MCExplorer: Exploring the Design Space of Multiple Compute-Engine Deep Learning Accelerators

MCExplorer: Exploring the Design Space of Multiple Compute-Engine Deep Learning Accelerators

FAREED QARARYAH, Computer Science And Engineering, Chalmers University of Technology, Goteborg, Sweden and University of Gothenburg, Gothenburg, Sweden
MOHAMMAD ALI MALEKI, Computer Science And Engineering, Chalmers University of Technology, Gothenburg, Sweden and University of Gothenburg, Gothenburg, Sweden
PEDRO TRANCOSO, Chalmers University of Technology, Computer Science And Engineering, Sweden and University of Gothenburg, Gothenburg, Sweden

Model-aware Deep Learning (DL) accelerators surpass generic ones in terms of performance and efficiency. These model-aware accelerators typically comprise *multiple* dedicated *Compute Engines (CEs)* to handle the varying computational characteristics of the operations within a DL model. Multiple-CE accelerators usually target Field-Programmable Gate Arrays (FPGAs), as FPGAs' reconfigurability enables tailoring the CEs architectures to the varying computational characteristics of the model operations. The continuous evolution of DL models and their use in application domains with diverse optimization objectives, including low latency, high throughput, and energy efficiency, makes it challenging to identify highly optimized multiple-CE accelerator architectures. The design space of multiple-CE accelerators is vast, and the state of the art explores only limited parts of this space, which hinders the identification of accelerators with high performance and efficiency.

To address this challenge, we propose a framework for exploring the design space of FPGA-based multiple-CE accelerators (MC**E**xplorer). MC**E**xplorer comprises a set of single and multiobjective optimization heuristics that target throughput, latency, energy efficiency, and tradeoffs among these metrics. MC**E**xplorer searches for optimized multiple-CE accelerator architectures given a DL model, a hardware resource budget, and a single or multiple objectives. MC**E**xplorer explores a space beyond that explored in the literature by not restricting the accelerator inter-CE arrangements and exploring distinct configurations of individual CEs. We evaluate MC**E**xplorer with various DL models and hardware resource budgets. The evaluation shows that by exploring a search space beyond that in the literature, MC**E**xplorer identifies highly optimized multiple-CE accelerators. These accelerators achieve up to 2.8× throughput improvement, 2.1× speedup, and 45% energy reduction compared to the state of the art. Moreover, the evaluation demonstrates that broad space exploration is key

This work was supported by the VEDLIoT project, which received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 957197. The work was also supported by the Swedish Foundation for Strategic Research (contract number CHI19-0048) under the PRIDE project and the European High-Performance Computing Joint Undertaking (JU) under Framework Partnership Agreement No 800928 and Specific Grant Agreement No 101036168 (EPI SGA2). The JU is supported by the European Union's Horizon 2020 research and innovation program and by Croatia, France, Germany, Greece, Italy, Netherlands, Portugal, Spain, Sweden, and Switzerland.
Authors' Contact Information: Fareed Qararyah (corresponding author), Computer Science and Engineering, Chalmers University of Technology, Goteborg, Sweden and University of Gothenburg, Gothenburg, Sweden; e-mail: qarayah@chalmers.se; Mohammad Ali Maleki, Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden and University of Gothenburg, Gothenburg, Sweden; e-mail: mohammad.ali.maleki@chalmers.se; Pedro Trancoso, Chalmers University of Technology, Computer Science and Engineering, Sweden and University of Gothenburg, Gothenburg, Sweden; e-mail: ppedro@chalmers.se.

ACM 1544-3973/2025/12-ART157
https://doi.org/10.1145/3774913

to identifying multiple-CE accelerators with the best performance-efficiency tradeoffs. MC**E**xplorer code is available at https://github.com/fqararyah/MCExplorer.

CCS Concepts: • **Computer systems organization → Parallel architectures**; • **Hardware → Reconfigurable logic and FPGAs**; • **Computing methodologies → Neural networks**; **Search methodologies;**

Additional Key Words and Phrases: Design space exploration (DSE), deep learning accelerators, multiple compute-engine accelerators, field-programmable gate arrays (FPGAs)

ACM Reference Format:
Fareed Qararyah, Mohammad Ali Maleki, and Pedro Trancoso. 2025. MC**E**xplorer: Exploring the Design Space of Multiple Compute-Engine Deep Learning Accelerators. *ACM Trans. Arch. Code Optim.* 22, 4, Article 157 (December 2025), 27 pages. https://doi.org/10.1145/3774913

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 [4, 5, 53, 63]. As a result, there is a trend toward modular accelerators composed of multiple CEs [6, 12, 38, 45]. 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 [6, 38, 63].

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 [3, 38, 44, 50, 54, 61, 63]. These model-aware multiple-CE accelerators usually outperform generic ones, considering different optimization goals and under varying resource budgets [38, 50, 63]. However, the design space of multiple-CE accelerators has not been adequately explored. Hence, the potential of such accelerators has not been fully leveraged [39].

Because an exhaustive exploration of the design space of multiple-CE accelerators is impractical [39, 44, 63], 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 [3, 44, 54, 61, 63]. *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 [3, 44, 54, 61]. For example, Wei et al. [54] 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 [38]. 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**) [39]. 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 [54], others target throughput [3, 44].

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**xplorer is the first framework that does not restrict accelerator inter-CE arrangements and considers multiple distinct per-CE configurations, including parallelism strategies and dataflow 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 tradeoffs among these metrics. MCExplorer 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 MCExplorer, a framework that explores the design space of multiple-CE accelerators with flexible inter-CE arrangements and per-CE configurations. MCExplorer 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 tradeoffs. We use the optimizer to identify multiple-CE accelerators that simultaneously optimize performance and energy efficiency.
- We demonstrate the effectiveness of MCExplorer through an extensive evaluation using representative DNNs and hardware resource budgets. The evaluation shows that MCExplorer identifies optimized multiple-CE accelerators that outperform the state of the art, considering different optimization objectives.

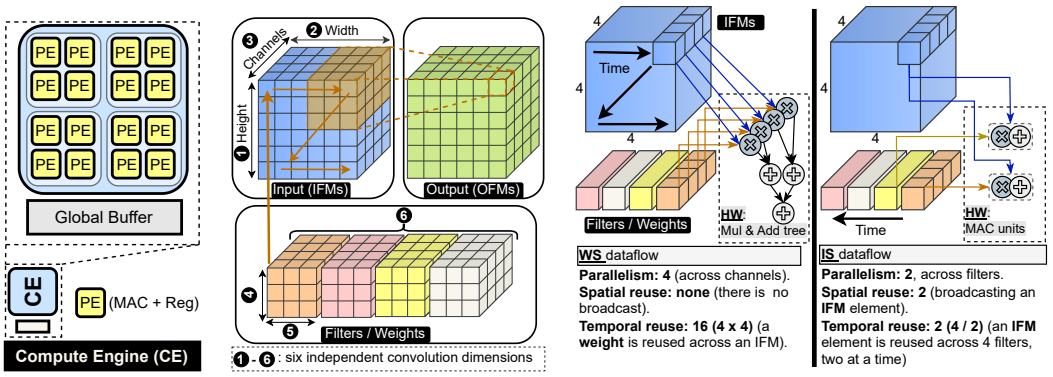
Our evaluation of MCExplorer with various DL models and FPGA boards shows that exploring the design space of multiple-CE accelerators without restricting the inter-CE arrangements and per-CE configurations results in accelerators that outperform state of the art in all targeted optimization objectives, 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 multiple-CE accelerators with the best performance-efficiency tradeoffs.

2 Background and Motivation

2.1 Compute Engines and Their Parallelism Strategies and Dataflow

We use the term *CE* to describe an array of **Processing Elements** (**PEs**) that cannot be independently controlled but work as a unit to compute a workload. Figure 1(a) shows a high-level overview of a CE. In this context, a PE is a 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 [27], our CEs mainly perform convolution operations. Figure 1(b) 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 CE are largely determined by its *parallelism strategy* and *dataflow* [23]. 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 1(b). The CEs in most state-of-the-art accelerators apply parallelism on one to three of the six dimensions [2, 6, 27, 28, 34, 44, 50, 60, 61, 63]. Figure 1(c) shows two examples of simple one-dimensional parallelism. The parallelism strategy dictates the required hardware, PE utilization, and data reuse. A mismatch between FMs and weight topologies of a layer and the parallelism dimensions of a CE results in PE underutilization [39]. PE underutilization



(a) Compute Engine high-level overview.

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

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

Fig. 1. CE overview, convolution operation, and simplified convolution engines parallelism and dataflow examples.

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 1(c) 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 [25]. 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 1(c) 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 × IFM height).

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 [4, 6, 54]. 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 [41, 53]. This increase can be mitigated by having large and high-bandwidth on-chip memory, but this results in area and energy overhead [4, 53]. 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 [56, 59]. For example, Xie et al. [56] 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

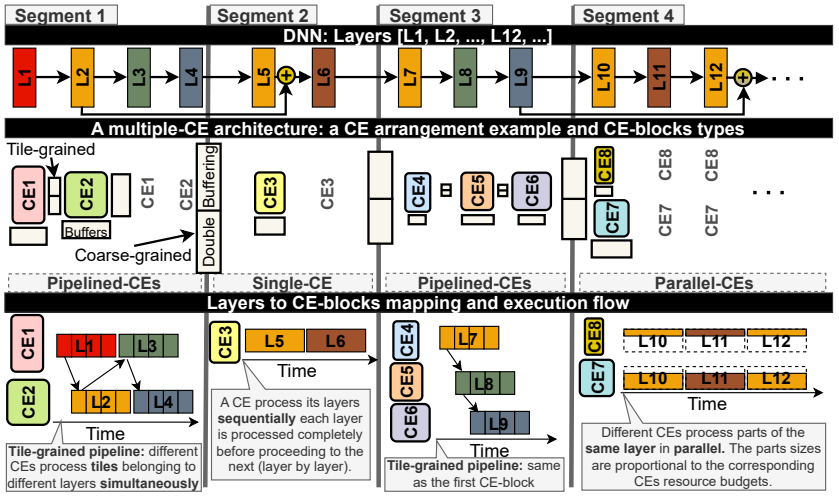


Fig. 2. DNN to a multiple-CE architecture mapping example. The example shows four segments that divide the figure vertically, and three sections that divide the figure horizontally. The sections show DNN layers, the types of CE-blocks used, and the mapping and execution flow of the layers on their CE-blocks. CEs and buffers vary in size and shape, proportional to their segment layers' needs.

terms of area and energy [9, 62]. 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 [5, 6, 12, 43, 44, 51, 54].

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 [3, 35, 38, 44, 50, 54, 61, 63]. A recent survey shows that single-engine approaches are becoming a minority among FPGA acceleration toolflows [19]. Figure 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 [39]. 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 2. A Parallel-CEs block improves the scalability and enables higher performance when further scaling of a single-CE block becomes

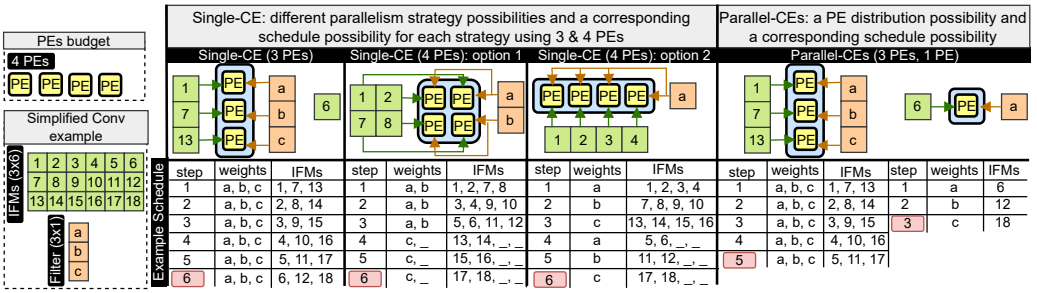
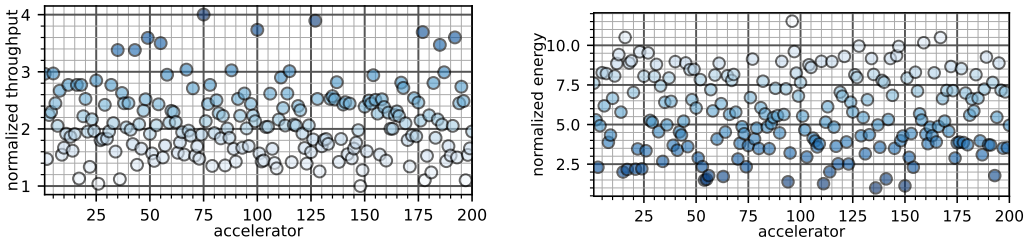


Fig. 3. A simplified convolution example. PE distributions and example schedules of single-CE and parallel-CEs blocks using up to 4 PEs. The examples use a convolution with a single 3×1 filter and 3×6 IFM. A single-CE requires 6 steps using 3 PEs and does not scale effectively when using 4 PEs. Arranging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

ineffective. Figure 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 1(b). The choice of dimensions to partition across impacts data sharing. To give an example, considering the simplified convolution in Figure 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 spitting 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.

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 [39]. However, the arrangement of the CEs has a considerable impact on the accelerator performance and efficiency. Figure 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 [14] 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.

Fig. 4. Throughput and energy of 200 randomly generated multiple-CE accelerators. The values are obtained using MCCM [39].

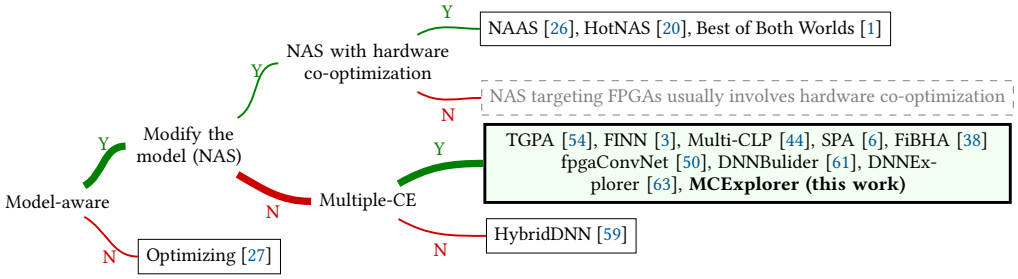


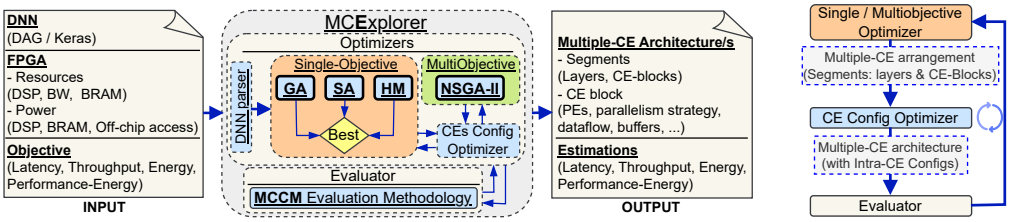
Fig. 5. The scope of this work within the broader context of FPGA-based deep learning accelerator design approaches.

2.5 Scope of This Work

Figure 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 article. 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.

2.6 Design Space Exploration Metaheuristics

2.6.1 Genetic Algorithm. GA are a population-based metaheuristic inspired by natural selection [16]. 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 *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



(a) MCExplorer core modules. (b) MCExplorer main loop.

Fig. 6. A high-level overview of MCExplorer components and its main optimization loop.

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 [23, 33].

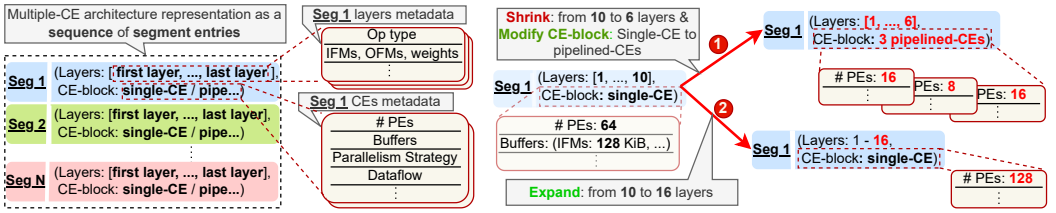
2.6.2 Simulated Annealing. SA is a single-solution-based probabilistic metaheuristic inspired by the physical process of solid annealing [24]. SA starts with a usually randomly generated initial solution and a high *temperature*. The temperature in SA is a hyperparameter corresponding to the probability 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 [5, 22, 64]

2.6.3 Non-Dominated Sorting Genetic Algorithm. **Non-Dominated Sorting Genetic Algorithm II (NSGA-II)** is a type of GA 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 tradeoffs (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* [10]. 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 [33].

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

Figure 6(a) shows a high-level overview of MCExplorer. MCExplorer takes as *inputs*: **(1)** DNN representation. **(2)** FPGA's 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 tradeoff. 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



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

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

Fig. 7. Multiple-CE architecture unified representation and change operator.

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 article 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, *GA*, and *SA*, and a novel heuristic named *HM*. GA is a scalable alternative that has been shown to outperform various optimization methods, including **Reinforcement Learning** (**RL**) [23, 40, 49]. We selected SA as it has provided high-quality results and relatively fast convergence on a related problem, namely scheduling DNNs on tiled architectures [5, 64]. In addition to these metaheuristics, we proposed HM with a special focus on speed and scalability, as described in Section 3.2.3. The user can select to use MCExplorer 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 tradeoff. We formulate a multiobjective optimizer based on *NSGA-II* [10]. NSGA-II is a commonly used multiobjective optimization metaheuristic that has been shown to outperform RL in finding Pareto-optimal solutions [33]. **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 2, adopting a unified parallelism strategy and dataflow across all CEs would lead to suboptimal results. MCExplorer'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 [39]. MCCM evaluates the performance and resource efficiency of multiple-CE accelerators in a few milliseconds. The flow of the main loop of MCExplorer optimization is depicted in Figure 6(b). 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 MCExplorer and the optimization process are described in detail in the rest of this section.

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 7(a) 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 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 2.6), the sequence of segment entries corresponds to a chromosome, and each entry corresponds to a gene. In the SA (Section 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 [5]. 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 layers 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 7(b) describes two examples of the change operator. The first example (❶) 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 (❷) 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 3.4.

3.2 Single-Objective Optimizers

MC**E**xplorer 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 article focuses on the first three objectives. However, MC**E**xplorer 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 metaheuristics-based optimizers, namely GA and SA, and a fast heuristic-based optimizer named HM.

3.2.1 Genetic Algorithm-Based Optimizer. The proposed GA utilizes the *unified representation* described in Section 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 [39], 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.

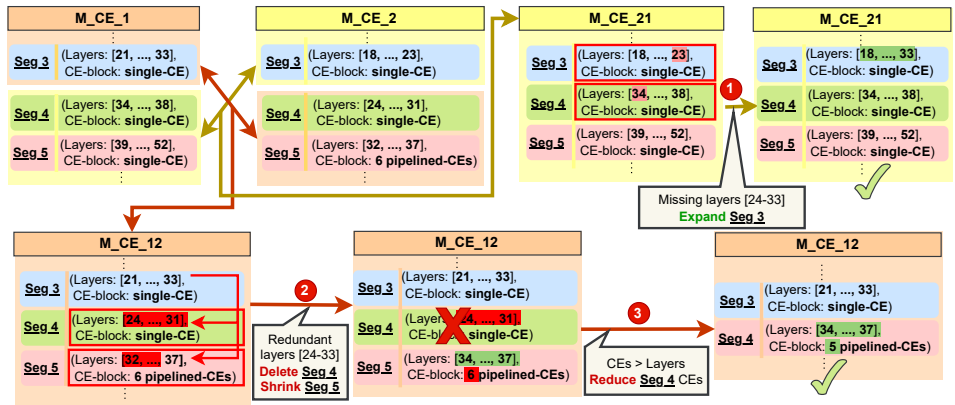


Fig. 8. An example of the special crossover operator handling of the flexible-length genes (i.e., segments) and chromosomes ((i.e., multipl-CE architecture representations). The example depicts crossover between *M_CE_1* and *M_CE_2* to produce *M_CE_12* and *M_CE_21*. Initially, neither *M_CE_12* nor *M_CE_21* is a valid mapping, which triggers the depicted set of modifications.

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 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 3.1); or deleting whole segments. Algorithm 1 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 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 3.1. 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.

ALGORITHM 1: Single-Point Crossover

Input : Crossover probability P_c, parents p_1, p_2
Output: Offspring c_1, c_2

if random() < P_c **then**
  $crossPt \leftarrow$ randInt(max(1, min(numSegments(p_1), numSegments($P2$) - 1))) // Crossover point
  $piece_{11} \leftarrow p_1[0 : crossPt]$, $piece_{12} \leftarrow p_1[crossPt + 1$:numSegments($p_1$)$]$
  $piece_{21} \leftarrow p_2[0 : crossPt]$, $piece_{22} \leftarrow p_2[crossPt + 1$: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
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
  $offspring \leftarrow$ combine($piece_1$, $piece_2$) // After being modified, combining the pieces guarantees valid offspring
  return R

ALGORITHM 2: Hierarchical Mapping

Input : minimum clusters C_{min}, maximum clusters C_{max}, DNN representation *dnnConfig*, FPGA resources *hwConfig*, optimization objective *optObjective*
Output: multiple-CE accelerator description $mulCE_{best}$

Initialize($mulCE_{best}$, *dnnConfig*, *hwConfig*)
for $numClusters \leftarrow C_{min}$ **to** C_{max} **do**
  $layerClusters \leftarrow$ cluster(*numClusters*, *dnnConfig*)
  $mulCE_{localBest} \leftarrow$ exploreMappings(*layerClusters*, *hwConfig*, *optObjective*)
  $mulCE_{localBest} \leftarrow$ refine($mulCE_{localBest}$, *optObjective*)
  $mulCE_{best} \leftarrow$ best($mulCE_{best}$, $mulCE_{localBest}$, *optObjective*)
return $mulCE_{best}$

3.2.2 Simulated Annealing-Based Optimizer. As in the GA case, we use the unified representation described in Section 3.1 for the SA. The adopted termination criterion is the number of iterations. We use the change operator described in Section 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 [39] 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.

3.2.3 Hierarchical Mapping. We propose 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-optimization, where the design spaces of both the model and the accelerator are explored [1, 26]. Algorithm 2 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 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 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 $\approx$ 2 thousand possibilities. In other words, clustering shrinks the design space roughly 50 billion times in this example. HM clustering (line 3 in Algorithm 2) is mainly based on *K-means* clustering [29]. 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 [39]. While the concurrent processing reduces FMs' data movement, it requires more weight data movement [30, 39]. 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 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 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 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 [39]. 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.

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 an 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 tradeoffs. To identify such accelerators, we formulate a NSGA-II metaheuristic. NSGA-II identifies a well-distributed set of optimal tradeoffs, i.e., Pareto front [10], as described in Section 2.6.3.

In this work, we consider optimizing throughput-energy and latency-energy tradeoffs of multiple-CE accelerators. Our NSGA-II uses the same termination criteria, the unified representation, and the mutation operator used in GA (Section 3.2.1). However, in our experiments, a higher mutation probability compared to GA (details in Section 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 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 2.6.3).

3.4 CE Configuration Optimizer

The main goal of the optimizers described in Sections 3.2 and 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 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. MC**E**xplorer'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 7(b).

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 [3, 38]. 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 [22, 23, 31, 37]. As a result, applying state-of-the-art techniques to each explored multiple-CE architecture is not an option.

Table 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

Fortunately, there are a few well-known parallelism strategies, such as those of NVDLA [36], DPU [55], and ShiDianNao [11], 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 2.1), and **Output Stationary-Local Weight Stationary** (**OS-LWS**) and **Weight Stationary - Local Output Stationary** (**WS-LOS**) [52] 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 Experimental Setup

4.1 Platforms and Systems

Table 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.* We use **high-level synthesis** (**HLS**) to implement and validate a sample of the multiple-CE architectures suggested by MCExplorer. Vitis-IDE (2021.2) is used for synthesis and validation. MCExplorer 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.

4.2 Workloads

Table 2 summarizes the models used in the evaluation. We evaluate MCExplorer using a representative set of **Convolutional Neural Networks** (**CNNs**) that have varying structures, depths, and weight sizes. These are ResNet152 and ResNet50 [14], XCeption [8], DenseNet [17], and MobileNetV2 [42]. 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 [46] and EfficientNet [47]. The residual blocks of ResNets and the Extreme Inception blocks of XCeption are also used in recent CNNs. We also evaluate MCExplorer using models with heterogeneous backbones. These are: CMT [13], a hybrid architecture that combines CNNs with transformer-based modules in a unified backbone, and SmAt-UNet [48],

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

Table 2. Evaluated DL Models

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

which is based on UNet architecture equipped with attention modules and depthwise-separable convolutions.

4.3 Baseline Architectures

We evaluate MCExplorer 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 2.3) [39]. 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 [39] to model the following three state-of-the-art accelerators, representing the three categories, using a common set of models and boards:

- **FiBHA** [38]. There is a handful of hybrid FPGA-based accelerators in the literature. These include Alwani et al. [2], DNNExplorer [63], Nguyen et al. [35], 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** [44]. 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** [54]. Several accelerators in the literature can be categorized as SegmentedRR [3, 6, 50, 54, 61]. 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** (**PD**) [34, 54] or DSP efficiency [19, 63]. We use a PD metric defined in Equation (1).

$$\text{PD} = \frac{\text{GOP/s}}{\textit{Frequency} \times \textit{number of DSP Slices}^{\dagger} \times \alpha} \tag{1}$$

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 5.5 we compare with two DSE frameworks which achieve highly optimized performance and efficiency, namely fpgaConvNet [50] and SPA [6]. In general, all baseline architectures are made

*†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.

up of different combinations of the CE-blocks discussed in Section 2.3. Due to space limitations, we refer the reader to the corresponding papers for a detailed description of each baseline architecture.

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 [31, 37, 58], as these dominate the overall energy consumption. To estimate energy efficiency, we use MCCM [39] to obtain accelerator PE utilization, on-chip buffer sizes, and the number of accesses to all memory levels. We utilize **Xilinx Power Estimator** (**XPE**) [57] 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 [32].

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 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 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 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.

5 Evaluation

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

5.1.1 Using MCExplorer as a Black-Box Optimizer. This section analyzes the results of running MCExplorer 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 4.3. 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 9 compares the throughput, latency, and energy efficiency achieved by MCExplorer suggested accelerators to Hybrid, Segmented, and SegmentedRR baselines described in Section 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

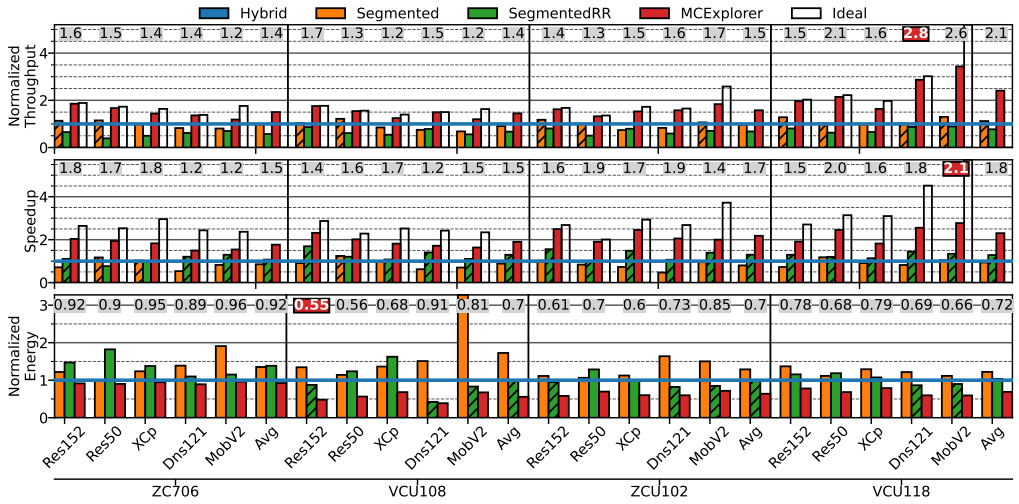


Fig. 9. Throughput, latency, and energy efficiency comparison with the baselines. Hatched baseline bars indicate the best among the three baselines. Experiments with no hatched baseline bars are those where Hybrid (the normalization anchor) is best among the baselines. The labels at the top of the figures report MC**E**xplorer results normalized to the best of the three baselines in an experiment.

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 [38]. 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 MC**E**xplorer; in general, close to the 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 [5, 39, 64]. Considering *energy efficiency*, MC**E**xplorer suggested accelerators use down to 55% of the energy of to 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.

*5.1.2 Heterogeneity of CE Arrangements and CE Configurations Among MC**E**xplorer 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 3(a) summarizes the contributions of these two differences in the sixty experiments depicted in Figure 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 3.4). The breakdown in Table 3(b) 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 3(a). This shows that in most cases, flexibility

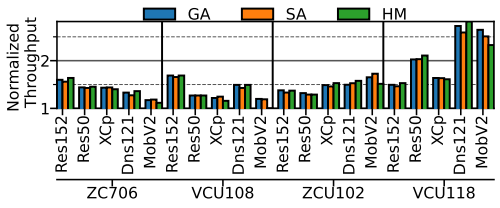
Table 3. CE-blocks and Per-CE Configurations Heterogeneity

(a) Summary of results of the best-performing accelerators in the 60 experiments discussed in Section 5.1.1

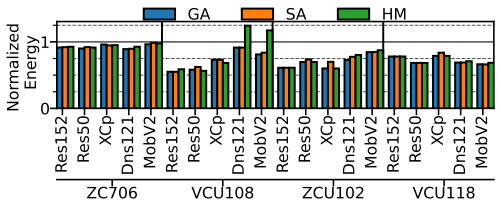
		CE Configurations (parallelism strategy and dataflow)		
		Homogeneous	Heterogeneous	Sum
CE-blocks	Homogeneous	1 (1.67%)	19 (31.67)%	20 (33.33%)
	Heterogeneous	0 (0%)	40 (66.67%)	40 (66.67%)
	Sum	1 (1.67%)	59 (98.33%)	60 (100%)

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



(a) Throughput comparison of different optimizers.



(b) Energy per inference comparison of different optimizers.

Fig. 10. Comparison of different optimizer-suggested accelerators, normalized to the best of Segmented, SegmentedRR, and Hybrid.

in both CE arrangements and per-CE configurations achieves the best results. The breakdown in Table 3(b) 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 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.

5.1.3 Comparing Single-Objective Optimizers. Figure 10 compares the throughput and energy efficiency of multiple-CE accelerators suggested by GA, SA, and HM. The reason for equipping MCExplorer with GA and SA is that both have achieved high-quality results on related problems [5, 23, 40, 49, 64]. As Figure 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 10(a). Moreover, although both explore the same number of data points (Section 4.5), SA is considerably faster. This is shown in Table 4(a), which presents the execution times of the three heuristics. On the other hand, as Table 4(b) 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 MCExplorer with HM. By pruning the design space, HM is faster and scales better than GA and SA. Table 4(a) 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 layers [15, 18], or when used in NAAS in the context of a NAS with hardware co-optimization,

Table 4. Optimizers Execution Time and Stability Considering All Boards and CNNs

(a) 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

(b) 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%

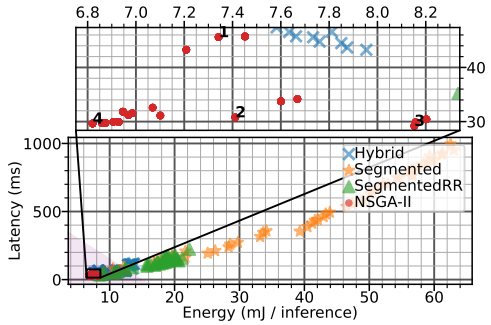
where the design space expands as both the DNN and the accelerator architectures are explored. Regarding the quality of HM results, Figure 10 shows that HM found multiple-CE architectures that outperform the best baseline considering all DNNs and boards. There are a few cases where HM results are similar 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 time-to-solution or quality of results by reducing the chances of getting a local minimum.

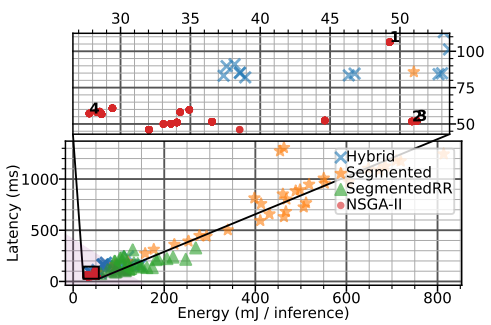
5.2 Multiobjective Optimization: Performance-Energy-Efficiency Tradeoffs

Figure 11 shows some of the performance-energy-efficiency tradeoffs 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 11(b), 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 11(d), 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 tradeoffs, 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.

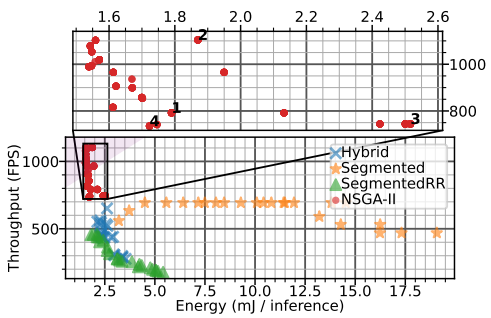
To put NSGA-II results in perspective, we compare them with the three baselines. As can be seen in Figure 11(a)–(d), the Pareto fronts contain architectures that offer better tradeoffs than the three baselines. Comparing the results on the Pareto front with the baselines and comparing the baselines against each other suggests that more heterogeneous and flexible architectures have better tradeoffs. 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 2.3), i.e., single-CE in Segmented and pipelined-CEs in SegmentedRR (Section 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 tradeoffs. However, Hybrid architectures still restrict the CE arrangements; they use the two blocks in a fixed and predefined order. NSGA-II exploration,



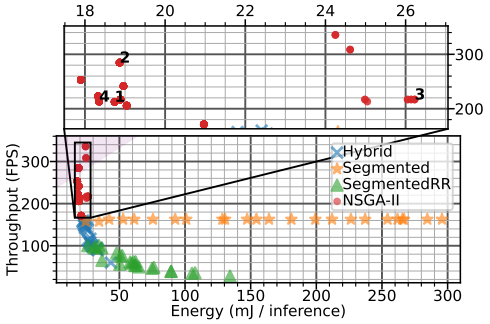
(a) Latency energy trade-off using ZC706 and Dns121.



(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.

Fig. 11. Performance energy tradeoff 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 tradeoffs between the two metrics. The shaded corner points to the best in both metrics.

which restricts neither the CE arrangements nor per-CE configurations, achieves better tradeoffs 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 3.

5.3 Using MCExplorer with DL Models of Heterogeneous Backbones

The evaluation of MCExplorer in the previous sections focused on the design of accelerators for CNNs with pure convolutional backbones. Nevertheless, MCExplorer is also useful for modern DL models of heterogeneous backbones. When used with models of heterogeneous backbones, MCExplorer can search for optimized CE-block arrangements to process the convolutional parts of these models. Figure 12 compares MCExplorer 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, MCExplorer accelerators improve over the baselines in all cases, considering different metrics and boards. These results demonstrate that the applicability of MCExplorer extends beyond CNNs.

5.4 Implementing and Validating MCExplorer Results

The accuracy of MCCM [39], 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

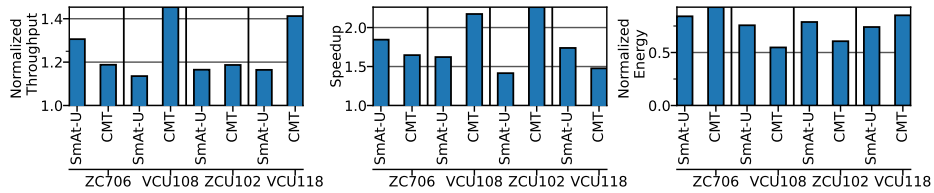


Fig. 12. Throughput, latency, and energy efficiency of MCExplorer suggested accelerators normalized to the best of Hybrid, Segmented, and SegmentedRR using the convolutional backbones of models with heterogeneous backbones.

Table 5. The Accuracy of the Estimated Improvements Compared to the Measured Improvements Using HLS

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

The results are for the accelerators that achieve the best speedup, throughput, and energy efficiency, respectively, using ResNet152 on ZCU102.

Table 6. Direct PD Comparison Multiple-CE Accelerators and DSE Frameworks

Model	Mob_v2					Res152				
Design	FiBHA[38] TACO	SPA[6] MICRO		MCExplorer This work		fpgaConvNet[50] TNNLS	TGPA[54] ICCAD	SPA[6] MICRO	MCExplorer This work	
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	Zynq-7045	VU9P	KU115	ZC706	VCU118
DSP slices	2520	900	5520	900	6840	900	6840	5520	900	6840
Frequency (MHz)	180	200	200	200	200	125	200	200	200	200
Performance(GOP/s) / α	375	242*	1594*	405	3092	188	1463	1583*	2655	353
PD (ops/DSP/cycle)	0.83	1.34	1.17	1.35	**1.36**	1.67	1.07	1.43	1.94	**1.96**

*α = 2, as SPA [6] applies packing of two activations together.

accelerators and compare the estimated improvement with the actual improvement, considering different metrics. The estimation accuracies are reported in Table 5. As the table shows, the estimated improvements are within 9% of the actual ones. While the actual speedup and throughput improvements are higher 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 4.3). The accelerator that achieves the best throughput improvement comprises both single-CE and pipelined-CE blocks. Although it is similar to the Hybrid (Section 4.3) 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 parallel-CEs block (Section 2.3), which is not used by the state-of-the-art.

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

In this section, we compare MCExplorer 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 4.3, we use PD (Equation 1) as a normalizing metric. Table 6 shows the PD

achieved by MC**E**xplorer compared to state-of-the-art using models 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 resources 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.

6 Related Work

The performance and efficiency of single 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 dataflow and scheduling, reduced precision arithmetic, and the use of structured and unstructured sparsity [7, 21, 53]. However, these improvements have plateaued due to the inherent bottlenecks of single-CE accelerators [53]. Consequently, to maintain improvements in performance and efficiency, many state-of-the-art DL accelerators are adopting multiple-CE architectures [5, 6, 12, 43, 44, 51, 54]. 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 [3, 61]. 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, 6, 35, 38, 44, 54, 63]. 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 dataflow and parallelism strategy, with different degrees of parallelism, in all CEs [44, 54, 61, 63]. In other words, the previous work constrains different design aspects and does not adequately explore the design space of multiple-CE accelerators [39]. Moreover, previous work usually optimizes for a single performance metric, e.g., latency [54] or throughput [3, 44], 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 [39], MC**E**xplorer effectively explores a design space beyond that of previous work, identifying multiple CE accelerators with better performance and efficiency.

7 Conclusion

In this article, we proposed 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 tradeoffs 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 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 tradeoffs.

References

[1] Mohamed S. Abdelfattah, Łukasz Dudziak, Thomas Chau, Royson Lee, Hyeji Kim, and Nicholas D. Lane. 2020. Best of both worlds: AutoML codesign of a CNN and its hardware accelerator. In *Proceedings of the 2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[2] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-layer CNN accelerators. In *Proceedings of the 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–12.

[3] Michaela Blott, Thomas B. Preußer, Nicholas J. Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. 2018. FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 11, 3 (2018), 1–23.

[4] Amirali Boroumand, Saugata Ghose, Berkin Akin, Ravi Narayanaswami, Geraldo F. Oliveira, Xiaoyu Ma, Eric Shiu, and Onur Mutlu. 2021. Google neural network models for edge devices: Analyzing and mitigating machine learning inference bottlenecks. In *Proceedings of the 2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 159–172.

[5] Jingwei Cai, Yuchen Wei, Zuotong Wu, Sen Peng, and Kaisheng Ma. 2023. Inter-layer scheduling space definition and exploration for tiled accelerators. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–17.

[6] Xuyi Cai, Ying Wang, Xiaohan Ma, Yinhe Han, and Lei Zhang. 2022. DeepBurning-SEG: Generating DNN accelerators of segment-grained pipeline architecture. In *Proceedings of the 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1396–1413.

[7] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (2019), 292–308.

[8] François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1251–1258.

[9] Steven Colleman, Man Shi, and Marian Verhelst. 2023. COAC: Cross-layer optimization of accelerator configurability for efficient CNN processing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 31, 7 (2023), 945–958.

[10] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.

[11] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. 92–104.

[12] Mingyu Gao, Xuan Yang, Jing Pu, Mark Horowitz, and Christos Kozyrakis. 2019. Tangram: Optimized coarse-grained dataflow for scalable NN accelerators. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems*. 807–820.

[13] Jianyuan Guo, Kai Han, Han Wu, Yehui Tang, Xinghao Chen, Yunhe Wang, and Chang Xu. 2022. Cmt: Convolutional neural networks meet vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12175–12185.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *ECCV 2016: Proceedings of the 14th European Conference on Computer Vision, Part IV 14*. Springer, 630–645.

[16] John H. Holland. 1992. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press.

[17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4700–4708.

[18] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. 2016. Deep networks with stochastic depth. In *ECCV 2016: Proceedings of the 14th European Conference on Computer Vision, Part IV 14*. Springer, 646–661.

[19] Junye Jiang, Yaan Zhou, Yuanhao Gong, Haoxuan Yuan, and Shuanglong Liu. 2025. FPGA-based acceleration for convolutional neural networks: A comprehensive review. *ArXiv*. arXiv:2505.13461. Retrieved from https://arxiv.org/abs/2505.13461

[20] Weiwen Jiang, Lei Yang, Sakyasingha Dasgupta, Jingtong Hu, and Yiyu Shi. 2020. 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* 39, 11 (2020), 4154–4165.

[21] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 1–12.

[22] Victor JB Jung, Arne Symons, Linyan Mei, Marian Verhelst, and Luca Benini. 2023. SALSA: Simulated annealing based loop-ordering scheduler for DNN accelerators. In *Proceedings of the 2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 1–5.

[23] Sheng-Chun Kao and Tushar Krishna. 2020. GAMMA: Automating the HW mapping of DNN models on accelerators via genetic algorithm. In *Proceedings of the 39th International Conference on Computer-Aided Design*. 1–9.

[24] Scott Kirkpatrick, C. Daniel Gelatt Jr, and Mario P. Vecchi. 1983. Optimization by simulated annealing. *Science* 220, 4598 (1983), 671–680.

[25] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. 2019. 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*. 754–768.

[26] Yujun Lin, Mengtian Yang, and Song Han. 2021. NAAS: Neural accelerator architecture search. In *Proceedings of the 2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1051–1056.

[27] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. 2018. Optimizing the convolution operation to accelerate deep neural networks on FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26, 7 (2018), 1354–1367.

[28] Yufei Ma, Naveen Suda, Yu Cao, Jae-sun Seo, and Sarma Vrudhula. 2016. Scalable and modularized RTL compilation of convolutional neural networks onto FPGA. In *Proceedings of the 2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 1–8.

[29] James MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, Vol. 5. University of California Press, 281–298.

[30] Linyan Mei, Koen Goetschalckx, Arne Symons, and Marian Verhelst. 2023. Defines: Enabling fast exploration of the depth-first scheduling space for DNN accelerators through analytical modeling. In *Proceedings of the 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 570–583.

[31] Linyan Mei, Pouya Houshmand, Vikram Jain, Sebastian Giraldo, and Marian Verhelst. 2021. ZigZag: Enlarging joint architecture-mapping design space exploration for DNN accelerators. *IEEE Transactions on Computers* 70, 8 (2021), 1160–1174.

[32] Inc. Micron Technology. 2017. TN-40-07: Calculating Memory Power for DDR4 SDRAM. Retrieved from https://www.mouser.com/pdfDocs/tn4007_ddr4_power_calculation.pdf

[33] Pierpaolo Morì, Manoj-Rohit Vemparala, Nael Fasfous, Saptarshi Mitra, Sreetama Sarkar, Alexander Frickenstein, Lukas Frickenstein, Domenik Helms, Naveen Shankar Nagaraja, Walter Stechele, et al. 2022. Accelerating and pruning CNNs for semantic segmentation on FPGA. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 145–150.

[34] Mohammad Motamedi, Philipp Gysel, Venkatesh Akella, and Soheil Ghiasi. 2016. Design space exploration of FPGA-based deep convolutional neural networks. In *Proceedings of the 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 575–580.

[35] Duy Thanh Nguyen, Hyun Kim, and Hyuk-Jae Lee. 2020. 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* 31, 6 (2020), 2450–2464.

[36] NVIDIA. 2017. NVDLA: NVIDIA Deep Learning Accelerator. Retrieved from https://nvdla.org

[37] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel Emer. 2019. Timeloop: A systematic approach to DNN accelerator evaluation. In *Proceedings of the 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 304–315.

[38] Fareed Qararyah, Muhammad Waqar Azhar, and Pedro Trancoso. 2024. An efficient hybrid deep learning accelerator for compact and heterogeneous CNNs. *ACM Transactions on Architecture and Code Optimization* 21, 2 (2024), 1–26.

[39] Fareed Qararyah, Mohammad Ali Maleki, and Pedro Trancoso. 2025. An analytical cost model for fast evaluation of multiple compute-engine CNN accelerators. In *Proceedings of the 2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 1–13. DOI: 10.1109/ISPASS64960.2025.00030

[40] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *ArXiv*. arXiv:1703.03864. Retrieved from https://arxiv.org/abs/1703.03864

[41] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. Scale-sim: Systolic CNN accelerator simulator. *ArXiv*. arXiv:1811.02883. Retrieved from https://arxiv.org/abs/1811.02883

[42] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4510–4520.

[43] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. 2019. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 14–27.

[44] Yongming Shen, Michael Ferdman, and Peter Milder. 2017. Maximizing CNN accelerator efficiency through resource partitioning. In *Proceedings of the 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 535–547.

[45] Arne Symons, Linyan Mei, Steven Colleman, Pouya Houshmand, Sebastian Karl, and Marian Verhelst. 2023. Stream: A modeling framework for fine-grained layer fusion on multi-core DNN accelerators. In *Proceedings of the 2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 355–357.

[46] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. 2019. MnasNet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2820–2828.

[47] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the International Conference on Machine Learning*. PMLR, 6105–6114.

[48] Kevin Trebing, Tomasz Stanczyk, and Siamak Mehrkanoon. 2021. SmaAt-UNet: Precipitation nowcasting using a small attention-UNet architecture. *Pattern Recognition Letters* 145, 2021 (2021), 178–186.

[49] Nicolas Vasilache, Oleksandr Zinenko, Theodoros Theodoridis, Priya Goyal, Zachary DeVito, William S. Moses, Sven Verdoolaege, Andrew Adams, and Albert Cohen. 2018. Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions. *ArXiv*. arXiv:1802.04730. Retrieved from https://arxiv.org/abs/1802.04730

[50] Stylianos I. Venieris and Christos-Savvas Bouganis. 2018. fpgaConvNet: Mapping regular and irregular convolutional neural networks on FPGAs. *IEEE Transactions on Neural Networks and Learning Systems* 30, 2 (2018), 326–342.

[51] Swagath Venkataramani, Ashish Ranjan, Subarno Banerjee, Dipankar Das, Sasikanth Avancha, Ashok Jagannathan, Ajaya Durg, Dheemanth Nagaraj, Bharat Kaul, Pradeep Dubey, et al. 2017. Scaledeep: A scalable compute architecture for learning and evaluating deep networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 13–26.

[52] Rangharajan Venkatesan, Yakun Sophia Shao, Miaorong Wang, Jason Clemons, Steve Dai, Matthew Fojtik, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. 2019. Magnet: A modular accelerator generator for neural networks. In *Proceedings of the 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[53] Marian Verhelst, Man Shi, and Linyan Mei. 2022. ML processors are going multi-core: A performance dream or a scheduling nightmare? *IEEE Solid-State Circuits Magazine* 14, 4 (2022), 18–27.

[54] Xuechao Wei, Yun Liang, Xiuhong Li, Cody Hao Yu, Peng Zhang, and Jason Cong. 2018. TGPA: Tile-grained pipeline architecture for low latency CNN inference. In *Proceedings of the 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. ACM, 1–8.

[55] Di Wu, Yu Zhang, Xijie Jia, Lu Tian, Tianping Li, Lingzhi Sui, Dongliang Xie, and Yi Shan. 2019. A high-performance CNN processor based on FPGA for MobileNets. In *Proceedings of the 2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 136–143.

[56] Zheren Xie, Kui Dai, Zhilin Wu, Jinyue Wang, Xin Lu, and Shuanglong Liu. 2024. Design space exploration of CNN accelerators based on GSA algorithm. In *Proceedings of the 2024 9th International Conference on Signal and Image Processing (ICSIP)*. IEEE, 319–323.

[57] Xilinx Inc. [n. d.]. *Xilinx Power Estimator (XPE)*. Retrieved from https://www.amd.com/en/products/adaptive-socs-and-fpgas/technologies/power-efficiency/power-estimator.html

[58] Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, et al. 2020. Interstellar: Using Halide's scheduling language to analyze DNN accelerators. In *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*. 369–383.

[59] Hanchen Ye, Xiaofan Zhang, Zhize Huang, Gengsheng Chen, and Deming Chen. 2020. HybridDNN: A framework for high-performance hybrid DNN accelerator design and implementation. In *Proceedings of the 2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[60] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-programmable Gate Arrays*. 161–170.

[61] Xiaofan Zhang, Junsong Wang, Chao Zhu, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. 2018. DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs. In *Proceedings of the 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[62] Xiaofan Zhang, Hanchen Ye, and Deming Chen. 2021. Being-ahead: Benchmarking and exploring accelerators for hardware-efficient AI deployment. *ArXiv*. arXiv:2104.02251. Retrieved from https://arxiv.org/abs/2104.02251

[63] Xiaofan Zhang, Hanchen Ye, Junsong Wang, Yonghua Lin, Jinjun Xiong, Wen-mei Hwu, and Deming Chen. 2020. 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*. 1–9.

[64] Shixuan Zheng, Xianjue Zhang, Leibo Liu, Shaojun Wei, and Shouyi Yin. 2022. Atomic dataflow based graph-level workload orchestration for scalable DNN accelerators. In *Proceedings of the 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 475–489.

Received 29 May 2025; revised 15 September 2025; accepted 30 October 2025