the CEs' parallelism strategies, and dataflow. **(2)** Estimations of performance and energy efficiency of the described architectures.

MCExplorer has two main modules, the *Optimizers* module and the *Evaluator* module. The main contribution of this paper is the Optimizers module, which in turn contains four sub-modules. **First**, *DNN parser*, which transforms a DNN into a unified representation used by MCExplorer optimization heuristics. **Second**, a set of single-objective optimizers. These include two metaheuristics, *Genetic algorithm (GA)* and *Simulated Annealing (SA)*, and a novel heuristic named *Hierarchical Mapping (HM)*. GA is a scalable alternative that has been shown to outperform various optimization methods, including Reinforcement Learning (RL) [74, 139, 164]. We selected SA as it has provided high-quality results and relatively fast convergence on a related problem, namely scheduling DNNs on tiled architectures [12, 201]. In addition to these

metaheuristics, we proposed HM with a special focus on speed and scalability, as described in Section 4.3.2.3. The user can select to use MC**E**xplorer as a *black-box* where different optimizers explore the design space in parallel and produce the best solution, or to use one of these optimizers alone. **Third**, a multiobjective optimizer to identify Pareto optimal architectures with an optimized performance-energy trade-off. We formulate a multiobjective optimizer based on *Non-Dominated Sorting Genetic Algorithm II (NSGA-II)* [32]. NSGA-II is a commonly used multiobjective optimization metaheuristic that has been shown to outperform RL in finding Pareto-optimal solutions [114]. **Fourth**, a *CE configuration optimizer*, which handles the distribution of the FPGA resources among the CEs and searches for optimal parallelism strategy and dataflow within a CE. As discussed in Section 4.2, adopting a unified parallelism strategy and dataflow across all CEs would lead to suboptimal results. MC**E**xplorer's second module is an *Evaluator*. A fast and accurate evaluation methodology is key to enabling time-efficient DSE. To achieve such a fast evaluation, we use MCCM [130]. MCCM evaluates the performance and resource efficiency of multiple-CE accelerators in a few milliseconds. The flow of the main loop of MC**E**xplorer optimization is depicted in Figure 4.6b. The single or multiobjective optimizer produces a CE arrangement, *i.e.* segments layers, CE-blocks, and the mapping between them. Given this arrangement, the CE configuration optimizer identifies optimal configurations of each CE, producing a complete description of a multiple-CE architecture. Then the evaluator estimates the performance and efficiency of the architecture. The estimation is fed to the optimizer to decide the next point to explore. The modules of MC**E**xplorer and the optimization process are described in detail in the rest of this section.



(a) Optimizers representation of a multiple-CE architecture.



(b) Change operator: two example modifications of a segment.

Figure 4.7: Multiple-CE architecture unified representation and change operator.

4.3.1 Multiple-CE Architecture Representation and Change Operator

Formulating a unified representation of a multiple-CE design allows reusability by different optimizers, which reduces the effort when extending MCExplorer by adding new optimizers. We formulate a unified representation and use it with all our single- and multiobjective optimizers. Figure 4.7a depicts this representation. A multiple-CE architecture is represented as a sequence of *segment* entries; each segment corresponds to a set of DNN layers and a *CE-block* (Section 4.2.3). An entry contains the metadata of the segment layers and CEs. This includes the segment layer's operation types, input and output topologies, and the dependencies between layers. It also includes the compute and memory resources, parallelism strategy, and dataflow of each CE. In the GA and NSGA-II (Section 4.2.6), the sequence of segment entries corresponds to a chromosome, and each entry corresponds to a gene. In the SA (Section 4.2.6.2), the sequence corresponds to a solution, and an entry corresponds to the unit of change during neighbor generation.

We also formulate a unified change operator. The change operator is responsible for mutation in GA and NSGA-II, and for neighbor generation in SA. To facilitate finding near-optimal solutions, the change operator must guarantee that any two multiple-CE architectures in the design space can be transformed into each other in a finite number of steps [12]. The proposed change operator guarantees this. The operator modifies one or more configurations of a segment, where the modification type is determined randomly. The modification types are as follows: **(1)** shrink or expand; this shrinks or expands a segment by changing its layer range, **(2)** modify CE-block, *e.g.*, changing pipelined-CEs to single-CE, **(3)** modify the number of CEs, **(4)** a combination of two or more of the previous three modifications. Note that the modifications are not always applicable. For example, the first layer of the first segment cannot be changed. Moreover, sometimes one modification necessitates the other. For example, when segment layers are shrunk, the number of CEs needs to be modified if it exceeds the new number of layers. Figure 4.7b describes two examples of the change operator. The first example (1) depicts shrinking a segment by reducing the layers from 10 to 6 and changing the CE-block type from single-CE to a pipelined-CEs block with 3 CE. The second example (2) only changes the number of layers. In both cases, the distribution of resources among the CEs needs to be adjusted. The adjustment is handled using the CE configuration optimizer as described in Section 4.3.4.

4.3.2 Single-Objective Optimizers

MCExplorer single-objective optimizers search for optimized multiple-CE accelerators given a custom objective such as throughput, latency, energy per inference, on-chip buffer size, or off-chip access. The rest of the paper focuses on the first three objectives. However, MCExplorer can be used to optimize on-chip buffer size and off-chip access using the same flow. We formulate three optimizers with different design philosophies. This improves the search quality, as the probability of all optimizers converging to local minima is lower, and provides different levels of scalability. The three optimizers include two

metaheuristic-based optimizers, namely Genetic Algorithm (GA) and Simulated Annealing (SA), and a fast heuristic-based optimizer named Hierarchical Mapping (HM).

4.3.2.1 Genetic Algorithm-Based Optimizer.

The proposed GA utilizes the *unified representation* described in Section 4.3.1, where each segment in a multiple-CE accelerator corresponds to a gene, and the sequence of segments corresponds to a chromosome. Although this representation is general and flexible, a main distinction from traditional GA is that both the chromosomes and the genes, in this representation, have variable lengths. To handle variable-length representations, we formulate a specialized crossover operator, which is described later in this section. An individual's fitness is the value of the metric being optimized, *i.e.*, throughput, latency, or energy. Fitness evaluation uses MCCM [130], where the unified representation is first converted to the multiple-CE representation compatible with MCCM. The adopted *termination criterion* is the number of generations. After running for G generations, the algorithm returns the fittest individual, *i.e.*, the best-performing multiple-CE architecture representation. We adopt a one-point *crossover* where two chromosomes of the population, *i.e.*, representations of two multiple-CE architectures, are split around a randomly selected segment, and the offspring are formed by blending the splits. However, unlike the fixed-length chromosome case in traditional GA, simple concatenation of the splits does not guarantee valid multiple-CE architectures. Rather, it can result in omissions or duplications of some layers or whole segments. Hence, we propose a special crossover operator that, in most cases, performs additional modifications on the splits to ensure that the offspring are valid multiple-CE architectures. These modifications take the form of a ripple of modifications of the segments, starting from the segments around the crossover point. Figure 4.8 shows examples of omissions and duplications that occur when applying a crossover over two parents, *M_CE_1* and *M_CE_2*, and the modification used by the special crossover to handle them. These modifications include shrinking or expanding segments by adding or removing layers or changing the number of CEs in a segment, similar to the change operator (Section 4.3.1); or deleting whole segments. Algorithm 2 describes the crossover operator. The function "generateValidOffspring" (lines 11-21) describes the cases that necessitate each of the mentioned modifications. For example, lines 12-13 describe the case of expanding the segment just before the crossover point if there are missing layers. This case is depicted in modification ❶ of Seg 3 in the chromosome *M_CE_21* in Figure 4.8. The loop (lines 15-19) does two types of modifications starting from the segment following the crossover point. First, deleting all segments that are completely formed of redundant layers (line 17). An example of this case is depicted in modification ❷ of Seg 4 in *M_CE_12*. Second, shrinking segments that have redundant layers (line 19). An example of this case is depicted in modification ❷ of Seg 5 in *M_CE_12*. This shrinking may trigger a reduction in the CEs, as depicted in modification ❸ of Seg 5 in *M_CE_12*. Note that after modification ❸, Seg 5 becomes Seg 4 as the original Seg 4 was deleted.

The *mutation* operator uses the change operator described in Section 4.3.1.

Algorithm 2 Single-Point Crossover

Require: Crossover probability P_c, parents p_1, p_2
Ensure: Offspring c_1, c_2

if random() $< P_c$ **then**
    $crossPt \leftarrow$ randInt(max(1, min(numSegments(p_1), numSegments(p_2)) -1)) ▷ Crossover point
    $piece_{11} \leftarrow p_1[0 : crossPt]$
    $piece_{12} \leftarrow p_1[crossPt + 1 : \text{numSegments}(p_1)]$
    $piece_{21} \leftarrow p_2[0 : crossPt]$
    $piece_{22} \leftarrow p_2[crossPt + 1 : \text{numSegments}(p_2)]$
    $offspring_1 \leftarrow$ generateValidOffspring(p_{11}, p_{22})
    $offspring_2 \leftarrow$ generateValidOffspring(p_{21}, p_{12})
else
    $offspring_1 \leftarrow p_1$
    $offspring_2 \leftarrow p_2$ ▷ No crossover
end if
return $offspring_1$, $offspring_2$

---

function GENERATEVALIDOFFSPRING($piece_1$, $piece_2$)
    if missingLayers($piece_1$, $piece_2$) **then**
        $piece_1[-1] \leftarrow$ expandSegment($piece_1[-1]$, $piece_2[0]$) ▷ Expand the last segment of the first piece
    else
        for *segment in* $piece_2$ **do**
            if Layers(*segment*) $\subset$ Layers($piece_1$) **then**
                piece$_2 \leftarrow$ deleteSegment(*segment,* $piece_2$) ▷ Delete a segment if all its layers are redundant
            else if Layers(*segment*) $\cap$ Layers($piece_1$) $\neq \emptyset$ **then**
                piece$_2 \leftarrow$ shrinkSegment(*segment,* $piece_2$) ▷ Shrink a segment if it contains redundant layers
            end if
        end for
    end if
    $offspring \leftarrow$ combine($piece_1$, $piece_2$) ▷ Combining modified pieces guarantees valid offspring
    return R
end function

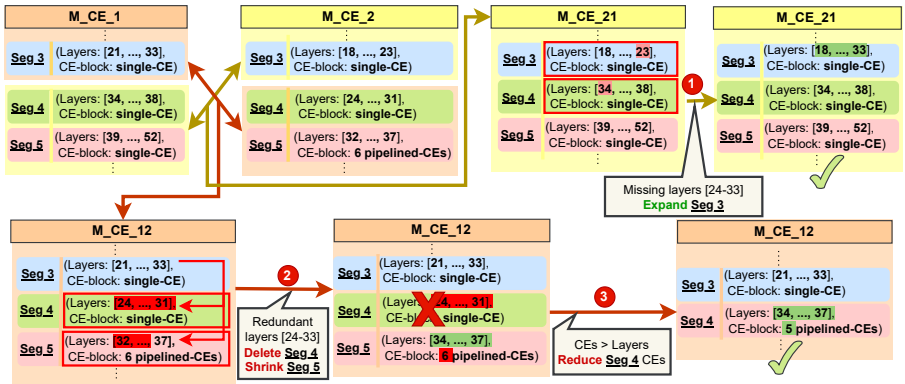


Figure 4.8: An example of the special crossover operator handling of the flexible-length genes (*i.e.* segments) and chromosomes ((*i.e.* multipl-CE architecture representations). The example depicts crossover between *M_CE_1* and *M_CE_2* to produce *M_CE_12* and *M_CE_21*. Initially, neither *M_CE_12* nor *M_CE_21* is a valid mapping, which triggers the depicted set of modifications.

The *replacement strategy* is based on the $(\mu + \lambda)$ model. This means that a combination of a generation (g) and its offspring is used to produce the next generation $(g + 1)$. Regarding the selection operator, we apply tournament selection with elitism. In *tournament selection*, a small group of multiple-CE architectures is randomly chosen from the population, and the one with the highest fitness is selected. This process is repeated until a new generation is formed. The tournament size controls the selection pressure; we use a binary tournament as it maintains a moderate selection pressure. *Elitism* is a strategy in which a certain number of individuals with the highest fitness values, *i.e.* the set of multiple-CE architectures that perform best in the target objective, are automatically carried over to the next generation without being mutated.

4.3.2.2 Simulated Annealing-Based Optimizer.

As in the GA case, we use the unified representation described in Section 4.3.1 for the SA. The adopted termination criterion is the number of iterations. We use the change operator described in Section 4.3.1 for SA *neighbor generation*. In SA, the metric optimized is often referred to as energy (E). Our SA energy evaluation, similar to the GA fitness evaluation, uses MCCM [130] after converting the unified representation to an MCCM-compatible multiple-CE representation. Regarding the *temperature*, we set the initial temperature to the maximum energy difference in a random walk through a set of solutions. We apply an exponential *cooling schedule* because we found it to produce the best results, probably since it balances exploration and exploitation.

4.3.2.3 Hierarchical Mapping.

We propose Hierarchical Mapping (HM) to enable fast and scalable DSE. HM leverages the model's characteristics to prune the design space significantly. A fast exploration becomes crucial in the context of NAS with hardware co-

Algorithm 3 Hierarchical Mapping

Require: minimum clusters C_{min}, maximum clusters C_{max}, DNN representation *dnnConfig*, FPGA resources *hwConfig*, optimization objective *optObjective*
Ensure: multiple-CE accelerator description $mulCE_{best}$
1: Initialize($mulCE_{best}$, *dnnConfig*, *hwConfig*)
2: **for** *numClusters* $\leftarrow C_{min}$ to C_{max} **do**
3: *layerClusters* $\leftarrow$ cluster(*numClusters*, *dnnConfig*)
4: $mulCE_{localBest} \leftarrow$ exploreMappings(*layerClusters*, *hwConfig*, *optObjective*)
5: $mulCE_{localBest} \leftarrow$ refine($mulCE_{localBest}$, *optObjective*)
6: $mulCE_{best} \leftarrow$ best($mulCE_{best}$, $mulCE_{localBest}$, *optObjective*)
7: **end for**
8: **return** $mulCE_{best}$

optimization, where the design spaces of both the model and the accelerator are explored [1, 99]. Algorithm 3 describes the flow of HM. The main idea of HM is to *prune* the design space by creating a coarse-grained representation of a DNN, then do *DSE at the coarse-grained level*, and finally *refine* the architecture at finer granularity.

Pruning the design space. As mentioned in Section 4.2.2, one reason to move from monolithic to multiple-CE accelerators is that DNN layers have varying compute characteristics, which require diverse parallelism strategies and dataflow choices to maintain good performance and efficiency. However, a DNN usually comprises groups of layers with the same or similar characteristics. In other words, the number of layers with distinct characteristics in a DNN is usually much smaller than the total number of layers. As a result, the layers can be grouped into clusters, and a DNN can be represented as a graph of layer clusters, each containing layers of similar characteristics. Such a coarse-grained representation significantly prunes the design space. To give an example, considering the simple case mentioned in Section 4.2.3 of ResNet152 and 10 CEs, where there are roughly 10^{14} possibilities. If ResNet152 layers are grouped into 15 clusters, searching at the cluster level would require ≈ 2 thousand possibilities. In other words, clustering shrinks the design space roughly 50 billion times in this example. HM clustering (line 3 in Algorithm 3) is mainly based on *K-means* clustering [108]. We use four features in the clustering. The first three features describe the *topology* of the FMs and weights. These are *number of channels*, which is common between the FMs and the weights, $IFMs\ width + IFMs\ height$, and $OFMs\ width + OFMs\ height$. Clustering layers based on their FMs and weight topologies facilitates finding CE parallelism strategies that maximize PE utilization. The fourth feature is *weights to FMs size ratio*. This feature aims to group layers based on the type of scheduling that reduces their data movement. The CE-blocks of multiple-CE accelerators have two processing schedules, sequential and concurrent/fused [130]. While the concurrent processing reduces FMs' data movement, it requires more weight data movement [110, 130]. The opposite is the case for sequential processing. Clustering based on weights to FMs' size ratio ensures that, ideally, the layers in one cluster experience minimum data movement when processed by the same CEs (using a common scheduling policy). Other features, such as strides and convolution kernel area, did not significantly improve the quality of clustering. Regarding kernel area, none of

our CE parallelism strategies, described in Section 4.3.4, targets it. Strides would be redundant, as they are implicit in the ratio of the mentioned second and third features.

Coarse-grained exploration. As the pruning step produces clusters of homogeneous layers, each cluster should be efficiently processed by one type of CE-blocks (Section 4.2.3). Hence, the exploration step (line 4), which aims to identify an architecture optimized at the layer-cluster level, needs to effectively find the optimal cluster-to-CE-block mapping. The result of this step transforms each cluster into *segment* in our terminology. The evaluation part of this step utilizes MCCM. Because the pruned design space is usually small, containing only thousands of possibilities, this step applies a brute-force exploration of the full design space. However, the optimal solution at the layer-cluster level is not necessarily the optimal solution at the layer level. Consequently, the output of this step may need further refinement.

Segment refinement. Some factors may result in the cluster-level optimal solution being different from the optimal solution to the original problem. We discuss two such factors. One factor is the discrete nature of CE's PE array scaling, which occurs in quantized steps rather than continuously. This discrete scaling may prevent the CE configuration optimizer (Section 4.3.4) from finding PE distributions that guarantee good resource utilization. This, in turn, affects performance and energy efficiency. A second factor is that segments may need to exchange large intermediate results, which requires either large on-chip buffers or numerous off-chip memory accesses [130]. While large on-chip buffers consume more energy, off-chip accesses are both energy- and time-costly. To mitigate these inefficiencies, HM's segment refinement (line 5) iterates the segments in topological order and explores layer transfers between adjacent segments to further improve the targeted objective by enhancing PE utilization, reducing the required buffering and memory access, or both, while maintaining *segments' contiguity*. For a segment x the transfers that maintain contiguity are: (1) transferring x's first layer to segment $x-1$, (2) x's last layer to segment $x+1$, (3) the last layer of segment $x-1$ to x, (4) the first layer of $x+1$ to x. If no transfers are applied, the exploration moves to the next segment. Otherwise, depending on the applied transfer, the boundaries change, creating new possible transfers that maintain contiguity. Deciding whether to commit a transfer is based on comparing the quality of the original mapping with the one after transfer for each possibility (given the targeted objective) using MCCM, and choosing the best.

HM repeats these three steps with different cluster counts (the main loop in Line 2), where the minimum and maximum cluster counts are user inputs. It eventually returns the description of the multiple-CE architecture with the best results, considering the objective being optimized.

4.3.3 Multiobjective Optimizer

In addition to scenarios where the application domain requires optimizing performance and efficiency simultaneously, the extensive use of DL models makes it always beneficial to seek to optimize the efficiency of DL accelerators. Such extensive use means that even small improvements in training or inference efficiency have a considerable positive net impact. The vast design

space of multiple-CE accelerators contains different CE arrangements that offer comparable performance while varying considerably in terms of efficiency, and vice versa. Multiobjective optimization helps identify accelerators that achieve the best performance and efficiency trade-offs. To identify such accelerators, we formulate a Non-Dominated Sorting Genetic Algorithm II (NSGA-II) meta-heuristic. NSGA-II identifies a well-distributed set of optimal trade-offs, *i.e.* Pareto front [32], as described in Section 4.2.6.3.

In this work, we consider optimizing throughput-energy and latency-energy trade-offs of multiple-CE accelerators. Our NSGA-II uses the same termination criteria, the unified representation, and the mutation operator used in GA (Section 4.3.2.1). However, in our experiments, a higher mutation probability compared to GA (details in Section 4.4) was needed to maintain a Pareto front with a considerable number of unique multiple-CE accelerator architectures. The same special one-point crossover operator described in Section 4.3.2.1 is applied in NSGA-II to handle the flexible-length unified representation. Regarding the replacement strategy, we use $(\mu + \lambda)$, where a combination of a generation of multiple-CE accelerators and their offspring is used to produce the next generation. Regarding the selection mechanism, we apply the NSGA-II standard non-dominated sorting and crowding distance selection (Section 4.2.6.3).

4.3.4 CE configuration optimizer

The main goal of the optimizers described in Sections 4.3.2 and 4.3.3 is to identify optimal CE arrangements, *i.e.* CE-block types and DNN segment-to-CE-block mapping. However, given a single arrangement of CEs, there are numerous per-CE *hardware resource distributions*, *parallelism strategies*, and *dataflow options*. As discussed in Section 4.2, adopting a uniform CE configuration, *i.e.* parallelism and dataflow, is not optimal. Hence, there is a need for mutual optimization of the CEs' arrangement, on one hand, and the resource distribution and per-CE configurations, on the other. MCExplorer's module is responsible for the second, which is *CE configuration optimizer*. CE configuration optimizer has two main tasks: **(1)** deciding the HW resources distribution among the CEs, **(2)** searching for optimized per-CE configurations. These tasks must be performed at each optimizer iteration as depicted in Figure 4.6b.

The *distribution of the HW resources* on the CE-blocks and then on the CEs aims to balance the resources among the CEs proportional to their workloads. CE configuration optimizer applies the resource-balancing heuristic adopted by state-of-the-art multiple-CE accelerators [9, 128]. This heuristic has low time complexity and is practical to apply to each explored multiple-CE architecture. Unlike resource distribution, searching for optimal per-CE parallelism strategy and dataflow is in itself a separate optimization problem with a large design space [72, 74, 111, 122]. As a result, applying state-of-the-art techniques to each explored multiple-CE architecture is not an option. Fortunately, there are a few well-known parallelism strategies, such as those of NVDLA [118], DPU [176], and ShiDianNao [37], each of which performs well for certain DNN layer types. When combined, these strategies cover all computational requirements of DNN layer types, offering good performance and efficiency.

The same applies to dataflow. A combination of the three mentioned dataflow variants (Section 4.2.1) and Output Stationary-Local Weight Stationary (OS-LWS) and Weight Stationary - Local Output Stationary (WS-LOS) [171] can capture the minimum data movement for most DNN layers. Given the fact that a combination of a relatively small set of parallelism strategies and dataflow options achieves highly optimized results, our CE configuration optimizer strategy is to explore these combinations exhaustively.

Deciding *parallelism strategy* has two components: **(1)** deciding the number of CE parallelism levels and the specific dimensions to parallelize, **(2)** deciding the degree of parallelism in each of the dimensions. Regarding the levels of CE parallelism and dimensions, the CE configuration optimizer explores a set of commonly used options, including NVDLA, DPU, and ShiDianNao-like parallelism. Regarding the degree of parallelism, it conducts an exhaustive search while considering only the degrees of parallelism on each dimension that perfectly divide the inputs, weights, or outputs on that dimension to avoid PE idleness. The combination that achieves the best results is selected. The optimizer decides on the *dataflow* similarly, by exhaustively exploring the mentioned dataflow options and selecting the one that maximizes spatial and temporal data reuse, leading to minimized data movement.

4.4 Experimental setup

4.4.1 Platforms and Systems

Table 4.1 shows the FPGA boards we use in the evaluation. The table mainly shows the PEs (DSPs), on-chip memory (Block RAM), and off-chip bandwidth. The boards used represent various resource budget points. There are order-of-magnitude differences in their DSPs and on-chip memory[1]. We use high-level synthesis (HLS) to implement and validate a sample of the multiple-CE architectures suggested by MC**E**xplorer. Vitis-IDE (2021.2) is used for synthesis and validation. MC**E**xplorer heuristics are implemented mainly in Python. Our experiments are conducted on a machine with 16 logical cores, Intel(R) Core(TM) i7-10700 CPU @ 2.90GHz.

Table 4.1: Evaluation FPGA boards

	ZC706	VCU108	ZCU102	VCU118
DSPs	900	768	2520	6840
Block RAM(MiB)	2.4	7.6	4	43.2
Off-chip memory BW (GB/s)	3.2	19.2	19.2	19.2

4.4.2 Workloads

Table 4.2 summarizes the models used in the evaluation. We evaluated MC**E**xplorer using a representative set of Convolutional Neural Networks (CNNs) that have varying structures, depths, and weight sizes. These are

[1]The on-chip memory values are reported in MiB rather than Mb, which is commonly used by AMD

ResNet152 and ResNet50 [50], XCeption [21], DenseNet [58], and MobileNetV2 [142]. These CNNs are extensively used in the literature, and their core blocks are reused in many recent CNNs. To give an example, the core block of MobileNetV2, known as MBConv, is used in MnasNet [159] and EfficientNet [160]. The residual blocks of ResNets and the Extreme Inception blocks of XCeption are also used in recent CNNs. We also evaluate MC**E**xplorer using models with heterogeneous backbones. These are: CMT [47], a hybrid architecture that combines CNNs with transformer-based modules in a unified backbone, and SmAt-UNet [162], which is based on the UNet architecture equipped with attention modules and depthwise-separable convolutions.

Table 4.2: Evaluated DL models

Model	Abbreviation	Weights (M)	Conv layers
ResNet152	Res152	60.4	155
ResNet50	Res50	25.6	53
XCeption	XCp	22.9	74
DenseNet121	Dns121	8.1	120
MobileNetV2	MobV2	3.5	52
CMT (small)	CMT	25.1	157
SmaAt-UNet	SmAt-U	4.1	42

4.4.3 Baseline Architectures

We evaluate MC**E**xplorer in comparison to two types of baselines. *First*, we conduct a somewhat generic evaluation against baseline accelerators that represent distinct multiple-CE accelerator categories. Multiple-CE accelerators in the literature can be placed into one of three categories based on the use of *single-CE* and *pipelined-CEs* blocks (Section 4.2.3) [130]. The first category, known as *Hybrid*, uses an ordered combination of pipelined-CEs and single-CE blocks. The second, *Segmented*, uses only single-CE blocks. The third, *SegmentedRR*, uses only pipelined-CEs blocks. As works in the literature representing these categories use different models and FPGA boards, there is a need to reproduce their results using common settings. We use MCCM [130] to model the following three state-of-the-art accelerators, representing the three categories, using a common set of models and boards:

- **FiBHA** [128]. There is a handful of hybrid FPGA-based accelerators in the literature. These include Alwani *et al.* [2], DNNExplorer [199], Nguyen *et al.* [117], and FiBHA. Our Hybrid baseline uses FiBHA as it is the most recent and covers a wider range of CNNs by supporting depthwise convolutions.

- **Multi-CLP** [150]. The Segmented architecture is, to our knowledge, the least common in the literature. We target Multi-CLP as it is the classic and widely referred work on Segmented FPGA-based accelerators. Multi-CLP's CEs apply 2D parallelism across channels and filters.

- **TGPA** [174]. Several accelerators in the literature can be categorized as SegmentedRR [9, 14, 168, 174, 197]. Our SegmentedRR baseline tiling and buffer granularity modeling are based on TGPA. Regarding CEs, we

apply a 2-D parallelism as the original work uses a systolic array-like 2-D parallelism.

To put our results in perspective, the *second* form of comparison is a direct one with results reported in previous work. Side-by-side comparison is challenging as different works use different FPGAs and apply various low-level optimizations. To provide a relatively fair comparison in such cases, the related work that compares accelerators on different FPGAs uses different variants of efficiency metrics, like performance density [115, 174] or DSP efficiency [68, 199]. We use a performance density (PD) metric defined in Equation 4.1.

$$\text{PD} = \frac{\text{GOP/s}}{\textit{Frequency} \times \textit{number of DSP Slices}^{3} \times \alpha} \tag{4.1}$$

As some state-of-the-art use *operand packing* optimization to pack 1, 2, or 4 operands depending on the precision to perform more operations per DSP per cycle, while others do not, α normalizes the OP/s considering operand packing. In addition to the architectures included in the first comparisons, in Section 4.5.5 we compare with two DSE frameworks which achieve highly optimized performance and efficiency, namely fpgaConvNet [168] and SPA [14]. In general, all baseline architectures are made up of different combinations of the CE-blocks discussed in Section 4.2.3. Due to space limitations, we refer the reader to the corresponding papers for a detailed description of each baseline architecture.

4.4.4 Performance and energy consumption

We measure the performance of multiple-CE accelerators in terms of throughput and latency, and the efficiency in terms of energy-per-inference. The latency results are for a batch size of 1. Following the literature, energy consumption is estimated as the sum of the energy consumed by the PE array, registers, on-chip buffers, and off-chip accesses [111, 122, 187], as these dominate the overall energy consumption. To estimate energy efficiency, we use MCCM [130] to obtain accelerator PE utilization, on-chip buffer sizes, and the number of accesses to all memory levels. We utilize Xilinx Power Estimator (XPE) [183] to obtain the power consumption of the DSPs, registers, and on-chip buffers. The energy per off-chip access is calculated based on the DRAM datasheets of the boards used, following the methodology described in [112].

4.4.5 Optimizers Key Parameters

In **GA**, we use a population size of 400, and the number of generations is 50. The mutation probability is set to 10% and the crossover probability to 90%. We use a binary tournament; therefore, the tournament size is 2. The top 1% of the population is considered elite. In **SA**, the number of main iterations is set to 20, and the number of iterations per temperature is set to 1000. Hence, the SA terminates after 20000 iterations. The initial temperature is set to the

[5] Some literature uses the number of allocated DSPs only. However, using the board DSPs is fairer since all our baselines target scaling and maximizing resource utilization to deliver the best performance. Using only allocated DSPs ignores the ability of some to scale and utilize DSPs better than others.

maximum energy difference in a random walk across 100 solutions. SA applies an exponential cooling schedule with an *alpha* of 0.95. **HM** requires the user to provide only the minimum and maximum number of clusters (Section 4.3.2.3). We set these two parameters to 2 and 8 in all experiments. In **NSGA-II**, we use a population size of 400, and the number of generations is 50. Mutation probability is set to 40%, and crossover probability to 90%. The mutation probability is considerably higher than in GA. We needed a high mutation probability in addition to NSGA-II's crowding distance mechanism to maintain a diverse Pareto front of solutions. Throughout the evaluation section, we compare the results of different optimizers. Hence, we set the number of explored data points to 20000 for all metaheuristics (400×50 in the case of GA and NSGA-II). GA and SA results are averages of 10 runs with different random seeds, and in Section 4.5.1.3 we report a summary of the variance in their results. Regarding NSGA-II, the variance was negligible, as NSGA-II mechanisms to maintain Pareto front diversity help reduce its sensitivity to different initializations.

4.5 Evaluation

4.5.1 Single-Objective Optimization: Throughput, Latency, and Energy-Efficiency

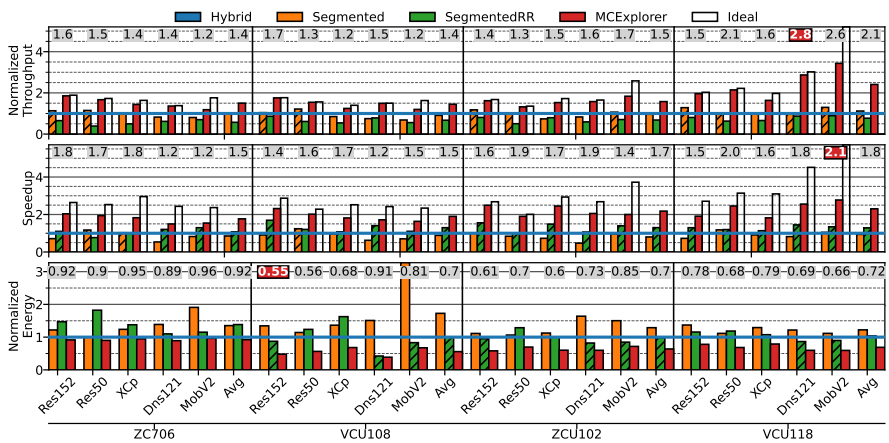


Figure 4.9: Throughput, latency, and energy efficiency comparison with the baselines. Hatched baseline bars indicate the best among the three baselines. Experiments with no hatched baseline bars are those where Hybrid (the normalization anchor) is best among the baselines. The labels at the top of the figures report MC**E**xplorer results normalized to the best of the three baselines in an experiment.

4.5.1.1 Using MCExplorer as a Black-Box Optimizer.

This section analyzes the results of running MC**E**xplorer single-objective optimizers as a black box where GA, SA, and HM explore the design space in parallel

and MCExplorer produces the best solution. We compare MCExplorer results with the three baselines described in Section 4.4.3. We also include the theoretical *ideal performance* where it is assumed that the PE utilization is 100%, the memory bandwidth is infinite, and the access latency to all memory levels is one cycle. The ideal performance is not always practically achievable, but it gives an upper bound.

Figure 4.9 compares the throughput, latency, and energy efficiency achieved by MCExplorer suggested accelerators to Hybrid, Segmented, and SegmentedRR baselines described in Section 4.4.3. MCExplorer finds accelerators that outperform the baselines in all experiments. Compared to the best baseline, MCExplorer achieves up to 2.8× *throughput* improvement. For Res50, Res152, and Dns121, MCExplorer results are very close to the ideal. For MobV2 and XCP, the results are not as close to the ideal, as these models contain depthwise convolutions, which are usually memory-bound. Moreover, they have heterogeneous convolutions, making a perfect balance of the workloads of different CEs unattainable given the available PE budgets [128]. The speedup is up to 2.1× compared to the best baseline. The speedups are not very close to the ideal. This is not specific to MCExplorer; in general, close-to-ideal speedups are usually not practically feasible. One reason for the differences is the absence of batch-level parallelism, which limits common parallelism among layers (batching is not used when measuring latency). Moreover, not all CEs are active from a single input perspective throughout an inference, causing pipeline filling and draining overhead [12, 130, 201]. Considering *energy efficiency*, MCExplorer suggested accelerators use down to 55% of the energy of the best baseline saving up to 45%. Energy reduction on ZC706 is low as its relatively small number of PEs and on-chip memory give little room to improve energy efficiency by optimizing PE utilization, on-chip buffer sizes, or off-chip accesses.

Table 4.3: CE-blocks and per-CE configurations heterogeneity.
(a) Summary of results of the best-performing accelerators in the 60 experiments discussed in Section 4.5.1.1

		CE Configurations (parallelism strategy and dataflow)		
		Homogeneous	Heterogeneous	Sum
CE-blocks	Homogeneous	1 (1.67%)	19 (31.67)%	20 (33.33%)
	Heterogeneous	0 (0%)	40 (66.67%)	40 (66.67%)
	Sum	1 (1.67%)	59 (98.33%)	60 (100%)

(b) Per-model breakdown of CE-block types (12 per-model experiments using 4 boards×3 metrics). S stands for single-CE, Pa for parallel-CEs, and Pi for pipelined-CEs.

Model	CE-block types	Experiments
Res152	[S]; [Pa,S]; [Pa]; [Pi,S]	3/12; 7/12; 1/12; 1/12
Res50	[S]; [Pa,S]; Pa	5/12; 6/12; 1/12
XCp	[Pi,S]; [Pa,Pi,S]	10/12; 2/12
Dns121	[S]; [Pa,S]; [Pi,S]; [Pa]; [Pi]; [Pa,Pi,S]	6/12; 2/12; 2/12; 1/12; 1/12; 1/12
MobV2	[Pi,S]; [Pi]	9/12; 2/12

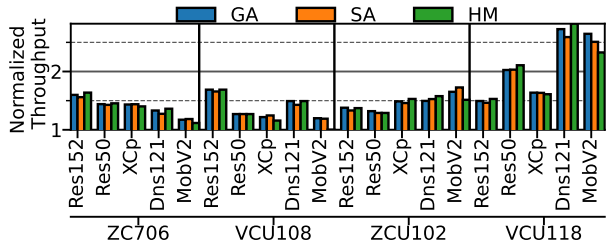
4.5.1.2 Heterogeneity of CE Arrangements and CE Configurations Among MCExplorer Results.

MC**E**xplorer explores a wider design space compared to state-of-the-art multiple-CE accelerators in two ways. First, it does not restrict the inter-CE arrangements. Secondly, it explores distinct per-CE configurations, including parallelism strategies and dataflow options. Table 4.3a summarizes the contributions of these two differences in the sixty experiments depicted in Figure 4.9. In only one of the sixty experiments, the best-performing accelerator has one type of CE-blocks and all CEs have the same configurations (*i.e.* dataflow and parallelism strategies). This shows that restricting both CE arrangements and per-CE configurations is usually suboptimal. In nineteen of the experiments, the best-performing accelerators have one type of CE-block, but the CEs have different configurations. Since these results isolate the impact of CE configurations, they demonstrate the effectiveness of the heuristics adopted in our *CE configuration optimizer* (Section 4.3.4). The breakdown in Table 4.3b shows that most homogeneous architectures are composed of Single-CE blocks. In two-thirds of the experiments, the best-performing accelerators have both different CE arrangements (CE-block types) and per-CE configurations as shown in Table 4.3a. This shows that in most cases, flexibility in both CE arrangements and per-CE configurations achieves the best results. The breakdown in Table 4.3b shows that the CE-block types in most of these cases are combinations of single-CE and parallel-CEs or single-CE and pipelined-CEs. MobV2 and XCp have a higher rate of cases where block types are heterogeneous. The fact that these models contain heterogeneous layer types (Section 4.5.1.1) explains this trend. Regarding per-CE configurations, 55 dataflow and parallelism strategy combinations are explored. 37 of the 55 were used at least once, and no single combination dominates considerably. The combination most commonly used appears 9% of the time.

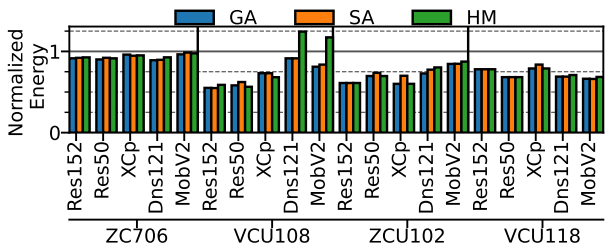
4.5.1.3 Comparing Single-Objective Optimizers.

Figure 4.10 compares the throughput and energy efficiency of multiple-CE accelerators suggested by GA, SA, and HM. The reason for equipping MC**E**xplorer with GA and SA is that both have achieved high-quality results on related problems [12, 74, 139, 164, 201]. As Figure 4.10 indicates, GA achieves better or equally good results in most cases. However, there are cases where the application of SA was beneficial, such as the throughput of MobV2 in ZCU102 in Figure 4.10a. Moreover, although both explore the same number of data points (Section 4.4.5), SA is considerably faster. This is shown in Table 4.4, which presents the execution times of the three heuristics. On the other hand, as Table 4.5 shows, GA is more stable and less sensitive to initialization than SA. Both the average and maximum variance of GA are less than those of SA. For example, the average variance of GA ranges from 0.4% to 2%, while that of SA ranges from 1% to 4%.

A DSE framework should provide a fast and scalable exploration option. This is the reason for equipping MC**E**xplorer with HM. By pruning the design space, HM is faster and scales better than GA and SA. Table 4.4 shows that HM is at least an order of magnitude faster than GA and SA. DSE speed and scalability become crucial when the targeted DNN models have thousands of



(a) Throughput comparison of different optimizers.



(b) Energy per inference comparison of different optimizers.

Figure 4.10: Comparison of (a) throughput and (b) energy per inference of different optimizer-suggested accelerators, normalized to the best of Segmented, SegmentedRR, and Hybrid.

Table 4.4: Optimizers execution times in minutes.

	Res152	Res50	XCp	Dns121	MobV2
GA	186.1	58.3	89.9	116.8	52.1
SA	53.2	17.1	22	34	16.8
HM	1.5	0.8	1.6	1.4	1.1

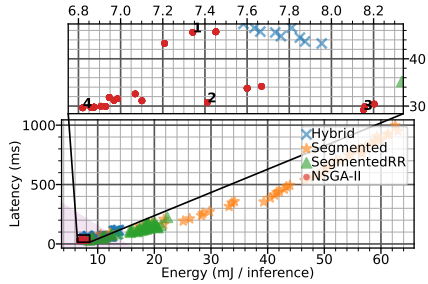
layers [51, 59], or when used in NAAS in the context of a NAS with hardware co-optimization, where the design space expands as both the DNN and the accelerator architectures are explored. Regarding the quality of HM results, Figure 4.10 shows that HM found multiple-CE architectures that outperform the best baseline considering all DNNs and boards. There are a few cases where HM results are similar or have negligible improvement compared to the state-of-the-art. Moreover, with few exceptions, HM results are close to those of GA and SA. The memory requirements of the optimizers are very small across different models. Note that the optimizers do not need the model's parameters; they take as input a DAG of the model metadata whose size is in KiBs.

To summarize, MC**E**xplorer identifies accelerators that outperform the state-of-the-art using different boards, DNNs, and metrics. Moreover, the fact that the three exploration methods find architectures that improve over the state-of-the-art in almost all experiments demonstrates that restricting the design space leads the state-of-the-art to miss promising architectures. Finally, having different optimizers provides scalability and flexibility to prioritize

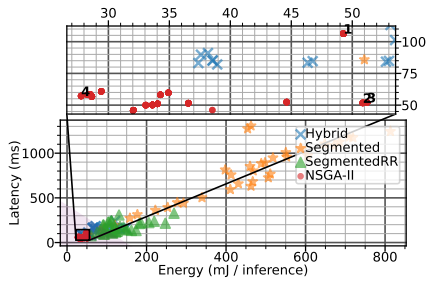
Table 4.5: Optimizers' coefficient of variation (CV) across 10 runs.

	Throughput		Latency		Energy	
	Avg CV	Max CV	Avg CV	Max CV	Avg CV	Max CV
GA	2%	6%	0.4%	5%	1%	4%
SA	2%	6%	1%	6%	4%	7%

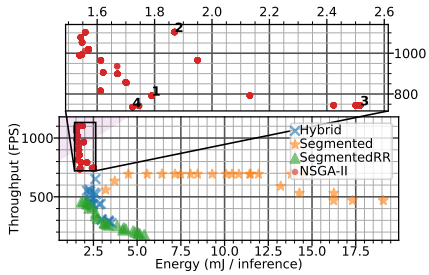
time-to-solution or quality of results by reducing the chances of getting a local minimum.



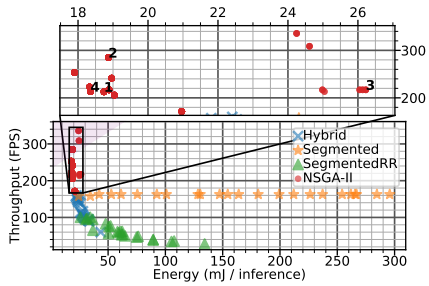
(a) Latency energy trade-off using ZC706 and Dns121.



(b) Latency energy trade-off using VCU108 and XCp.



(c) Throughput energy trade-off using ZCU102 and MobV2.



(d) Throughput energy trade-off using VCU118 and Res50.

Figure 4.11: Performance-energy trade-off and Pareto front results of NSGA-II. The Pareto fronts are scaled, as many points have relatively close latency and energy efficiency values. The annotated points in the zoomed figure represent two pairs of architectures ($1 \rightarrow 2$, $3 \rightarrow 4$) that have sharp trade-offs between the two metrics. The shaded corners designate the best in both metrics.

4.5.2 Multiobjective Optimization: Performance-Energy-Efficiency Trade-offs

Figure 4.11 shows some of the performance-energy-efficiency trade-off exploration results using different DNNs and boards. The figure shows the performance and energy efficiency of both the points on the Pareto front identified by NSGA-II search and those of the baseline accelerators. Examining the Pareto fronts (the red circles) shows that considerable energy reductions can be achieved at the cost of negligible performance loss. In other words, a more energy-efficient inference is possible almost for free. For example, in

Figure 4.11b, architecture **4** reduces the energy consumption by 45% compared to **3** while incurring a negligible increase in latency by only 2%. Similarly, the figures show that considerable performance gains can be achieved at the expense of an insignificant increase in energy consumption. For example, in Figure 4.11d, architecture **2** improves throughput by up to 34% at the cost of only an increase of 0.7% in energy consumption compared to **1**. The figures show other pairs of architectures with sharp trade-offs, which highlights the importance of multiobjective optimization as it allows identifying architectures that achieve considerable gains in one metric without noticeable losses in the other.

To put NSGA-II results in perspective, we compare them with the three baselines. As can be seen in Figures 4.11a-4.11d, the Pareto fronts contain architectures that offer better trade-offs than the three baselines. Comparing the results on the Pareto front with the baselines and comparing the baselines against each other suggests that more heterogeneous and flexible architectures have better trade-offs. As the figures show, Segmented architectures always have the worst energy efficiency, and the SegmentedRR architectures have the worst throughput. The reason for these trends is that Segmented and SegmentedRR use only one type of CE-blocks (Section 4.2.3), *i.e.* single-CE in Segmented and pipelined-CEs in SegmentedRR (Section 4.4.3). Each of these blocks has its performance and efficiency bottlenecks when considering DNN layers of certain computational characteristics. These bottlenecks worsen when scaling the number of CEs. By contrast, Hybrid architectures use both single-CE and pipelined-CEs to match the variations in the DNN layers' characteristics. Hence, they maintain better trade-offs. However, Hybrid architectures still restrict the CE arrangements; they use the two blocks in a fixed and predefined order. NSGA-II exploration, which restricts neither the CE arrangements nor per-CE configurations, achieves better trade-offs than the Hybrid. The trends in heterogeneity of the CE arrangements and per CE-configurations of the accelerators on the Pareto front are similar to those reported in Table 4.3.

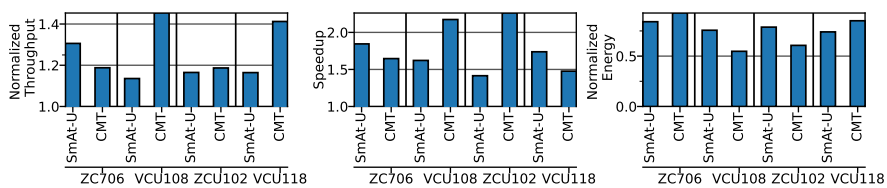


Figure 4.12: Throughput, latency, and energy efficiency of MC**E**xplorer suggested accelerators normalized to the best of Hybrid, Segmented, and SegmentedRR using the convolutional backbones of models with heterogeneous backbones.

4.5.3 Using MCExplorer with DL Models of Heterogeneous Backbones

The evaluation of MC**E**xplorer in the previous sections focused on the design of accelerators for CNNs with pure convolutional backbones. Nevertheless, MC**E**xplorer is also useful for modern DL models of heterogeneous backbones.

When used with models of heterogeneous backbones, MC**E**xplorer can search for optimized CE-block arrangements to process the convolutional parts of these models. Figure 4.12 compares MC**E**xplorer suggested accelerators with the best baseline using the convolutional parts of two models with heterogeneous backbones, namely CMT and SmAt-Unet. As the Figure shows, MC**E**xplorer accelerators improve over the baselines in all cases, considering different metrics and boards. These results demonstrate that the applicability of MC**E**xplorer extends beyond CNNs.

4.5.4 Implementing and Validating MCExplorer Results

The accuracy of MCCM [130], the cost model used in our DSE, is validated compared to HLS and is shown to be greater than 90%. As the multiple-CE accelerators from our DSE use similar CE-blocks, the estimation accuracy is expected to be within the same range. To verify that, we implement a few accelerators and compare the estimated improvement with the actual improvement, considering different metrics. The estimation accuracies are reported in Table 4.6. As the table shows, the estimated improvements are within 9% of the actual ones. While the actual speedup and throughput improvements are greater than the estimated ones, the estimated energy reduction is slightly lower than the actual one. The accelerators' CE-blocks in the table show that their architectures do not belong to any of the three state-of-the-art categories (Section 4.4.3). The accelerator that achieves the best throughput improvement comprises both single-CE and pipelined-CE blocks. Although it is similar to the Hybrid (Section 4.4.3) in terms of CE-block-types, the order of the CE-blocks is different. The accelerator that achieves the best speedup and energy reduction uses a parallel-CEs block (Section 4.2.3), which is not used by the state-of-the-art.

Table 4.6: The accuracy of the estimated improvements compared to the measured improvements using HLS. The results are for the accelerators that achieve the best speedup, throughput, and energy efficiency, respectively, using ResNet152 on ZCU102.

	Throughput improvement	Speedup	Energy reduction
Estimated	1.37	1.60	0.39
Actual	1.51	1.72	0.37
Accuracy	91%	93%	95%
CE-blocks	{single-CE, 2 pipelined-CEs, single-CE, single-CE}	{2 paralllel-CEs, single-CE}	{2 paralllel-CEs, single-CE}

4.5.5 Direct Comparison with Existing Multiple-CE Accelerators and DSE Frameworks

In this section, we compare MC**E**xplorer results directly with the state-of-the-art multiple-CE accelerators using their reported results to augment the comparisons in previous sections. As discussed in Section 4.4.3, we use Performance Density (PD) (Equation 4.1) as a normalizing metric. Table 4.7 shows the PD achieved by MC**E**xplorer compared to the state-of-the-art using models

Table 4.7: Direct Performance Density (PD) comparison Multiple-CE Accelerators and DSE Frameworks.

Model	Mob_v2					Res152				
Design	FiBHA [128]	SPA [14]		MCExplorer		fpgaConvNet [168]	TGPA [174]	SPA [14]	MCExplorer	
Precision	8-bit	8-bit	8-bit	8-bit	8-bit	16-bit	16-bit	8-bit	8-bit	8-bit
Device	ZCU102	7Z045	KU115	ZC706	VCU118	Z-7045	VU9P	KU115	ZC706	VCU118
DSP slices	2520	900	5520	900	6840	900	6840	5520	900	6840
Freq (MHz)	180	200	200	200	200	125	200	200	200	200
(GOP/s) / α	375	242*	1594*	405	3092	188	1463	1583*	2655	353
PD	0.83	1.34	1.17	1.35	**1.36**	1.67	1.07	1.43	1.94	**1.96**

* $\alpha = 2$, as SPA [14] applies packing of two activations together.

common among these works, namely MobV2 and Res152. MC**E**xplorer accelerators are either on par with or outperform the state-of-the-art presented in the table, considering different PE resource budgets (DSP). In the case of MobV2, the improvements range from 64% over FiBHA to roughly 1% over the best of SPA's. In the case of Res152, the improvements range from 17% over fpgaConvNet to 83% over TGPA. These results show that MC**E**xplorer identified accelerators that better utilize the PEs, resulting in higher performance densities.

4.6 Related Work

The performance and efficiency of single Compute Engine (CE) DNN accelerators have improved over time as a result of the introduction of novel architectures and the application of a wide range of optimizations targeting data flow and scheduling, reduced-precision arithmetic, and the use of structured and unstructured sparsity [19, 71, 172]. However, these improvements have plateaued due to the inherent bottlenecks of single-CE accelerators [172]. Consequently, to maintain improvements in performance and efficiency, many state-of-the-art DL accelerators are adopting multiple-CE architectures [12, 14, 43, 145, 150, 170, 174]. Due to their reconfigurability, which enables arranging resources into a custom number of dedicated CEs, FPGAs are commonly used platforms in designing multiple-CE accelerators. One class of FPGA-based multiple CE accelerators comprises fully pipelined and layer-specific CEs, where each core DNN layer has its own CE [9, 197]. This approach is tightly coupled and highly dependent on the structure and depth of the DNN model. This limits its performance and scalability. A more flexible and scalable class of multiple-CE accelerators adjusts the number of CEs considering the DNN structure, the available hardware, and the optimization objectives [2, 14, 117, 128, 150, 174, 199]. Having an adjustable number of CEs enables the design of DNN model-aware accelerators at different scales, leading to improved performance and efficiency.

Although it comprises an adjustable number of CEs, previous work adopts fixed CE arrangements and usually applies a single data flow and parallelism

strategy, with different degrees of parallelism, in all CEs [150, 174, 197, 199]. In other words, the previous work constrains different design aspects and does not adequately explore the design space of multiple-CE accelerators [130]. Moreover, previous work usually optimizes for a single performance metric, *e.g.,* latency [174] or throughput [9, 150], and discusses improvements in other metrics only as a byproduct. To the best of our knowledge, there is no generic multiple-CE accelerator design framework that enables unrestricted DSE and optimization for custom metrics.

In this work, we propose a framework for exploring the design space of multiple-CE accelerators, namely MC**E**xplorer. Using MCCM fast evaluation [130], MC**E**xplorer effectively explores a design space beyond that of previous work, identifying multiple CE accelerators with better performance and efficiency.

4.7 Conclusion

In this chapter, we presented MC**E**xplorer, a framework for exploring the design space of FPGA-based multiple-CE DL accelerators. MC**E**xplorer is equipped with a set of heuristics that optimize for latency, throughput, energy efficiency, and trade-offs among these objectives. We evaluated MC**E**xplorer using representative DNNs and FPGAs with varying resource budgets. The evaluation demonstrates that restricting the design space by adopting fixed CE arrangements and per-CE configurations prevented the state-of-the-art from achieving highly optimized results. It shows that, by exploring flexible CE arrangements and per-CE configurations, MC**E**xplorer can identify accelerators that outperform the state-of-the-art in all targeted optimization objectives, achieving up to 2.8× throughput improvement, 2.1× speedup, and 45% energy reduction. Moreover, the evaluation demonstrates that such a comprehensive exploration is crucial to identifying multiple-CE accelerators with the best performance and efficiency trade-offs.

Chapter 5

Multi-Engine DL Accelerator Design Methodology

The previous chapters presented models, tools, and methodologies for designing model-specific multi-engine DL accelerators. This chapter switches to flexible accelerators. Unlike model-specific accelerators, flexible accelerators aim to optimize for diverse DL workloads [13, 70, 86, 145]. Consequently, accelerator engines must maximize the coverage of a diverse workload landscape. This chapter introduces a two-stage Multi-Engine DL accelerator Design Methodology (MEDEM). MEDEM first constructs a pool of engines tailored to distinct computational characteristics of the targeted workloads' layers, and then selects a combination of a subset of engines that maximizes workload coverage under resource constraints. Despite the exponential complexity of the design space, the proposed methodology employs efficient heuristics, making exploration tractable and practical.

5.1 Introduction

As monolithic Deep Learning (DL) accelerators face diminishing returns from further optimization, multi-engine accelerators offer a more scalable alternative and open up new design opportunities [43, 45, 86, 172, 201]. Hence, they are increasingly adopted both in research and in industry. Representative research-driven multi-engine accelerators include Simba [144], Tangram [43], Maelstrom [86], and Gemini [13]. Representative accelerators widely deployed include Meta's MTIA [25], Tesla's Dojo [158], and Google's TPU v4[1] [70].

Multi-engine DL accelerators comprise independently controllable computing engines interconnected through a high-bandwidth communication fabric, where each engine mainly consists of an array of Processing Elements (PEs), as well as local and global buffers [12, 13, 29, 43, 86, 144, 201]. These accelera-

[1]The TPU architecture evolved from a single systolic array in v1 to multiple TensorCores, each integrating four matrix multiplicative units (MXUs), in v4.

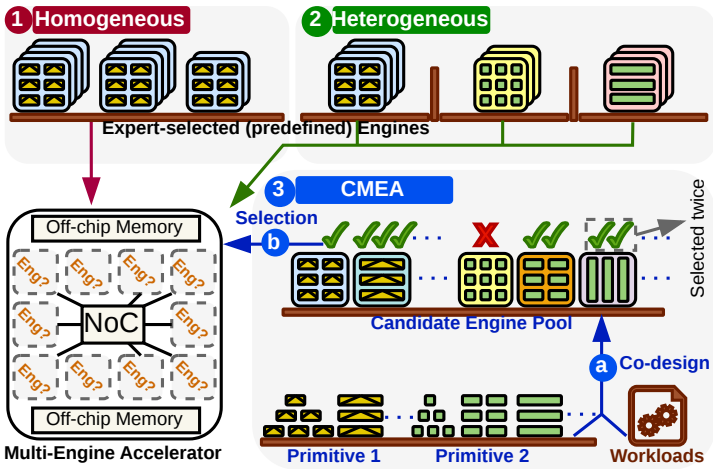


Figure 5.1: Engine design alternatives in multi-engine DL accelerators: 1 homogeneous engines 2 heterogeneous engines and 3 co-designed engines (CMEA).

tors can be *model-specific* or *flexible*. Model-specific ones typically leverage reconfigurable hardware and are optimized for a single or a few very similar workloads (models) [49,131,148]. In contrast, flexible accelerators are optimized for diverse DL workloads [13, 43, 86]. We classify existing flexible multi-engine DL accelerators, depending on engine heterogeneity, into the two categories depicted in Figure 5.1: 1 homogeneous accelerators, which use the same hardware configurations across all engines [13, 144]; and 2 heterogeneous accelerators, in which engines differ at the architectural or microarchitectural level [29, 86, 120].

Existing flexible multi-engine DL accelerator design approaches combine *independently optimized expert-selected* engines such as Eyeriss [18, 19], ShiDiaNao [37], or NVDLA [118]. These approaches exhibit several limitations. **First**, although expert-driven selection can produce optimized accelerators for particular scenarios, *it lacks a robust and reusable methodology*. A reusable methodology is needed as the fast pace of DL model evolution requires continuous accelerator co-design to sustain performance and efficiency [138]. Furthermore, DL workloads span a broad range of deployment environments and resource constraints; designs tailored for edge settings may not translate effectively to cloud-scale systems [86]. **Second**, due to the exponentially large DL accelerator design space, *systematic methodologies consistently outperform expert-driven ones*, as demonstrated in single-engine design where improvements range from two times to an order of magnitude [31, 56, 77, 122, 138, 195]. **Third**, independently designed engines may become *optimized for the same or substantially overlapping subsets of workloads*, resulting in poor coverage of the overall workload landscape. We define workload *coverage* as an accelerator's ability to process a set of workloads with minimal aggregate execution cost (such as execution time or energy consumption).

To overcome these limitations, this work proposes Multi-Engine DL accelerator DESign Methodology (MEDEM). The idea behind MEDEM is to decouple engine co-design and selection as highlighted in Figure 5.1. MEDEM's **first**

stage (a) aims to *co-design a pool of candidate engines* that conceptually achieve a maximal coverage of diverse workloads, by co-designing an engine for each kernel with distinct computational characteristics. However, in practice, resource constraints often limit the number of engines that can be included. Hence, MEDEM's **second stage** (b) *selects a multiset* [2] of engines (from the pool) that maximizes workload coverage given a practical resource budget. We call accelerators designed using this methodology **Co-designed Multi-Engine Accelerators (CMEAs)**. On the one hand, co-designing a relatively large pool of engines boosts the chance of identifying a combination of engines with complementary capabilities, hence achieving maximum coverage. On the other hand, it requires very efficient co-design and selection strategies to navigate exponentially large design spaces.

Engine co-design entails exploring two exponentially large and non-convex subspaces. These are the hardware space and the mapping space [53, 56]. The state-of-the-art hardware-mapping co-design approaches use techniques ranging from simple to sophisticated DL-based ones [31, 53, 56, 60, 77, 138, 180]. However, as important as the techniques themselves, if not more, is effectively navigating the design space [31, 60, 180]. The engine co-design stage of MEDEM proposes a **Hierarchical Co-designer (HCo)** to navigate the hardware-mapping space effectively. The idea is to *prioritize* certain axes of the space - such as PE arrangements or tiling - based on their *impact* on the execution cost, regardless of whether they belong to the hardware or mapping space. As prioritized axes have a large impact, they are explored exhaustively, whereas lower-priority axes are explored heuristically. To enable exhaustive exploration, HCo proposes pruning techniques that use domain knowledge to eliminate redundant and provably suboptimal regions of the space.

Identifying a combination of engines that achieve maximum workload coverage through exhaustive exploration is intractable. For example, selecting a multiset of 64 engines from 100 candidates requires exploring $\mathcal{O}(10^{46})$ possible combinations. To tackle this problem, we propose a strategy that balances two intuitive design principles: (1) designing for the common case by replicating the engine with the best average performance, and (2) maximizing specialization through engine heterogeneity. Rather than committing to either extreme, the strategy explores multiple engine combinations, drawn from the pool generated by MEDEM's first stage, that span the spectrum from one extreme to the other. Each combination is then incrementally refined, and the combination with the lowest aggregate execution cost is finally selected. We call this strategy **Balanced Average and Specialization Search (BASS)**.

Our contributions are as follows:

- We propose MEDEM, a systematic design methodology for flexible multi-engine DL accelerators. MEDEM is a two-stage methodology that decouples engine co-design and selection.
- We propose HCo, a co-design strategy that efficiently navigates an engine hardware–mapping space by exhaustively exploring high-impact axes and heuristically exploring the rest.

[2] A multiset is a collection of elements in which repetitions are allowed.

- We propose BASS, a strategy for selecting engine combinations that maximizes workload-set coverage by exploring combinations ranging from ones designed for the common case to highly specialized ones.
- We comprehensively evaluate MEDEM and its derived CMEAs against state-of-the-art homogeneous and heterogeneous multi-engine accelerator designs, demonstrating MEDEM's scalability and generalizability.

Comprehensive evaluation of MEDEM using 51 single- and multi-model workloads shows that it identifies CMEAs that outperform state-of-the-art homogeneous and heterogeneous multi-engine accelerator baselines modeled after Simba [144] and Maelstrom [86], achieving geometric mean improvements up to 4.84× in energy-delay product (EDP) and 1.59× in throughput. These improvements hold across various resource budgets ranging from those typical of edge to cloud systems.

5.2 Background and Motivation

5.2.1 Deep Learning Models and their Unique Layers

Deep Learning (DL) models, such as Convolutional Neural Networks (CNNs) [91] and Transformers [165], consist of a sequence of layers. Each layer performs a specific tensor operation to transform input data into alternative feature representations. Two core operations of these DL models, which are characterized by their multi-dimensional iteration spaces, are convolution and matrix multiplication [87, 122]. These operations fundamentally combine *input* activations with learned *weights* to produce output activations. A standard convolutional layer operates within a 6-dimensional space (K, C, H, W, R, S) shown in Figure 5.2, while a matrix multiplication operation has a 3-dimensional space (M, N, K), assuming no batching. We define a *unique layer* as a layer that has a specific set of values for all its dimensions. *Modern DL models, despite having tens to thousands of layers, are often composed of only a few unique layers that repeat throughout the model.* We refer to layers sharing identical dimensional values as *instances or repetitions* of a single unique layer.

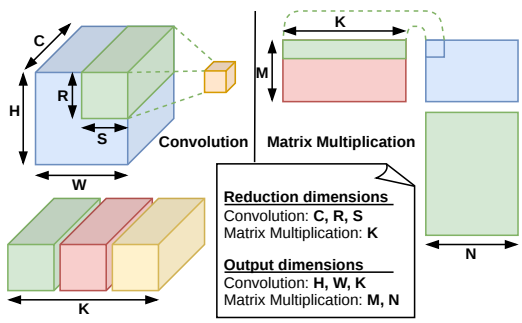


Figure 5.2: Convolution and matrix multiplication dimensions, reduction and output dimensions.

5.2.2 DL Computing Engines

We define an engine as an array of Processing Elements (PEs) with a multi-level memory hierarchy, including private register files (RFs), local and global buffers. Engines can be broadly categorized based on which dimensions of the DL layers can be mapped in parallel (i.e., spatially) across the PE array. Different dimensions of the core deep learning operations have two main forms of parallelism. First, parallelism on the output dimensions: (K, H, W) in convolution (Figure 5.2) and (M, N) in matrix multiplication. We call parallelism on these dimensions *non-reductive (NR)*. Second, parallelism on dimensions that require partial-sum accumulation: (C, R, S) in convolution and (K) in matrix multiplication, *reductive (R)*. Supporting spatial mapping of these dimension types has implications for an accelerator microarchitecture and for data reuse opportunities [66, 84, 122, 187]. For example, when parallelizing on NR dimensions, each PE computes an output element independently and accumulates the result at its local register [37]. Consequently, the *hardware primitives* in the NR parallelism case do not require spatial reduction logic. By contrast, parallelizing R dimensions requires spatial reduction primitives, typically implemented via inter-PE adder trees or global accumulation registers [66, 144].

Table 5.1: Engine types used in existing multi-engine accelerators. NVDLA/Simba (**NV/Si**), Eyeriss (**Eye**), Systolic Array (**SA**), and ShiDianNao (**Shi**). **Het.** indicates heterogeneous engines. If an accelerator is homogeneous, multiple ✓mean that different engines are used mutually exclusively, one type at a time, in different experiments.

Work	NV/Si	Eye	SA	Shi	Het.
Tangram [43]		✓			✗
AI-MT [3], Planaria [44], SPA [14], TPU v4 [70]			✓		✗
Simba [144], NN-baton [161], SET [12], Gemini [13]	✓				✗
Atomic [201]	✓			✓	✗
Crane [45]	✓		✓		✗
ConfuciuX [73]	✓	✓		✓	✗
MAGMA [75]	✓	✓			✓
SCAR [120]	✓			✓	✓
HDA [86], MOHaM [29]	✓	✓		✓	✓

5.2.3 Multi-Engine DL Accelerators

Multi-engine DL accelerators consist of independently controllable engines interconnected via a high-bandwidth communication fabric. These accelerators offer several advantages over monolithic ones. *First*, they offer more specialization opportunities by dedicating the engines to address the heterogeneity of DL model kernels, improving performance and efficiency [10, 14, 29, 86, 197]. *Second*, they exploit model-level parallelism by enabling concurrent execution of different model layers [14, 43, 86, 201]. *Third*, their modular design improves scalability

and flexibility for large models [45,144]. *Finally*, they expose more mapping and scheduling optimizations for multi-model workloads [13,45,120,169,172].

We define *flexible multi-engine accelerator* as an accelerator capable of executing a diverse set of DL workloads, to distinguish from *model-specific accelerators* usually optimized for specific DL models [49, 131]. Designing the engines of flexible multi-engine accelerators is largely expert-driven. As Table 5.1 shows, existing flexible multi-engine accelerators use combinations of independently optimized engine architectures.

5.2.4 Motivating a Two-Stage Design Methodology

To fully exploit the potential of multi-engine DL accelerators, systematic design methodologies are needed. A natural extension of single-engine hardware-mapping co-design is to treat engine configurations and engine combinations as dimensions of a *single joint design space* and explore them simultaneously. This approach is exemplified by MOSAIC [30], the first framework to comprehensively target the design of flexible multi-engine DL accelerators. However, this approach has multiple shortcomings.

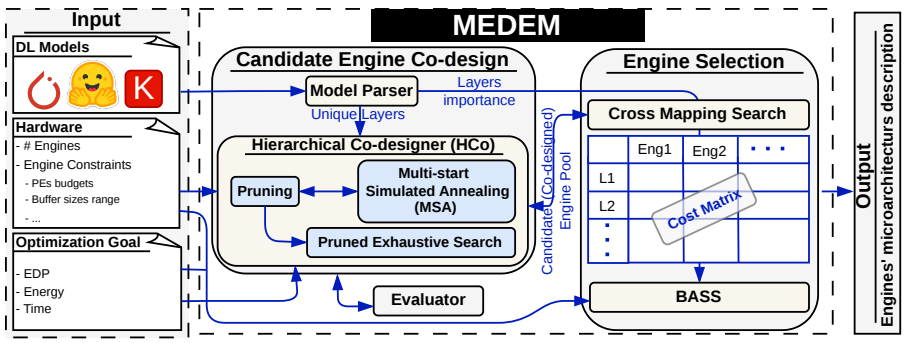
First, the joint design space is dominated by combinations containing inefficient engines. This is because *optimized configurations constitute only a small fraction* of all feasible designs in the per-engine design space [31,74,122,138]. Moreover, substantial quality variation exists even among top-performing engines; for example, TimeLoop reports up to an 11× difference in energy efficiency among 6582 designs that all achieve the minimum number of DRAM accesses [122]. Consequently, sampling the joint space is likely to spend most of its effort evaluating combinations composed of suboptimal engines. **Second**, jointly exploring engine configurations and engine combinations induces multiplicative growth in the search space, increasing exploration complexity and *limiting scalability*. The practical impact of this growth can be observed in MOSAIC, whose evaluation limits the design space to accelerators with 24 engines or fewer (1–3 engine types and 1–8 instances per type [30]). **Third**, another impact of the space growth is practically *limiting per-engine hardware-mapping co-design*. For instance, although MOSAIC explores 12 design knobs, where DAG-aware mapping across heterogeneous engines is considered, within-engine mapping remains largely unexplored: loop ordering is not considered, and data tiling is restricted to values that fit the working set [30]. However, co-designing architectural and mapping dimensions is crucial for discovering highly efficient accelerator designs [31,56,77,111,138,180].

These shortcomings motivate *decoupling engine co-design and combination selection*. **First**, decoupling enables the identification of high-quality engine candidates, avoiding the expenditure of search effort on combinations involving poor or mediocre engines. **Second**, it enables a more structured and scalable exploration compared to simply exploring joint space. **Third**, as the decoupled (isolated) engine design space is smaller, co-designing engines by co-exploring the hardware and mapping spaces becomes more practical. Moreover, as the engine co-design problem is extensively studied in the literature, isolating it (the co-design) enables direct applications of insights from the rich literature.

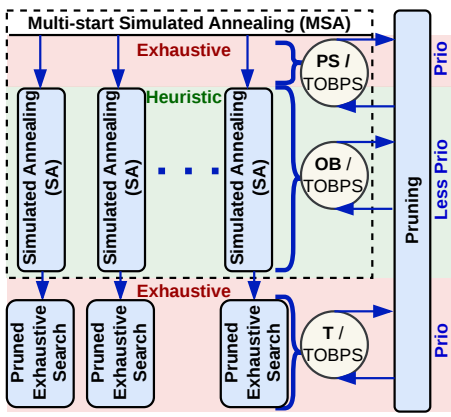
5.3 MEDEM Overview

5.3.1 MEDEM Flow and Core Modules

Figure 5.3a shows a high-level overview of the MEDEM flow and its core modules. MEDEM takes as **input** a set of DL models expressed using widely used DL frameworks, such as PyTorch, a hardware resource budget including the number of engines and the resource limit per engine, and an optimization goal. As an **output**, MEDEM produces hardware configurations of engines that form an optimized Co-designed Multi-Engine Accelerator (CMEA) and the number of instances of each engine. MEDEM has two main modules. The first is the **candidate engine co-design** module(Section 5.4), which produces a pool of candidate engines, each optimized for a unique kernel (layer). The second is the **engine selection** module (Section 5.5), which selects a multiset of co-designed candidates that maximizes workload coverage, or minimizes aggregate execution costs with respect to the optimization goal. We quantify coverage using the *geometric mean* of execution cost across the workload set.



(a) MEDEM overview.



(b) Hierarchical Co-designer (HCo) flow.

Figure 5.3: MEDEM overview and Hierarchical Co-designer flow.

Although at runtime the workloads of a CMEA are complete DL models, at design time, MEDEM considers individual *unique layers* (defined in

Section 5.2.1). MEDEM is *model-topology-agnostic*, prioritizing flexibility for a diverse workload set. Considering model topologies introduces a trade-off between generality and the ability to maximize optimization opportunities for specific models. Since our goal is to design a *flexible* accelerator for diverse models, MEDEM avoids optimizations tailored to specific model topologies. Instead, it aims to identify CMEAs that optimize aggregate execution cost across any set of models constructed from layers with given characteristics, independent of their topologies.

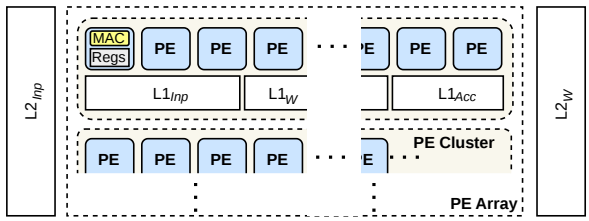


Figure 5.4: MEDEM engine abstraction, focusing on compute and memory components.

5.3.2 MEDEM Supported Engine Architectures

An engine is abstracted as a 2-D array of PEs exemplified in Figure 5.4. Each PE contains a Multiply–Accumulate (MAC) unit and a small set of local registers. The PEs are grouped into 1-D PE clusters. A PE cluster has three buffers local to the cluster and accessible to all its PEs (L1 buffers). These buffers hold input, weights, and partial sums (accumulation results). The engine has two global buffers accessible by all its PE clusters, one for inputs and one for weights (L2 buffers). We model the buffers as buffets [123]. Buffets decouple fills and accesses through fine-grained synchronization, making them more efficient than the commonly used double-buffered scratchpads. We broadly categorize our engines into two types based on their hardware primitives support for parallelism on R and NR dimensions (Section 5.2.2). We call these two types *R-engines* and *NR-engines*. In the rest of the paper, we use the term *unique engine* to refer to an engine that has a specific set of values for the type, PE array shape, and size of all buffers. Note that the R and NR are abstractions rather than two concrete engine configurations. This work explores instances of these abstractions with various PE array shapes, spatial mapping, dataflow options, and buffer sizes. Moreover, given an engine instance, all valid loop orders and tiling options at each level of the memory hierarchy are explored.

5.4 Candidate Engine Co-design

This section presents MEDEM's first core module, the candidate engine co-design module, which comprises two components. First, *model parser*, which extracts *unique layers* (defined in Section 5.2.1) from the input DL models. Second, *Hierarchical Co-designer (HCo)*, which is the key component of this module. HCo co-designs a pool of engines given the unique layers and resource

budgets. As the same co-designed engine can be optimal for multiple layers, the engine pool may have fewer engines than the unique layers.

Hierarchical Co-designer (HCo)

The size and structure of an engine hardware-mapping co-design space make uniform exploration unlikely to produce consistently high-quality solutions. Hence, HCo uses hierarchical exploration *prioritizing important axes* of the space based on their impact on the execution cost. The prioritized axes are explored exhaustively, and the rest are explored heuristically. Exhaustive exploration is enabled through domain-informed design space pruning.

5.4.0.1 Importance-based Design Axes Prioritization

HCo's axes prioritization relies on a four-axis representation of the hardware-mapping space, named *TOPS* [76]. TOPS stands for tiling (T), loop orders (O), parallelization (P), and PE array shapes (S). However, we separate the tiling and buffer sizes (B), naming this variant TOBPS. An extensive analysis of these axes suggests that optimizing *parallelism and PE array shape (PS)* yields considerable performance gains [76]. The same applies for *blocking or tiling (T)*, [76, 138, 187]. Hence, HCo prioritizes these axes, namely (PS) and (T), and explores them exhaustively, and explores (OB) heuristically.

5.4.0.2 HCo Components and Flow

Figure 5.3b shows HCo's flow and sub-optimizers. The first sub-optimizer is a Multi-Start Simulated Annealing (MSA). The MSA explores *parallelism and PE array shape together (PS)* exhaustively. We apply simple pruning on array shape (S), limiting its dimensions to powers of two; with this, the P and S options are usually tens to hundreds, making an exhaustive exploration practical. Each SA search of the MSA explores the combinations of *loop orders and buffer sizes (OB)* (heuristically). And for each explored combination, HCo starts an exhaustive search for *optimal tile sizes and shapes (T)*. Unlike the (PS), the remaining axes have huge spaces and require considerable pruning. HCo's partitioning of the space and grouping of hardware and software axes together may sound counterintuitive. However, together with the applied pruning, it facilitates selective exploration of promising parts of the design space, leading to the identification of highly optimized design points (as demonstrated in Section 5.6.3.1).

5.4.0.3 Pruning Redundant, Dominated, and Undesired Design Points

Previous work primarily prunes *invalid* design points, but the design space also contains valid yet suboptimal points. We classify these as *redundant* (equivalent), *dominated* (provably suboptimal), or *undesired* (excluded due to designer preferences or implementation constraints). Because design-space axes are interdependent, pruning must be applied across all axes to enable exhaustive exploration of important ones. This section skips pruning invalid designs, which is well covered in the literature, and discusses the other three.

Examples of redundant design points. We define *interchangeable dimensions* as dimensions (Section 5.2.1) with identical values and input-to-output mappings. In the absence of parallelism on such dimensions, tiles (**T**) that differ only in the permutation of these dimensions' values provide equivalent PE utilization and reuse and are therefore redundant. Similarly, loop orders (**O**) that differ only in the permutations of not parallelized *interchangeable dimensions* are redundant. Pruning redundant points shrinks the space considerably. For example, as our engines permit any loop order at any memory level (Section 5.3.2), there are $(7!)^3 \approx 12.8$ billion possibilities when considering convolution with batching. Interchangeable dimensions-based pruning can reduce these to just thousands.

Examples of dominated design points. Orojenesis [62] shows that, for matrix multiplication, the maximum effectual buffer size (**B**) is determined by the smallest operand (smallest operand size + its smallest rank + 1). We generalize this to convolution with an extra constraint: the smallest rank must be shared with the other input. Global buffers larger than the maximal effectual size are suboptimal (dominated) as they cost more area and energy per access with no added value. To give another example, we define two tiles (**T**) A and B as *divisible*, with respect to a buffer size, if both fit in the buffer, A is larger than B, and each dimension of A is an integer multiple of the corresponding dimension of B. Any tile that divides A is pruned as *dominated*, since it is guaranteed to be suboptimal. This criterion is stricter than the common practice of retaining only the largest fitting tile. Because different dimensions may provide different reuse and PE utilization, a tile B that is smaller than A, but larger in at least one dimension, is not necessarily dominated by A.

Example of pruning undesired points. We define *collocated dimensions* as those that appear consecutively in a loop order (**O**). The idea of collocated dimensions is a *generalization of the concept of stationary* [18] applied to different levels of the memory hierarchy. For example, any loop order where the dimensions of a layer output at a certain memory level are not collocated as outermost is not "output stationary" relative to that level. One can enforce stationary loop orders by pruning loop orders that have certain non-collocated dimensions.

Gradual pruning relaxation. Applying all pruning types may leave the design space with no valid solutions under the specified hardware budget. In this case, pruning is gradually relaxed by reintroducing previously pruned dominated and undesired designs, while redundant designs remain excluded because equivalent counterparts already exist in the pruned space. After each reintroduction, the exploration is repeated. Designs are added back in order of increasing design space expansion, meaning that pruning types with the smallest impact on the design space are relaxed first.

5.5 Engine Selection

This section describes MEDEM's engine selection module, which identifies a multiset of co-designed engines that minimize an aggregate (measured as geometric mean) execution cost of workloads constructed from layers with given characteristics. Permitting multiple selections of the same engine enables

Algorithm 4 Balanced Average and Specialization Search (BASS)

Input: Engine Candidate Pool $\mathcal{P}$, Costs Matrix $C_{L\times E}$, Engine multiset Size K, Seeds Heterogeneity $H = \{1, 2, 4, \ldots, K\}$
$engCombs \leftarrow \emptyset$ ▷ Storage for local optima combinations
for $h \in H$ **do** ▷ Multiple searches: exploring the specialization spectrum
  $C_h \leftarrow \text{UniformReplicaInit}(\mathcal{P}, C_{L\times E}, K, h)$ ▷ Seed a combination by replicating top-h engines uniformly
  repeat ▷ Greedy Hill-Climbing refinement
    $swapped, \delta \leftarrow \text{bestOfAllSwaps}(C_h, \mathcal{P}, C_{L\times E})$ ▷ Try swapping every engine in the combination
    if $\delta > \epsilon$ **then** $C_h \leftarrow swapped$ ▷ Commit swap that improves by more than threshold ϵ
  until $\delta \leq \epsilon$ ▷ Terminate at local optimum
  $engCombs \leftarrow engCombs \cup \{C_h\}$ ▷ Add result to local optima set
end for
return $C \in engCombs$ with minimum weighted geometric mean ▷ Return best overall combination (multiset)

the reuse of high-utility engines, which increases the probability that relatively more *important layers* (Section 5.5.1) are executed on engines that best suit their computational characteristics. The size of the search space of selecting combinations of K engines out of E candidates is shown in Equation 5.1. The engine selection module (Figure 5.3a) contains two components. The *cross mapping search* constructs a *cost matrix* given the unique layers extracted from the input models and the co-designed engine pool, as described in Section 5.5.1. The second is *Balanced Average and Specialization Search (BASS)*, which selects the best combination (multiset) of candidate engines to construct a CMEA as described in Section 5.5.2.

$$\binom{E+K-1}{K} = \frac{(E+K-1)!}{K!(E-1)!} \tag{5.1}$$

5.5.1 cost matrix construction

Selecting the best K engines from E candidates requires first evaluating the cost of executing every unique layer on every engine. For engines other than the one co-designed for a given layer (Section 5.4), this cost is not uniquely defined, as it depends on the mapping. This is the task of the *cross mapping search* (Figure 5.3a), which basically reuses the HCo to search for the mapping parameters, while fixing the hardware parameters. The execution costs of the best mappings are weighted by the importance of the unique layers and recorded in the *cost matrix*. We define *layer importance* as the number of times instances of a layer appear across all input models. This definition does not account for the computational complexity of the layers, as the impact of complexity is already reflected in the obtained execution costs.[3]

[3]MEDEM also allows specifying custom importance values.

5.5.2 Balanced Average and Specialization Search (BASS)

When designing multi-engine accelerators, two principles can be pursued: optimizing for the common case by replicating an engine with the best average execution cost and maximizing specialization through engine heterogeneity. BASS views these principles as opposing endpoints of a design spectrum and explores engine combinations spanning different points along that spectrum. The combinations are drawn from the candidate pool generated in MEDEM's first stage. The combinations are refined independently, and ultimately the combination with the lowest aggregate execution cost is selected.

BASS is outlined in Algorithm 4. The algorithm describes searching for a combination (multiset) of K engines from a pool of E candidates. Each local search starts *at line 4* with an initial engine combination constructed by uniformly replicating the top h engines ranked by average execution costs across all layers (obtained from the cost matrix in Section 5.5.1). For example, when h is 4, and K is 64, each of the top 4 engines is replicated 16 times. Different initializations represent points on a spectrum from designing for the common case by replicating one engine ($h = 1$) that is best on average, to maximizing specialization or engine heterogeneity ($h = K$). Each initial combination is incrementally refined using a greedy hill-climbing search (lines 5–8). At each iteration, the algorithm evaluates all possible engine swaps—*i.e.*, replacing each of the K engines in the current combination with every candidate engine in E—and selects the swap that provides the greatest marginal gain (line 6). Where the gain is defined as the reduction in aggregate execution cost. The gain is a proxy indicating that the swap results in a combination that comprises engines with better complementary capabilities. If the gain exceeds a predefined threshold[4], the swap is accepted, and the combination is updated (line 7). Marginal gain is computed relative to the best combination found throughout the search. The process repeats until no swap yields an improvement above the threshold, at which point a local optimum is reached (line 8). This local optimum is then added to the set of candidate solutions (line 9). After exploring multiple points on the heterogeneity spectrum, the algorithm compares the best local optimum obtained from each search and returns the one with the best workload coverage, measured by the minimum weighted aggregate cost (line 11).

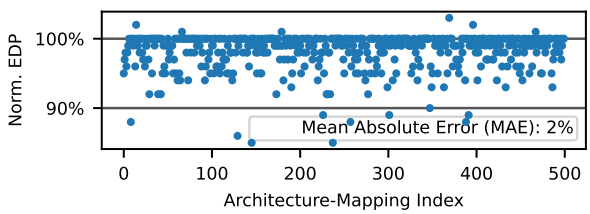


Figure 5.5: EDP estimated by our model normalized to Timeloop-Accelergy-Infrastructure using a random sample of 500 architecture-mapping.

[4]In all experiments, the threshold was set to $\epsilon = 10^{-9}$ solely to avoid floating-point round-off errors and did not affect the optimization.

5.5.3 Evaluator

Although pruning substantially reduces an engine hardware-mapping space, HCo may still explore billions of candidate configurations per layer, especially under relaxation. Hence, we develop a fast evaluation method. The core idea of our evaluator is similar to that of DOSA [56]. Namely, the evaluator is based on decoding Timeloop [122] and converting part of the iterative program into mathematical models. As the evaluator is not a major contribution of this work, we highlight the main differences between our evaluator model and DOSA's and refer the reader to DOSA for detailed information[5]. *First*, while DOSA supports only Systolic Arrays with square shapes and weight-stationary dataflow, our evaluator supports any engine that satisfies the abstraction described in Section 5.3.2. *Second*, unlike DOSA, our modeling is not differentiable, as supporting a wider range of architectures makes it harder to formulate a differentiable model. The evaluator mainly estimates engine execution cycles and memory accesses at all levels of the memory hierarchy. To estimate energy, we collect energy per operation and energy per memory access from Timeloop-Accelergy infrastructure [122, 177]. The values are collected offline for all supported configurations of each hardware component. During co-design, the evaluator performs a lookup in the collected values. Figure 5.5 shows the EDP estimated by the evaluator compared to Timeloop's. As depicted, the average error margin is only 2%, and only a few cases have accuracy less than 90%.

5.6 Evaluation

5.6.1 Experimental Setup

5.6.1.1 Baselines

To demonstrate the efficiency of MEDEM's engine co-design stage, we compare HCo to Spotlight [138], a state-of-the-art sample-efficient co-design tool. Spotlight originally uses MAESTRO [85]. However, the authors also present TimeLoop-based evaluation and show considerable differences between the two in some cases. Hence, for fairness, we re-evaluate the top-10 design points suggested by Spotlight using our Timeloop-Accelergy-based evaluator (Section 5.5.3) and use the best among the results as our baseline.

We evaluate MEDEM-derived CMEAs against Simba-like [144] (Simba-L), and Maelstrom-like [86] (Maelstrom-L) accelerators. Simba-L represents a homogeneous multi-engine accelerator, and Maelstrom-L represents a heterogeneous one. NVDLA-based multi-engine accelerators like Simba are the most common in related work [12, 13, 45, 120, 144, 161, 201]. Maelstrom heterogeneous engines use NVDLA and ShiDiaNao. To evaluate scalability, we evaluate engine counts from 4 (2×2) to 64 (8×8), representing a broad spectrum of resource budgets and deployment scenarios, from edge to cloud systems. As MEDEM is a design methodology, the goal is to compare its outcomes with accelerators that capture the key characteristics of Simba and Maelstrom, rather than specific implementations. However, for fairness, we consider two configurations for

[5] Moreover, MEDEM code, including the evaluator, will be open-sourced.

each baseline and compare with the one that gives the baseline the best results throughout the evaluation. In the first configuration, engines use hardware resources similar to those reported in the original papers. In the second, engine resources are scaled uniformly, up or down, so that the overall area is similar to the corresponding CMEA's.

5.6.1.2 System Modeling

The energy and execution time of the engines are modeled and validated against Timeloop [122] and Accelergy [177] (which use Aladdin [146] and CACTI [116] plug-ins under the hood) as described in Section 5.5.3. We use TimeLoop-Accelergy's 7nm technology node values in all our experiments, except for the comparison with Spotlight (Section 5.6.3.1), where we use TimeLoop-Accelergy's 28nm values to match Spotlight's technology node. Regarding the inter-engine communication fabric, we model a mesh NoC using a customized implementation of the open-source NoC modeling in SET [12]. The customization adds support for heterogeneous engines, which SET does not support natively. SET NoC modeling is used by previous work on multi-engine accelerators [13, 45]. The NoC bandwidth is set to 32 GB/s per link, and the energy per hop to 0.7 pJ/bit.

Table 5.2: Used workloads. *DT* stands for Design-Time.

CNN Models	
DT	**1**: VGG16 [157], **2**: VGG19, **3**: InceptionV3 [157], **4**: Inception-ResNet-v2 [155], **5**: ResNet50 [50], **6**: ResNet101, **7**: MobileNet [57], **8**: MobileNetV2 [142], **9**: DenseNet121 [58], **10**: DenseNet201
Unseen	**11**: ConvNeXt [102], **12**: ResNet-RS50 [7]
Transformer Models	
DT	**13**: BigBird [193], **14**: DeBERTa [52], **15**: BERT [34], **16**: RoBERTa [101], **17**: DistilBERT [143], **18**: ELECTRA [24], **19**: ALBERT [89], **20**: XLM-R [27], **21**: T5, **22**: GPT-Neo [8], **23**: CamemBERT [109], **24**: XLNet [189]
Unseen	**25**: ViT [36], **26**: BART [95], **27**: Llama-3 (8B) [46]
Multi-Model	
DT	**2-model**: (2,20), (4,15), (10,23), (1,12), (6,19), (3,24), (5,21), (8,18), (7,17), (9,14) **3-model**: (2,9,24), (6,8,15), (1,5,12), (10,3,19), (7,4,14)
Unseen	**2-model**: (25,12), (25,11), (26,11), (26,12), (27,12), (27,11) **3-model**: (25,12,11), (26,11,12), (27,11,12)

5.6.1.3 Workloads

We evaluate widely used CNN and Transformer models listed in Table 5.2. The design-time (DT) models in the table are those whose unique layers are used at the design phase. The Unseen models are used to evaluate MEDEM generalization. The unseen models comprise both evolved versions of the DT ones and models exhibiting structural or paradigm shifts. We form multi-model scenarios (2-model and 3-model) by combining a Transformer with one or two CNNs. As Transformers are built from a few repeating unique layers, the goal of combining a Transformer with two CNNs is to create multi-model workloads where heterogeneity, in terms of unique layer counts in a workload, is maximized. As there are numerous combinations of the DT models, the multi-model DT

workloads are formed randomly. For unseen models, we consider all possible combinations. The evaluation uses the commonly used batch size of 64. The weights and activations use int8 quantization, and the accumulators use int32. In this work, we assume that off-chip memory capacity is not a bottleneck, meaning that it can hold the weights of large models like Llama-3 (8B).

5.6.1.4 Workload Mapping Across Engines

We use two common heuristics to map a workload on the engines of a multi-engine accelerator, namely layer-sequential (LS) and layer-pipeline (LP) [12, 73, 201]. LS processes layers one at a time, potentially using all resources per layer. LP distributes the layers across the engines and pipelines their execution. Both heuristics are applied, and the results of the best among them are adopted both for CMEAs and for the baselines.

Table 5.3: The explored possibilities of key hardware and mapping parameters in the evaluation (per engine).

Parameter	**Explored possibilities**
PE Array	X: 8–128 Y: 8–128
Regs/PE	8–128 (B)
$L1_{(I/W)}$	2–32 (KiB)
$L1_{Acc}$	0.25–4 (KiB)
$L2_{(I/W)}$	64–1024 (KiB)
Parallelism and dataflow	All possible Convolution ∩ Matrix multiplication dimension mappings
Loop Order	$(3!)^3$, $(6!)^3$
Tiling	Divisors of layer shape ∪ Multipliers of PE array shape

5.6.1.5 Design Space and MEDEM Parameters

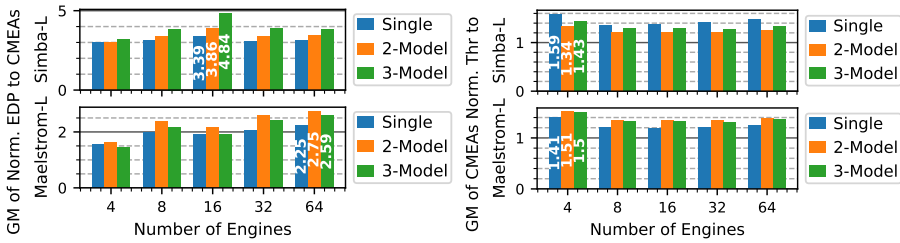
Table 5.3 summarizes the key per-engine parameters explored by the co-design stage of MEDEM. We avoid extremely skewed PE[6] arrays with a dimension smaller than 8 as they do not perform well on average. Regarding parallelism and dataflow, we explore all options common between convolution and matrix multiplication. This is because our accelerator must be flexible, where any layer must be executable on any engine. We explore all loop orders at the three levels of the memory hierarchy. This means up to 6! possibilities (Section 5.2.1) per memory level for convolution and 3! for matrix multiplication.

The Simulated Annealing (SA) optimizer, used in HCo, employs 50 main (cooling) steps with 100 iterations per step (temperature). The SA initial temperature T_0 is dynamically estimated based on a random walk over 100 samples from the space. An exponential cooling schedule is used with $\alpha = 0.95$. As SA is non-deterministic, we repeated each experiment 10 times using different random seeds. The observed variance on the aggregate results is negligible.

[6]The term PE is used differently in the literature. For example, the Simba paper uses it to refer to what corresponds to a PE cluster in our design.

5.6.1.6 Evaluation Metrics and Granularity

We mainly evaluate EDP, as is common in related work [12, 45, 53, 56, 60, 86, 120, 138]. We also present representative throughput results. We use the Geometric Mean (GM) to report aggregate performance across various workloads.



(a) Geometric mean of baselines EDP normalized to CMEAs'.

(b) Geometric mean of CMEAs' throughput normalized to baselines.

Figure 5.6: EDP and throughput comparison of MDEDEM-derived CMEAs to baselines using DT models using single- and multi-model workloads. Numbers in bars indicate the best case for a workload set.

5.6.2 End-to-End Evaluation of MEDEM

5.6.2.1 Evaluation of MEDEM-Derived CMEAs

This section evaluates MEDEM-derived CMEAs compared to homogeneous multi-engine accelerators (Simba-L instances) and heterogeneous ones (Maelstrom-L instances).

Figure 5.6 shows the EDP and throughput improvements achieved by CMEAs over Simba-L and Maelstrom-L. Compared to Simba-L, CMEAs achieve up to 4.84× lower EDP and 1.59× higher throughput. Compared to Maelstrom-L, CMEAs achieve up to 2.75× lower EDP and 1.51× higher throughput. Improvements in throughput are more modest than those in EDP. This is because optimizing throughput requires only maintaining full PE utilization, which is achievable for many layers by properly mapping a layer's dimensions on the PE array and by overlapping computation with data movement. In contrast, optimizing energy is more challenging as it requires maximizing spatial and temporal reuse across the memory hierarchy to reduce data movement costs. This makes EDP optimization more challenging; therefore, the remainder of the evaluation focuses on EDP.

Comparing results across different *engine counts* shows that CMEAs consistently outperform baselines at all engine counts. On the one hand, consistent improvements highlight the *MEDEM engine selection scalability*, as the design space of possible engine combinations grows exponentially with engine count. On the other hand, one would expect CMEAs' improvements over baselines to grow with engine count, as CMEAs can be more heterogeneous. In other words, while Simba-L and Maelstrom-L replicate one or two engine architectures, respectively, MEDEM can specialize to better capture workload heterogeneity. This expected trend does not always materialize due to the interplay of *two factors*: the limited heterogeneity of some models and the imbalance in layer contributions to the overall model computations. The impact of these factors

is described in detail in Section 5.6.4. Comparing improvements for *single- and multi-model* workloads in Figure 5.6 shows mixed trends. While improvements over Simba-L increase with the number of models per workload, the largest improvements over Maelstrom-L are observed using 2-model workloads. Again, the two mentioned factors help to explain the trends in these results as detailed in Section 5.6.4.

Overall, the fact that CMEAs improve over baselines across all experiments using representative workloads and at different scales demonstrates that *constructing multi-engine accelerators using co-designed engines rather than independently optimized ones achieves better workload coverage, i.e., minimizes aggregate execution cost.*

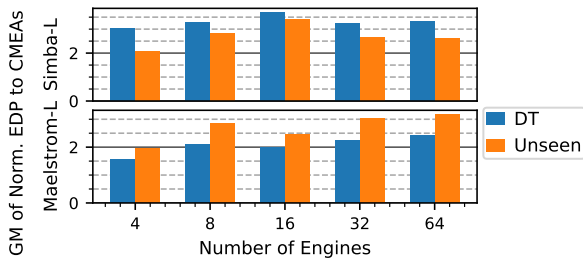


Figure 5.7: Comparison between improvements using design-time (DT) and unseen models. Each bar is the geometric mean (GM) of all the DT/unseen workloads, *i.e.* GM(single workloads, 2-model workloads, and 3-model workloads).

5.6.2.2 Evaluation of MEDEM Generalizability

Figure 5.7 compares EDP improvements over baselines for both DT and unseen workloads. MEDEM-derived CMEAs outperform baselines using unseen workloads across all engine counts, suggesting that CMEAs remain effective beyond the workloads whose layers were used during design-time. While CMEAs' improvements over Simba-L decrease for unseen compared to DT workloads, improvements over Maelstrom-L increase. When considering both baselines together, the opposing trends largely balance out. The overall trends indicate that *MEDEM generalizes relatively well.* As discussed in Section 5.3.1, MEDEM is model-agnostic; it relies solely on the overall distribution of unique layers across a representative set of workloads, rather than a workload topology. Hence, as long as the design-time workloads are representative of the DL workloads' landscape, the aggregate results of MEDEM-derived CMEAs should be generalizable.

5.6.3 Evaluation of MEDEM Engine Co-design Stage

5.6.3.1 Evaluation of HCo

This section demonstrates the effectiveness of MEDEM's HCo by comparing its co-designed engines with those co-designed using the Spotlight framework [138]. In this evaluation, we first ran MEDEM's candidate engine co-design module to co-design engines for the unique layers of all DT models, as described in

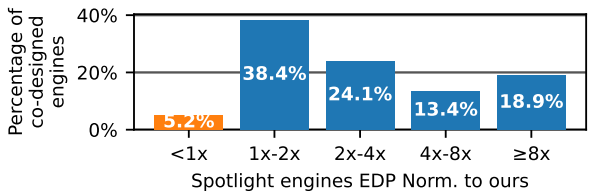


Figure 5.8: Histogram of EDP achieved by engines co-designed using Spotlight for all unique layers of the DT models, normalized to those co-designed using HCo. For example, 24.1% of engines co-designed using HCo have 2×–4× EDP improvements over to those co-designed using Spotlight.

Section 5.4. Then we replace HCo with Spotlight, use this modified setup to co-design the engines, and then compare the EDP values achieved by engines co-designed using each approach to the other. Figure 5.8 shows that HCo co-designed engines outperform Spotlight's in 94.8% of the experiments, while Spotlight outperforms HCo only in 5.2%. Moreover, HCo identifies engines with considerable improvements that exceed 8× compared to Spotlight in roughly one-fifth of the experiments. This is despite HCo being simpler than Spotlight, which relies on an expert-formulated sophisticated feature space [138]. These results demonstrate *the effectiveness of importance-based prioritization of high-impact axes of the design space.*

5.6.3.2 The Impact of Engine Co-design

This section focuses on isolating the impact of engine co-design on the efficiency of a multi-engine accelerator by comparing CMEAs with homogeneous engines to Simba-L and Maelstrom-L. These homogeneous CMEAs follow the principle of design for the common case by replicating the engine with the best average execution cost from the engine pool (Section 5.4). Figure 5.9 shows improvements over both Simba-L and Maelstrom-L of up to 2.64× and 1.92×, respectively. These results emphasize the *importance of co-designing the engines of a multi-engine accelerator*, even when they are homogeneous.

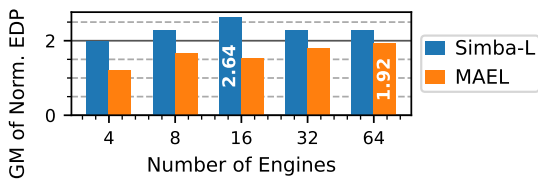


Figure 5.9: Comparsion of EDP achieved by CMEAs with homogeneous co-designed engines to the baselines (values are normalized to EDP of CMEAs with homogeneous co-designed engines). Each bar is the geometric mean (GM) of all 51 single- and multi-model DT and unseen workloads.

5.6.4 Evaluation of MEDEM Engine Selection Stage

This section isolates the impact of MEDEM's engine selection by comparing BASS with two alternative design principles: designing for the common case

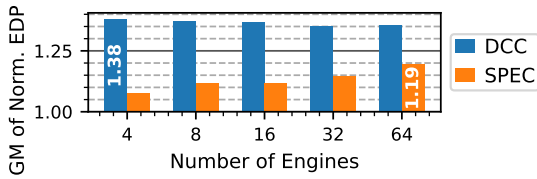


Figure 5.10: Comparison of BASS to two design alternatives: designing for common case (DCC), and maximizing specialization (SPEC). The values are normalized to BASS's. Each bar is the geometric mean (GM) of all 51 single- and multi-model DT and unseen workloads. DCC replicates one engine with the best average execution cost across all DT workloads K times (where K is the number of engines). SPEC comprises top-K unique engines ranked by their execution costs.

(DCC) and maximizing specialization (SPEC). Figure 5.10 presents this comparison. BASS identifies engine combinations that outperform both alternatives across all engine counts, demonstrating its effectiveness. Furthermore, while improvements over DCC remain relatively stable, improvements over SPEC increase with the engine count. From another angle, the performance gap between SPEC and DCC narrows as the engine count increases, suggesting that the benefits of engine heterogeneity exhibit diminishing returns. Explaining these trends requires considering two factors together: the *number of unique layers* in a workload, which is a proxy of that workload heterogeneity; and the *contribution* of these unique layers' instances to the overall workload MAC operations.

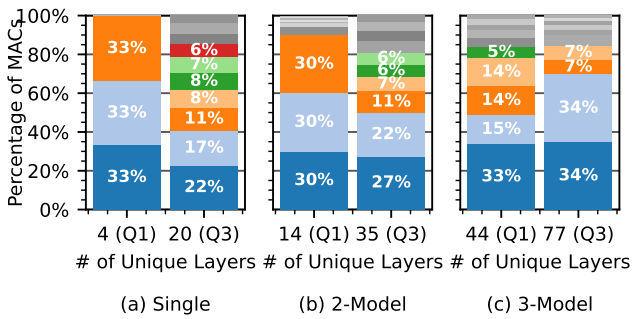


Figure 5.11: Unique layers and their contributions to the total MACs of a workload in single and multi-model workloads. A unique layer's contribution is calculated by aggregating MACs across all its instances (*i.e.* repetitions). A sample of two workloads is shown per workload set, representing the first (Q1) and third (Q3) quartiles of workloads sorted by their unique layer counts. Unlabeled gray slices in a bar represent layers whose overall instances contribute less than 5%.

Figure 5.11 depicts representative statistics of the two factors mentioned. Using single-model workloads as an example, Figure 5.11 (a) shows two representative models: one from the first quartile (Q1) and one from the third quartile (Q3) of the models sorted by the number of unique layers. The Q1 model represents relatively homogeneous models with few unique layers. It con-

sists mainly of instances (repetitions) of only four unique core layers. Three of the four layers contribute 99% to the model's total MACs. Consequently, using instances of more than four heterogeneous engines would exceed the workload's inherent heterogeneity and provide no additional benefit. In contrast, the Q3 model contains 20 unique layers. Nevertheless, only a small subset of unique layers (more precisely, all instances of these unique layers) accounts for the majority of the workload's MAC operations. Considering Q1 and Q3 models together suggests a sweet spot in engine heterogeneity, beyond which additional specialization yields diminishing returns or even degradation. Heterogeneous workloads such as Q3 benefit from specialized engines, but the gains diminish as new engine types target layers contributing fewer MACs. In contrast, Q1-like workloads may be harmed by further specialization. In other words, as the total number of engines is fixed, introducing additional engine types to improve support for the long tail of layers present in Q3-like workloads comes at the cost of reducing the replication of engines that are efficiently optimal for the instances of the few dominant layers in Q1-like workloads. Figures 5.11 (b) and (c) show that similar trade-offs exist in multi-model workloads; even though there are more heterogeneous layers in a workload, few of them still dominate the overall computations.

Given such trade-offs, maintaining consistent improvements over DCC, across engine counts, indicates that *MEDEM's engine selection avoids the trap of overspecialization*. Figure 5.12 shows the engine combinations selected by BASS across different engine counts. As observed, the number of unique engines is lower than the total engine count, suggesting diminishing returns from further specialization.

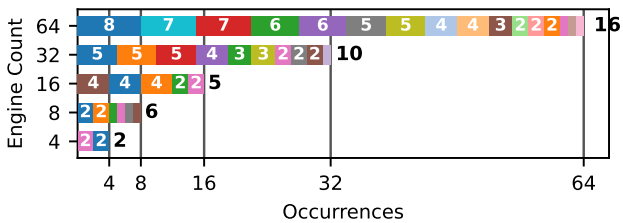


Figure 5.12: Engine combinations (multisets) selected by BASS and the contribution of unique engines. The number at the end of each bar indicates the number of unique engines in the selected combination. For example, for an engine count of 8, BASS selects a multiset of 6 unique engines, two of which are replicated twice, while the remaining four appear once. Colors encode engine configurations; slices with the same color across different bars correspond to the same engine configurations.

5.6.5 Overview of CMEAs Co-designed Engines

This section presents statistics and key architectural configurations of the engines selected by MEDEM. Figure 5.13 shows that 17 unique engines were selected across all experiments, with the most used engine selected 17% of the time. Both R and NR engines have similar overall usage (49.2% : 50.8%). Table 5.4 shows some hardware configurations of the most used 5 engines.

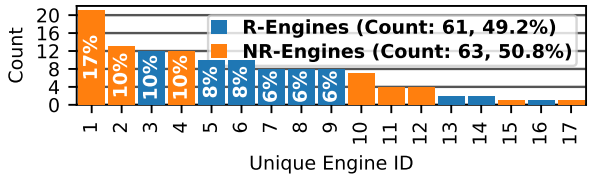


Figure 5.13: Occurrences of unique engines used across all CMEAs optimized for EDP. Percentages in bars are normalized to the total number of engines $(4+8+16+32+64)$.

Different PE array aspect ratios are used. Various $L1$ and $L2$ buffer sizes are used with different input-to-weight buffer size ratios. This reflects the varying input-to-weight ratios among layers, which is a feature that must be captured by the buffer distribution to minimize off-chip accesses [131].

Table 5.4: Key hardware configurations of the Top 5 most used engines across all the CEAs optimized for EDP.

Type	Percentage (%)	PE Array	$L1_{Inp}$	$L1_{W}$	$L1_{Acc}$	$L2_{Inp}$	$L2_{W}$
NR	17	32×32	2KiB	2KiB	1KiB	128KiB	128KiB
NR	10	128×8	8KiB	2KiB	1KiB	1MiB	512KiB
R	10	64×16	8KiB	2KiB	1KiB	128KiB	128KiB
NR	10	128×8	4KiB	16KiB	2KiB	64KiB	128KiB
R	8	64×16	2KiB	2KiB	256B	64KiB	128KiB

5.7 Related work

DL engines hardware-mapping co-design. Co-design of DL engines hardware–mapping is a widely explored open problem [31, 56, 61, 74, 76, 77, 138, 171, 180, 187]. Existing work uses a various exploration heuristics including Genetic Algorithms [74, 77], Bayesian Optimization (BO) [138, 180], Q-learning [180], variational autoencoders (VAEs) [60], and gradient descent [56]. The large size of the hardware and mapping spaces and the complex interactions between them necessitate sample-efficient exploration techniques [31, 56, 138].

Engines of flexible multi-engine DL accelerators. Existing multi-engine DL accelerators comprise homogeneous engines [13, 43, 70, 144, 195, 201], or heterogeneous ones [29, 86, 120]. For example, Simba [144] and Gemini [13] use NVDLA-like architecture in all their engines. SPA [14] engines are dataflow-hybrid SAs that support Weight Stationary (WS) and Output Stationary (OS) dataflow. Atomic [201] presents two multi-engine accelerator instances, one using NVDLA-like engines and one using ShiDiaNao-like [37]. Maelstrom [86] engines use NVDLA and ShiDiaNao. MOHam [29] engines use NVDLA, Eyeriss, and ShiDiaNao. Another viable option is to use Reconfigurable Dataflow Accelerators (RDA), like MAERI [88]. However, the literature shows that the overheads of RDA outweigh the gains [76, 86, 87].

Model-specific multi-engine DL accelerators. Model-specific, multi-engine DL accelerators usually use reconfigurable platforms. Reconfigurability facilitates dedicating the compute engines for a targeted model [9, 49, 168, 174,

197]. MCExplorer [131] is a comprehensive DSE framework for model-specific multi-engine accelerators. The model-specific accelerator design problem has a smaller and simpler design space as each engine is tailored to one or a few layers, with many mapping parameters fixed in hardware.

Comprehensive design methodologies of flexible multi-engine DL accelerators. MOSAIC [30] is unique in providing a comprehensive exploration framework for flexible multi-engine DL accelerators, supporting a broad range of both MAC and non-MAC layers. However, MOSAIC jointly explores the engine-configuration and engine-composition spaces in an unstructured manner, limiting both effectiveness and scalability.

Unlike existing flexible multi-engine DL accelerator design approaches, MEDEM proposes an efficient and scalable systematic methodology that decouples engine co-design from combination selection.

5.8 Conclusion

As monolithic DL accelerator optimizations reach diminishing returns, researchers are shifting toward multi-engine accelerators. This shift opens new design opportunities, but also introduces new challenges. A key challenge when designing a flexible multi-engine accelerator is identifying a combination of engines that maximizes coverage of diverse DL workloads to reduce aggregate execution costs. To tackle this challenge, this work proposes a systematic two-stage Multi-Engine DL accelerator DESign Methodology (MEDEM). Using generic engine abstractions, MEDEM co-designs candidate instances and selects a combination of the co-designed engines to improve coverage of diverse DL workloads. MEDEM encapsulates optimization strategies that efficiently navigate exponentially large design spaces, identifying flexible multi-engine accelerators that achieve geometric mean gains of up to 4.84× in EDP and 1.59× in throughput compared to the state-of-the-art. These gains are achieved using a representative set of DL workloads and under varying resource budgets.

Chapter 6

Conclusions

6.1 Summary

Multi-engine DL accelerators have emerged as a promising architectural paradigm with significant potential to improve the performance and efficiency of processing DL workloads. Compared to monolithic accelerators, they offer greater scalability and flexibility. Moreover, they provide more opportunities for specialization and parallelism by allowing independent engines to be tailored to the diverse computational characteristics of DL model layers and exploit latent parallelism across these layers. These advantages have driven the increasing adoption of multi-engine accelerators in both academia and industry. However, while multi-engine accelerators open new optimization opportunities, they also introduce new design challenges.

The *central premise* of this thesis is that the potential of multi-engine DL accelerators cannot be fully realized without systematic design methodologies that quantitatively evaluate key design choices. Fixing key design parameters a priori based on expert knowledge and intuition restricts the exploration to a limited subset of the available design space. Existing design methodologies, even those that are largely systematic and exploration-based, leave key design decisions unexplored. This problem is observed in the vast majority of both model-specific and flexible multi-engine DL accelerator design approaches.

This thesis addressed this problem and *advances the state-of-the-art* by expanding the set of design decisions that are subjected to systematic quantitative evaluation rather than expert assumption. **FiBHA** (Chapter 2) addresses the problem of presuming the high-level architecture of a multi-engine accelerator before evaluating its suitability for the target workload and resource budget. Rather than committing to a fixed architectural paradigm and optimizing within its constraints, FiBHA determines how different paradigms should be combined by quantifying their performance-resource trade-offs. In doing so, FiBHA demonstrates that the high-level architecture itself should be treated as a design decision to be evaluated rather than assumed beforehand. **MCCM** (Chapter 3) and **MCExplorer** (Chapter 4) extend this principle to a broader class of model-specific multi-engine DL accelerators. The state-of-the-art typically leaves several key architectural decisions unexplored and therefore fails to exploit the potential of the multi-engine design paradigm fully. MCCM

enables efficient quantitative evaluation of large numbers of architectural alternatives, while MCExplorer systematically explores design decisions that previous methodologies presume. Together, they advance the state-of-the-art by demonstrating that broader quantitative exploration uncovers high-quality architectures that remain inaccessible when such decisions are fixed *a priori*. **MEDEM** (Chapter 5) generalizes the same principle to flexible multi-engine DL accelerators. The state-of-the-art flexible multi-engine DL accelerator design methodologies typically assume that the best accelerator can be obtained by combining independently optimized expert-selected engines. MEDEM advances the state-of-the-art by demonstrating that the engines themselves should be co-designed and evaluated as part of the exploration process. By systematically exploring both engine design and engine combination selection, MEDEM identifies accelerator architectures with substantially better workload coverage and lower aggregate execution costs.

The advancement of the state-of-the-art achieved in this thesis can be summarized as follows: *key architectural decisions in multi-engine DL accelerator design should not be treated as assumptions, but as variables to be systematically explored and quantitatively evaluated.* The fundamental contribution of this thesis is therefore not a particular accelerator architecture, analytical model, or design-space-exploration framework in isolation. Rather, it is the demonstration that progressively moving design decisions from the category of *presumed choices* to the category of *explored alternatives* enables the discovery of accelerator architectures with superior performance and efficiency.

6.2 Research Questions and Contributions

6.2.1 Addressing Research Question 1

Research Question 1 *How can a model-specific DL accelerator be designed to optimize performance-resource trade-offs across different resource budgets?*

Research Question 1 is addressed by the first contribution of the thesis, namely the Fixed Budget Hybrid CNN Accelerator (*FiBHA*) and its associated design heuristic, *SplitCNN*. The work addresses the question in the context of compact and heterogeneous CNNs. The remainder of this discussion is organized into three parts. First, to help the reader place the contribution in context, it revisits the context of proposing FiBHA. Second, it explains how FiBHA and SplitCNN address the research question. Third, it highlights how FiBHA advances the state-of-the-art in connection with the main premise of this thesis.

6.2.1.1 Context

Existing accelerators targeting compact CNNs adopt two architectural paradigms that occupy opposite ends of a design spectrum. At one end are single-engine and Single-Engine Multiple-Layer (SEML) architectures, where each engine processes all layers or all layers of a given type, respectively. At the other end are Single-Engine Single-Layer (SESL) architectures, where each core layer is

assigned a dedicated engine. While SEML architectures do not sufficiently capture workload heterogeneity, SESL architectures sacrifice resource efficiency.

6.2.1.2 Addressing Research Question 1

FiBHA combines SEML and SESL paradigms in a hybrid architecture that balances *performance and resource utilization* in two ways. First, the high-level architecture is derived through empirical analysis of workload characteristics to determine how the *SEML and SESL components should be organized to minimize resource usage while maintaining performance*, as described in Chapter 2. Second, SplitCNN systematically partitions model layers and hardware resources among the available engines to *optimize performance-resource trade-offs* under a given resource budget.

6.2.1.3 Advancing State-of-the-Art in Light of the Thesis Premise

The limitation of the state-of-the-art for compact CNNs is that they adopt either an SEML or an SESL paradigm *without quantitatively justifying* this decision. Although some of these works employ sophisticated DSE and low-level optimizations, they already *fix the high-level architecture a priori*, thereby drastically constraining the design space. In contrast, *FiBHA* determines how SEML and SESL principles should be combined *by quantifying their performance-resource trade-offs* with respect to workload characteristics and resource constraints.

6.2.2 Addressing Research Question 2

Research Question 2: *How to systematically identify model-specific multi-engine DL accelerator architectures that fully exploit the potential of the multi-engine design paradigm?*

Research Question 2 is addressed by the second contribution of the thesis, namely the combination of the *MCCM* cost model and the *MCExplorer* DSE framework. The remainder of this discussion is organized into three parts. First, to help the reader place the contribution in context, it revisits the limitations of existing model-specific multi-engine accelerator design methodologies. Second, it explains how MCCM and MCExplorer address the research question. Third, it highlights how MCCM and MCExplorer advance the state-of-the-art in connection with the main premise of this thesis.

6.2.2.1 Context

The literature on model-specific multi-engine DL accelerator design contains works with varying degrees of modeling and design space exploration, ranging from almost none, where decisions are justified only through post-design comparisons with prior work, to more systematic exploration-based methodologies. However, even excellent systematic exploration-based methodologies leave key design decisions unexplored, leaving substantial potential unexploited.

6.2.2.2 Addressing Research Question 2

Systematically identifying high-quality multi-engine accelerator architectures requires quantitatively evaluating a vast number of design alternatives. Hence, addressing Research Question 2 requires a *fast evaluation methodology and an effective DSE framework*. MCCM and MCExplorer realize such a methodology and framework. First, MCCM addresses this challenge by enabling rapid architectural evaluation without relying exclusively on expensive synthesis-based design flows. MCCM achieves this through a high-level analytical cost model that captures the primary performance and efficiency bottlenecks of multi-engine accelerators, including processing-element underutilization, on-chip memory requirements, and off-chip memory accesses. Second, MCExplorer builds upon MCCM to systematically explore alternatives for key architectural decisions, including both inter-engine arrangements and intra-engine configurations. Through analytical evaluation and optimized search heuristics, MCExplorer efficiently identifies high-quality architectures from a significantly larger design space than those considered in prior work.

6.2.2.3 Advancing State-of-the-Art in Light of the Thesis Premise

The limitation of the state-of-the-art model-specific multi-engine DL accelerator design methodologies is that they *presumed* key architectural decisions, leaving substantial portions of the *design space unexplored*. Hence, they *fall short of exploiting the potential* of the multi-engine design paradigm. *MCCM* and *MCExplorer* demonstrate that *quantitatively evaluating a broader set of architectural decisions (that were presumed by the state-of-the-art)* uncovers high-quality design points that are *inaccessible to methodologies that presume these decisions*.

6.2.3 Addressing Research Question 3

Research Question 3: *How to systematically identify flexible multi-engine DL accelerator architectures that maximize coverage across a diverse DL workload landscape, thereby minimizing aggregate execution costs?*

Research Question 3 is addressed by the third contribution of the thesis, namely the Multi-Engine Deep Learning Accelerator Design Methodology (*MEDEM*). The remainder of this discussion is organized into three parts. First, it revisits the limitations of existing flexible multi-engine accelerator design methodologies. Second, it explains how MEDEM addresses the research question. Third, it highlights how MEDEM advances the state-of-the-art in connection with the main premise of this thesis.

6.2.3.1 Context

To efficiently process diverse DL workloads, flexible multi-engine DL accelerators must incorporate combinations of engines with complementary capabilities to match the distinct computational characteristics of these workloads' heterogeneous kernels. However, existing approaches generally construct such accelerators by combining *independently optimized expert-selected* engines. As a

consequence, the interaction between engines and the overall workload coverage of the resulting accelerator are not systematically considered during the design process.

6.2.3.2 Addressing Research Question 3

MEDEM showed that maximizing workload coverage requires the engines of a multi-engine accelerator to be *co-designed rather than independently optimized*. MEDEM then proposes a two-stage design methodology that decouples engine co-design and selecting an engine combination that forms a multi-engine accelerator. The *decoupling enables a structured and scalable exploration* of exponentially large design spaces. MEDEM proposes design strategies that enable highly efficient engine co-design and combination selection, leading to the systematic identification of flexible multi-engine DL accelerator architectures that maximize workload coverage.

6.2.3.3 Advancing State-of-the-Art in Light of the Thesis Premise

The limitation of the state-of-the-art flexible multi-engine accelerator design methodologies is that they implicitly presume that the best combination of engines is a combination of the best engines. Specifically, they explore combinations of *independently-optimized expert-selected engines*. Consequently, they fall short of maximizing workload coverage. In contrast, *MEDEM* constructs flexible multi-engine DL accelerators systematically by quantitatively evaluating a broad range of design alternatives, leading to architectures that minimize aggregate execution costs.

6.3 Limitations and Future Research Directions

The methodologies proposed in this thesis expand the role of systematic exploration in multi-engine DL accelerator design. Nevertheless, these methodologies have their limitations. This section highlights the main limitation of the proposed methodologies and discusses opportunities for future research.

The main limitation of this thesis concerns the *scope of the accelerator design problem*. Throughout this thesis, the proposed methodologies focus primarily on architectural decisions related to compute-engine organization and their impact on performance and efficiency. To enable systematic exploration, several other aspects of accelerator design are abstracted, including communication-fabric design, engine placement and interconnection strategies, and implementation-dependent effects. While these abstractions are appropriate for the objectives of this thesis, they also indicate that systematic accelerator design remains partially explored. A promising future direction is the development of *more holistic design methodologies* that extend the principles proposed in this thesis to a broader range of architectural decisions. Such methodologies could jointly consider compute-engine design, communication-fabric organization, engine placement, interconnect topology, and implementation-level constraints, enabling more comprehensive and quantitatively driven accelerator optimization.

Another future research direction is the *development of AI-driven accelerator design methodologies*. By leveraging knowledge acquired from previous

explorations, such methodologies could reason about architectural trade-offs, recommend promising design directions, generate candidate architectures, and adapt exploration strategies dynamically. Coupled with the systematic design-space representations proposed in this thesis, AI-driven methods could significantly improve the efficiency and scalability of accelerator design. At the same time, important research challenges remain regarding how such systems should acquire knowledge and generalize across diverse accelerator architectures, workloads, and design objectives.

A broader future research direction concerns extending the systematic design principles proposed in this thesis to *a wider range of domain-specific accelerators (DSAs)*. While this thesis focuses on multi-engine DL accelerators, the design of DSAs in other domains continues to rely heavily on expert knowledge and intuition when making key architectural decisions. As a result, important design alternatives may remain unexplored, limiting the achievable performance and efficiency. The methodologies proposed in this thesis suggest that a more systematic design approach, in which architectural decisions are treated as variables to be quantitatively evaluated rather than assumptions fixed *a priori*, yields superior outcomes. An interesting future research direction is therefore the development of methodologies that bring the same level of systematic and quantitative design to other classes of DSAs.

Bibliography

[1] 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," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[2] 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.

[3] 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.

[4] M. Baharani, U. Sunil, K. Manohar, S. Furgurson, and H. Tabkhi, "Deepdive: An integrative algorithm/architecture co-design for deep separable convolutional neural networks," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, 2021, pp. 247–252.

[5] L. Bai, Y. Zhao, and X. Huang, "A cnn accelerator on fpga using depthwise separable convolution," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 10, pp. 1415–1419, 2018.

[6] C. Baskin, N. Liss, E. Zheltonozhskii, A. M. Bronstein, and A. Mendelson, "Streaming architecture for large-scale quantized neural networks on an fpga-based dataflow platform," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2018, pp. 162–169.

[7] I. Bello, W. Fedus, X. Du, E. D. Cubuk, A. Srinivas, T.-Y. Lin, J. Shlens, and B. Zoph, "Revisiting resnets: Improved training and scaling strategies," *Advances in Neural Information Processing Systems*, vol. 34, pp. 22 614–22 627, 2021.

[8] S. Black, S. Biderman, E. Hallahan, Q. Anthony, L. Gao, L. Golding, H. He, C. Leahy, K. McDonell, J. Phang *et al.*, "Gpt-neox-20b: An open-source autoregressive language model," in *Proceedings of BigScience Episode# 5–Workshop on Challenges & Perspectives in Creating Large Language Models*, 2022, pp. 95–136.

[9] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks,"

ACM Transactions on Reconfigurable Technology and Systems (TRETS), vol. 11, no. 3, pp. 1–23, 2018.

[10] A. Boroumand, S. Ghose, B. Akin, R. Narayanaswami, G. F. Oliveira, X. Ma, E. Shiu, and O. Mutlu, "Google neural network models for edge devices: Analyzing and mitigating machine learning inference bottlenecks," in *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2021, pp. 159–172.

[11] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.

[12] J. Cai, Y. Wei, Z. Wu, S. Peng, and K. Ma, "Inter-layer scheduling space definition and exploration for tiled accelerators," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–17.

[13] J. Cai, Z. Wu, S. Peng, Y. Wei, Z. Tan, G. Shi, M. Gao, and K. Ma, "Gemini: Mapping and architecture co-exploration for large-scale dnn chiplet accelerators," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 156–171.

[14] X. Cai, Y. Wang, X. Ma, Y. Han, and L. Zhang, "Deepburning-seg: Generating dnn accelerators of segment-grained pipeline architecture," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1396–1413.

[15] X. Cai, Y. Wang, and L. Zhang, "Optimus: towards optimal layer-fusion on deep learning processors," in *Proceedings of the 22nd ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, 2021, pp. 67–79.

[16] X. Cai, Y. Wang, and L. Zhang, "Optimus: An operator fusion framework for deep neural networks," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 1, pp. 1–26, 2022.

[17] T. Chen, T. Moreau, Z. Jiang, H. Shen, E. Q. Yan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Tvm: end-to-end optimization stack for deep learning," *arXiv preprint arXiv:1802.04799*, vol. 11, no. 2018, p. 20, 2018.

[18] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH computer architecture news*, vol. 44, no. 3, pp. 367–379, 2016.

[19] 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.

[20] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[21] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[22] Z. Choudhury, S. Shrivastava, L. Ramapantulu, and S. Purini, "An fpga overlay for cnn inference with fine-grained flexible parallelism," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 19, no. 3, pp. 1–26, 2022.

[23] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman *et al.*, "Serving dnns in real time at datacenter scale with project brainwave," *iEEE Micro*, vol. 38, no. 2, pp. 8–20, 2018.

[24] 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.

[25] J. Coburn, C. Tang, S. A. Asal, N. Agrawal, R. Chinta, H. Dixit, B. Dodds, S. Dwarakapuram, A. Firoozshahian, C. Gao *et al.*, "Meta's second generation ai chip: Model-chip co-design and productionization experiences," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, 2025, pp. 1689–1702.

[26] S. Colleman, M. Shi, and M. Verhelst, "Coac: Cross-layer optimization of accelerator configurability for efficient cnn processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 31, no. 7, pp. 945–958, 2023.

[27] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov, "Unsupervised cross-lingual representation learning at scale," in *Proceedings of the 58th annual meeting of the association for computational linguistics*, 2020, pp. 8440–8451.

[28] W. J. Dally, Y. Turakhia, and S. Han, "Domain-specific hardware accelerators," *Communications of the ACM*, vol. 63, no. 7, pp. 48–57, 2020.

[29] A. Das, E. Russo, and M. Palesi, "Multi-objective hardware-mapping co-optimisation for multi-dnn workloads on chiplet-based accelerators," *IEEE Transactions on Computers*, vol. 73, no. 8, pp. 1883–1898, 2024.

[30] A. Das, H. Kim, S. Lee, A. Raha, D. A. Mathaikutty, and V. Raghunathan, "Mosaic: A workload-driven simulation and design-space exploration framework for heterogeneous npus," *arXiv preprint arXiv:2606.05362*, 2026.

[31] S. Dave, T. Nowatzki, and A. Shrivastava, "Explainable-dse: An agile and explainable exploration of efficient hw/sw codesigns of deep learning accelerators using bottleneck analysis," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4*, 2023, pp. 87–107.

[32] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[34] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[35] Z. Dong, Y. Gao, Q. Huang, J. Wawrzynek, H. K. So, and K. Keutzer, "Hao: Hardware-aware neural architecture optimization for efficient inference," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2021, pp. 50–59.

[36] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[37] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *Proceedings of the 42nd annual international symposium on computer architecture*, 2015, pp. 92–104.

[38] N. K. Dumpala, S. B. Patil, D. Holcomb, and R. Tessier, "Energy efficient loop unrolling for low-cost fpgas," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2017, pp. 117–120.

[39] A. Firoozshahian, J. Coburn, R. Levenstein, R. Nattoji, A. Kamath, O. Wu, G. Grewal, H. Aepala, B. Jakka, B. Dreyer *et al.*, "Mtia: First generation silicon targeting meta's recommendation systems," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.

[40] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi *et al.*, "A configurable cloud-scale dnn processor for real-time ai," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 1–14.

[41] J. Gao, Y. Qian, Y. Hu, X. Fan, W.-S. Luk, W. Cao, and L. Wang, "Leta: A lightweight exchangeable-track accelerator for efficientnet based on fpga," in *2021 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2021, pp. 1–9.

[42] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 751–764.

[43] 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.

[44] S. Ghodrati, B. H. Ahn, J. K. Kim, S. Kinzer, B. R. Yatham, N. Alla, H. Sharma, M. Alian, E. Ebrahimi, N. S. Kim *et al.*, "Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 681–697.

[45] Y. Gong, L. Huang, H. Chang, R. Liang, C. Yang, Z. Tang, J. Hu, and B. Yuan, "Crane: Inter-layer scheduling framework for dnn inference and training co-support on tiled architecture," in *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture®*, 2025, pp. 1250–1263.

[46] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[47] J. Guo, K. Han, H. Wu, Y. Tang, X. Chen, Y. Wang, and C. Xu, "Cmt: Convolutional neural networks meet vision transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 175–12 185.

[48] C. Hao and D. Chen, "Deep neural network model and fpga accelerator co-design: Opportunities and challenges," in *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*. IEEE, 2018, pp. 1–4.

[49] C. Hao, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W.-m. Hwu, and D. Chen, "Fpga/dnn co-design: An efficient design methodology for iot intelligence on the edge," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[50] 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.

[51] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer, 2016, pp. 630–645.

[52] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," *arXiv preprint arXiv:2006.03654*, 2020.

[53] K. Hegde, P.-A. Tsai, S. Huang, V. Chandra, A. Parashar, and C. W. Fletcher, "Mind mappings: enabling efficient algorithm-accelerator mapping space search," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 943–958.

[54] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[55] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[56] C. Hong, Q. Huang, G. Dinh, M. Subedar, and Y. S. Shao, "Dosa: Differentiable model-based one-loop search for dnn accelerators," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 209–224.

[57] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[58] 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.

[59] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer, 2016, pp. 646–661.

[60] Q. Huang, C. Hong, J. Wawrzynek, M. Subedar, and Y. S. Shao, "Learning a continuous and reconstructible latent space for hardware accelerator design," in *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2022, pp. 277–287.

[61] Q. Huang, M. Kang, G. Dinh, T. Norell, A. Kalaiah, J. Demmel, J. Wawrzynek, and Y. S. Shao, "Cosa: Scheduling by constrained optimization for spatial accelerators," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 554–566.

[62] Q. Huang, P.-A. Tsai, J. S. Emer, and A. Parashar, "Mind the gap: Attainable data movement and operational intensity bounds for tensor algorithms," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 150–166.

[63] 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.

[64] A. Ivanov, N. Dryden, T. Ben-Nun, S. Li, and T. Hoefler, "Data movement is all you need: A case study on optimizing transformers," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 711–732, 2021.

[65] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[66] J.-W. Jang, S. Lee, D. Kim, H. Park, A. S. Ardestani, Y. Choi, C. Kim, Y. Kim, H. Yu, H. Abdel-Aziz *et al.*, "Sparsity-aware and re-configurable npu architecture for samsung flagship mobile soc," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 15–28.

[67] H.-J. Jeong, J. Yeo, C. Bahk, and J. Park, "Pin or fuse? exploiting scratchpad memory to reduce off-chip data transfer in dnn accelerators," in *Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization*, 2023, pp. 224–235.

[68] J. Jiang, Y. Zhou, Y. Gong, H. Yuan, and S. Liu, "Fpga-based acceleration for convolutional neural networks: A comprehensive review," *arXiv preprint arXiv:2505.13461*, 2025.

[69] W. Jiang, L. Yang, S. Dasgupta, J. Hu, and Y. Shi, "Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 4154–4165, 2020.

[70] N. Jouppi, G. Kurian, S. Li, P. Ma, R. Nagarajan, L. Nai, N. Patil, S. Subramanian, A. Swing, B. Towles *et al.*, "Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proceedings of the 50th annual international symposium on computer architecture*, 2023, pp. 1–14.

[71] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[72] V. J. Jung, A. Symons, L. Mei, M. Verhelst, and L. Benini, "Salsa: Simulated annealing based loop-ordering scheduler for dnn accelerators," in *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2023, pp. 1–5.

[73] 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.

[74] S.-C. Kao and T. Krishna, "Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[75] S.-C. Kao and T. Krishna, "Magma: An optimization framework for mapping multiple dnns on multiple accelerator cores," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 814–830.

[76] S.-C. Kao, H. Kwon, M. Pellauer, A. Parashar, and T. Krishna, "A formalism of dnn accelerator flexibility," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 6, no. 2, pp. 1–23, 2022.

[77] S.-C. Kao, M. Pellauer, A. Parashar, and T. Krishna, "Digamma: Domain-aware genetic algorithm for hw–mapping co-optimization for dnn accelerators," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 232–237.

[78] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[79] O. Kopuklu, N. Kose, A. Gunduz, and G. Rigoll, "Resource efficient 3d convolutional neural networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019, pp. 0–0.

[80] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[81] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[82] R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan, "Heterogeneous chip multiprocessors," *Computer*, vol. 38, no. 11, pp. 32–38, 2005.

[83] S. R. Kuppannagari, R. Chen, A. Sanny, S. G. Singapura, G. P. C. Tran, S. Zhou, Y. Hu, S. P. Crago, and V. K. Prasanna, "Energy performance of fpgas on perfect suite kernels," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2014, pp. 1–6.

[84] 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.

[85] 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.

[86] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous dataflow accelerators for multi-dnn workloads," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 71–83.

[87] 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.

[88] 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.

[89] 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.

[90] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4013–4021.

[91] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[92] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proceedings of 2010 IEEE international symposium on circuits and systems*. IEEE, 2010, pp. 253–256.

[93] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[94] C. E. Leiserson, N. C. Thompson, J. S. Emer, B. C. Kuszmaul, B. W. Lampson, D. Sanchez, and T. B. Schardl, "There's plenty of room at the top: What will drive computer performance after moore's law?" *Science*, vol. 368, no. 6495, p. eaam9744, 2020.

[95] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proceedings of the 58th annual meeting of the association for computational linguistics*, 2020, pp. 7871–7880.

[96] G. Li, J. Zhang, M. Zhang, R. Wu, X. Cao, and W. Liu, "Efficient depthwise separable convolution accelerator for classification and uav object detection," *Neurocomputing*, vol. 490, pp. 1–16, 2022.

[97] 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.

[98] J. Liao, L. Cai, Y. Xu, and M. He, "Design of accelerator for mobilenet convolutional neural network based on fpga," in *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, vol. 1. IEEE, 2019, pp. 1392–1396.

[99] Y. Lin, M. Yang, and S. Han, "Naas: Neural accelerator architecture search," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 1051–1056.

[100] B. Liu, D. Zou, L. Feng, S. Feng, P. Fu, and J. Li, "An fpga-based cnn accelerator integrating depthwise separable convolution," *Electronics*, vol. 8, no. 3, p. 281, 2019.

[101] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[102] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 11 976–11 986.

[103] S. Lu, J. Chu, and X. T. Liu, "Im2win: Memory efficient convolution on simd architectures," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2022, pp. 1–7.

[104] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 553–564.

[105] 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.

[106] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing the convolution operation to accelerate deep neural networks on fpga," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 7, pp. 1354–1367, 2018.

[107] Y. Ma, N. Suda, Y. Cao, J.-s. Seo, and S. Vrudhula, "Scalable and modularized rtl compilation of convolutional neural networks onto fpga," in *2016 26th international conference on field programmable logic and applications (FPL)*. IEEE, 2016, pp. 1–8.

[108] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, vol. 5. University of California press, 1967, pp. 281–298.

[109] L. Martin, B. Muller, P. O. Suarez, Y. Dupont, L. Romary, É. V. de La Clergerie, D. Seddah, and B. Sagot, "Camembert: a tasty french language model," in *Proceedings of the 58th annual meeting of the association for computational linguistics*, 2020, pp. 7203–7219.

[110] L. Mei, K. Goetschalckx, A. Symons, and M. Verhelst, "Defines: Enabling fast exploration of the depth-first scheduling space for dnn accelerators through analytical modeling," in *2023 IEEE International Symposium*

on High-Performance Computer Architecture (HPCA). IEEE, 2023, pp. 570–583.

[111] 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.

[112] I. Micron Technology, "Tn-40-07: Calculating memory power for ddr4 sdram." [Online]. Available: https://www.mouser.com/pdfDocs/tn4007_ddr4_power_calculation.pdf

[113] T. Moreau, T. Chen, and L. Ceze, "Leveraging the vta-tvm hardware-software stack for fpga acceleration of 8-bit resnet-18 inference," in *Proceedings of the 1st on Reproducible Quality-Efficient Systems Tournament on Co-designing Pareto-efficient Deep Learning*, 2018, p. 1.

[114] P. Morì, M.-R. Vemparala, N. Fasfous, S. Mitra, S. Sarkar, A. Frickenstein, L. Frickenstein, D. Helms, N. S. Nagaraja, W. Stechele *et al.*, "Accelerating and pruning cnns for semantic segmentation on fpga," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 145–150.

[115] 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.

[116] N. Muralimanohar, R. Balasubramonian, N. P. Jouppi *et al.*, "Cacti 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.

[117] D. T. Nguyen, H. Kim, and H.-J. Lee, "Layer-specific optimization for mixed data flow with mixed precision in fpga design for cnn-based object detectors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 6, pp. 2450–2464, 2020.

[118] NVIDIA, "Nvdla: Nvidia deep learning accelerator," https://nvdla.org.

[119] Nvidia, "Nvidia rtx a4000," 2022, datasheet. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/gtcs21/rtx-a4000/nvidia-rtx-a4000-datasheet.pdf

[120] M. Odema, L. Chen, H. Kwon, and M. A. Al Faruque, "Scar: Scheduling multi-model ai workloads on heterogeneous multi-chiplet module accelerators," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 565–579.

[121] A. Pappalardo, "Xilinx/brevitas," 2021. [Online]. Available: https://doi.org/10.5281/zenodo.3333552

[122] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "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.

[123] M. Pellauer, Y. S. Shao, J. Clemons, N. Crago, K. Hegde, R. Venkatesan, S. W. Keckler, C. W. Fletcher, and J. Emer, "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.

[124] I. Pérez and M. Figueroa, "A heterogeneous hardware accelerator for image classification in embedded systems," *Sensors*, vol. 21, no. 8, p. 2637, 2021.

[125] L. Petrica, T. Alonso, M. Kroes, N. Fraser, S. Cotofana, and M. Blott, "Memory-efficient dataflow inference for deep cnns on fpga," in *2020 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2020, pp. 48–55.

[126] F. Qararyah, M. W. Azhar, M. A. Maleki, and P. Trancoso, "Fusing depthwise and pointwise convolutions for efficient inference on gpus," in *Workshop Proceedings of the 53rd International Conference on Parallel Processing*, 2024, pp. 58–67.

[127] F. Qararyah, M. W. Azhar, and P. Trancoso, "Fibha: Fixed budget hybrid cnn accelerator," in *2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 2022, pp. 180–190.

[128] F. Qararyah, M. W. Azhar, and P. Trancoso, "An efficient hybrid deep learning accelerator for compact and heterogeneous cnns," *ACM Transactions on Architecture and Code Optimization*, vol. 21, no. 2, pp. 1–26, 2024.

[129] F. Qararyah, M. A. Maleki, and P. Trancoso, "An analytical cost model for fast evaluation of multiple compute-engine cnn accelerators," *arXiv preprint arXiv:2503.07242*, 2025.

[130] F. Qararyah, M. A. Maleki, and P. Trancoso, "An analytical cost model for fast evaluation of multiple compute-engine cnn accelerators," in *2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2025, pp. 1–13.

[131] F. Qararyah, M. A. Maleki, and P. Trancoso, "Mcexplorer: Exploring the design space of multiple compute-engine deep learning accelerators," *ACM Transactions on Architecture and Code Optimization*, vol. 22, no. 4, pp. 1–27, 2025.

[132] F. Qararyah, M. Wahib, D. Dikbayır, M. E. Belviranli, and D. Unat, "A computational-graph partitioning method for training memory-constrained dnns," *Parallel Computing*, vol. 104, p. 102792, 2021.

[133] W. Qin, Z. Wu, J. Zhang, and F. Li, "Design and optimization of mobilenet neural network acceleration system based on fpga," in *2022 3rd International Conference on Electronics, Communications and Information Technology (CECIT)*. IEEE, 2022, pp. 7–12.

[134] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 26–35.

[135] A. Rahman, J. Lee, and K. Choi, "Efficient fpga acceleration of convolutional neural networks using logical-3d compute array," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 1393–1398.

[136] A. Rico, S. Pareek, J. Cabezas, D. Clarke, B. Ozgul, F. Barat, Y. Fu, S. Münz, D. Stuart, P. Schlangen *et al.*, "Amd xdna npu in ryzen ai processors," *IEEE Micro*, vol. 44, no. 6, pp. 73–82, 2024.

[137] Y. Sada, M. Shimoda, A. Jinguji, and H. Nakahara, "A dataflow pipelining architecture for tile segmentation with a sparse mobilenet on an fpga," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 267–270.

[138] C. Sakhuja, Z. Shi, and C. Lin, "Leveraging domain information for the efficient automated design of deep learning accelerators," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 287–301.

[139] 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.

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

[141] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[142] 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.

[143] 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.

[144] Y. S. Shao, J. Cemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Simba: scaling deep-learning inference with chiplet-based architecture," *Communications of the ACM*, vol. 64, no. 6, pp. 107–116, 2021.

[145] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, 2019, pp. 14–27.

[146] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 97–108, 2014.

[147] N. K. Shaydyuk and E. B. John, "Fpga implementation of mobilenetv2 cnn model using semi-streaming architecture for low power inference applications," in *2020 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/-SocialCom/SustainCom)*. IEEE, 2020, pp. 160–167.

[148] G. Shen, J. Zhao, Z. Wang, Z. Lin, W. Ding, C. Wu, Q. Chen, and M. Guo, "Mars: Exploiting multi-level parallelism for dnn workloads on adaptive multi-accelerator systems," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[149] Y. Shen, M. Ferdman, and P. Milder, "Overcoming resource underutilization in spatial cnn accelerators," in *2016 26th International Conference on field programmable logic and applications (FPL)*. IEEE, 2016, pp. 1–4.

[150] Y. Shen, M. Ferdman, and P. Milder, "Maximizing cnn accelerator efficiency through resource partitioning," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 535–547, 2017.

[151] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[152] J. Su, J. Faraone, J. Liu, Y. Zhao, D. B. Thomas, P. H. Leong, and P. Y. Cheung, "Redundancy-reduced mobilenet acceleration on reconfigurable logic for imagenet classification," in *International Symposium on Applied Reconfigurable Computing*. Springer, 2018, pp. 16–28.

[153] A. Symons, L. Mei, S. Colleman, P. Houshmand, S. Karl, and M. Verhelst, "Stream: A modeling framework for fine-grained layer fusion on multi-core dnn accelerators," in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2023, pp. 355–357.

[154] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[155] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, no. 1, 2017.

[156] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[157] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[158] E. Talpes, D. D. Sarma, D. Williams, S. Arora, T. Kunjan, B. Floering, A. Jalote, C. Hsiong, C. Poorna, V. Samant *et al.*, "The microarchitecture of dojo, tesla's exa-scale computer," *IEEE Micro*, vol. 43, no. 3, pp. 31–39, 2023.

[159] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 2820–2828.

[160] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[161] Z. Tan, H. Cai, R. Dong, and K. Ma, "Nn-baton: Dnn workload orchestration and chiplet granularity exploration for multichip accelerators," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1013–1026.

[162] K. Trebing, T. Stanczyk, and S. Mehrkanoon, "Smaat-unet: Precipitation nowcasting using a small attention-unet architecture," *Pattern Recognition Letters*, vol. 145, pp. 178–186, 2021.

[163] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, 2017, pp. 65–74.

[164] N. Vasilache, O. Zinenko, T. Theodoridis, P. Goyal, Z. DeVito, W. S. Moses, S. Verdoolaege, A. Adams, and A. Cohen, "Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions," *arXiv preprint arXiv:1802.04730*, 2018.

[165] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[166] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: A framework for mapping convolutional neural networks on fpgas," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2016, pp. 40–47.

[167] S. I. Venieris and C.-S. Bouganis, "Latency-driven design for fpga-based convolutional neural networks," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2017, pp. 1–8.

[168] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: Mapping regular and irregular convolutional neural networks on fpgas," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 2, pp. 326–342, 2018.

[169] S. I. Venieris, C.-S. Bouganis, and N. D. Lane, "Multiple-deep neural network accelerators for next-generation artificial intelligence systems," *Computer*, vol. 56, no. 3, pp. 70–79, 2023.

[170] S. Venkataramani, A. Ranjan, S. Banerjee, D. Das, S. Avancha, A. Jagannathan, A. Durg, D. Nagaraj, B. Kaul, P. Dubey *et al.*, "Scaledeep: A scalable compute architecture for learning and evaluating deep networks," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 13–26.

[171] R. Venkatesan, Y. S. Shao, M. Wang, J. Clemons, S. Dai, M. Fojtik, B. Keller, A. Klinefelter, N. Pinckney, P. Raina *et al.*, "Magnet: A modular accelerator generator for neural networks," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[172] M. Verhelst, M. Shi, and L. Mei, "Ml processors are going multi-core: A performance dream or a scheduling nightmare?" *IEEE Solid-State Circuits Magazine*, vol. 14, no. 4, pp. 18–27, 2022.

[173] L. Wei and L. Peng, "An efficient opencl-based fpga accelerator for mobilenet," in *Journal of Physics: Conference Series*, vol. 1883, no. 1. IOP Publishing, 2021, p. 012086.

[174] 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 *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. ACM, 2018, pp. 1–8.

[175] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "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.

[176] D. Wu, Y. Zhang, X. Jia, L. Tian, T. Li, L. Sui, D. Xie, and Y. Shan, "A high-performance cnn processor based on fpga for mobilenets," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 136–143.

[177] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[178] M. Xia, Z. Huang, L. Tian, H. Wang, V. Chang, Y. Zhu, and S. Feng, "Sparknoc: An energy-efficiency fpga-based accelerator using optimized lightweight cnn for edge computing," *Journal of Systems Architecture*, vol. 115, p. 101991, 2021.

[179] J. Xiao, Y. Chen, and T. Su, "A mobilenet accelerator with high processing-element-efficiency on fpga," in *2021 China Semiconductor Technology International Conference (CSTIC)*. IEEE, 2021, pp. 1–3.

[180] 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.

[181] X. Xie, G. Zhao, W. Wei, and W. Huang, "Mobilenetv2 accelerator for power and speed balanced embedded applications," in *2022 IEEE 2nd International Conference on Data Science and Computer Application (ICDSCA)*. IEEE, 2022, pp. 134–139.

[182] Z. Xie, K. Dai, Z. Wu, J. Wang, X. Lu, and S. Liu, "Design space exploration of cnn accelerators based on gsa algorithm," in *2024 9th International Conference on Signal and Image Processing (ICSIP)*. IEEE, 2024, pp. 319–323.

[183] Xilinx Inc., *Xilinx Power Estimator (XPE)*. [Online]. Available: https://www.amd.com/en/products/adaptive-socs-and-fpgas/technologies/power-efficiency/power-estimator.html

[184] R. Xu, S. Ma, Y. Wang, and Y. Guo, "Hesa: Heterogeneous systolic array architecture for compact cnns hardware accelerators," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 657–662.

[185] R. Xu, S. Ma, Y. Wang, Y. Guo, D. Li, and Y. Qiao, "Heterogeneous systolic array architecture for compact cnns hardware accelerators," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2860–2871, 2021.

[186] S. Yan, Z. Liu, Y. Wang, C. Zeng, Q. Liu, B. Cheng, and R. C. Cheung, "An fpga-based mobilenet accelerator considering network structure characteristics," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2021, pp. 17–23.

[187] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina *et al.*, "Interstellar: Using halide's scheduling language to analyze dnn accelerators," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 369–383.

[188] Y. Yang, J. S. Emer, and D. Sanchez, "Isosceles: Accelerating sparse cnns through inter-layer pipelining," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 598–610.

[189] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *Advances in neural information processing systems*, vol. 32, 2019.

[190] H. Ye, X. Zhang, Z. Huang, G. Chen, and D. Chen, "Hybriddnn: A framework for high-performance hybrid dnn accelerator design and implementation," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[191] Z. Yu and C.-S. Bouganis, "Auto ws: Automate weights streaming in layer-wise pipelined dnn accelerators," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2024, pp. 1–6.

[192] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[193] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang *et al.*, "Big bird: Transformers for longer sequences," *Advances in neural information processing systems*, vol. 33, pp. 17 283–17 297, 2020.

[194] 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.

[195] D. Zhang, S. Huda, E. Songhori, K. Prabhu, Q. Le, A. Goldie, and A. Mirhoseini, "A full-stack search technique for domain optimized deep learning accelerators," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 27–42.

[196] 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.

[197] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "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.

[198] X. Zhang, H. Ye, and D. Chen, "Being-ahead: Benchmarking and exploring accelerators for hardware-efficient ai deployment," *arXiv preprint arXiv:2104.02251*, 2021.

[199] X. Zhang, H. Ye, J. Wang, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "Dnnexplorer: a framework for modeling and exploring a novel paradigm of fpga-based dnn accelerator," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[200] T. Zhao, L. Qiao, Q. Chen, Q. Zhang, and N. Li, "A hardware accelerator based on neural network for object detection," in *Journal of Physics: Conference Series*, vol. 1486, no. 2. IOP Publishing, 2020, p. 022045.

[201] S. Zheng, X. Zhang, L. Liu, S. Wei, and S. Yin, "Atomic dataflow based graph-level workload orchestration for scalable dnn accelerators," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 475–489.

[202] Y. Zhou, M. Yang, C. Guo, J. Leng, Y. Liang, Q. Chen, M. Guo, and Y. Zhu, "Characterizing and demystifying the implicit convolution algorithm on commercial matrix-multiplication accelerators," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2021, pp. 214–225.