AutoAI2C: An Automated Hardware Generator for DNN Acceleration on Both FPGA and ASIC

Yongan Zhang, Xiaofan Zhang, *Member, IEEE*, Pengfei Xu, Yang Zhao, Cong Hao, *Associate Member, IEEE*, Deming Chen, *Fellow, IEEE*, and Yingyan Lin, *Member, IEEE*

***Abstract*—Recent advancements in deep neural networks (DNNs) and the slowing of Moore's law have made domain-specific hardware accelerators for DNNs (i.e., DNN chips) a promising means for enabling more extensive DNN applications. However, designing DNN chips is challenging due to: 1) the vast and nonstandardized design space and 2) different DNN models' varying performance preferences regarding hardware micro-architecture and dataflows. Therefore, designing a DNN chip often takes a large team of interdisciplinary experts months to years. To enable flexible and efficient DNN chip design, we propose *AutoAI2C*: a DNN chip generator that can automatically generate both FPGA- and ASIC-based DNN accelerator implementation (i.e., synthesizable hardware and deployment code) with optimized algorithm-to-hardware mapping, given the DNN model specification from mainstream machine learning frameworks (e.g., PyTorch). Specifically, *AutoAI2C* consists of two major components: 1) a *Chip Predictor*, which can efficiently and reliably predict a DNN accelerator's energy, latency, and resource consumption using the proposed graph-based intermediate accelerator representation and 2) a *Chip Builder*, which can generate and optimize DNN accelerator designs by automatically exploring the design space based on targeting metrics and the *Chip Predictor*'s performance feedback. Extensive experiments show that our *Chip Predictor*'s predictions differ by <10% from real-measured ones. Furthermore, *AutoAI2C* generated accelerators can achieve performance comparable to or better than state-of-the-art accelerators, achieving up to a 2.12× throughput improvements or 2.4× latency reduction with the same level of hardware resource usage, or reducing energy consumption by up to 1.6×, when running the same DNN workloads.**

***Index Terms*—AI chips, design automation, genetic algorithms, neural network hardware.**

I. Introduction

RECENT advancements in deep neural networks (DNNs) have led to their widespread adoption in various real-life applications, including self-driving vehicles [1], medical imaging [2], and the recent surge in large language models (LLMs) for natural language understanding and generation [3], [4]. However, the deployment of DNN-based applications often demands not only high accuracy but also exceptional hardware efficiency. This includes requirements for high throughput, low latency, and minimal energy and area consumption. Despite the exceptional task accuracy achieved by recently proposed DNNs, such as LLMs, their inherent complexity necessitates substantial computation and memory resources, posing significant challenges to practical deployment [3], [4].

While traditional computing platforms such as CPUs and GPUs have been instrumental in advancing the field of machine learning, they often fall short in achieving a satisfying hardware efficiency for DNN workloads. These platforms are generally not optimized for the high levels of parallelism and data reuse that DNNs demand, leading to suboptimal performance and energy inefficiency. Furthermore, the instruction overhead and rigid hardware architecture can often lead to substatial amount of hardware idleness. As a result, there has been growing interest in DNN hardware accelerators, which employ specialized hardware micro-architectures and dataflows to leverage the parallelism and data reuse opportunities inherent in DNNs, thus improving the efficiency of DNN inference/training.

However, developing customized DNN accelerators is a nontrivial task. Practitioners within this area are expected to possess a wide range of cross-disciplinary knowledge, spanning from machine learning algorithms to physical chip design. The result is a vast and diverse design space, necessitating a large team of domain experts and a development time of months or even years for a single DNN accelerator. Specifically, designing accelerators for FPGAs or ASICs typically involves: 1) customizing micro-architecture and dataflow design space for target DNN workloads; 2) using RTL programming to implement accelerator prototypes; and 3) iteratively verifying and tuning the design to improve hardware efficiency while maintaining correct functionality. Consequently, a deep understanding of both DNN algorithms and hardware design, as well as significant time and effort, are essential for DNN accelerator development. To address these challenges, recent advancements include the development of high-level synthesis (HLS) design flows [5], [6] and DNN design automation frameworks [7], [8], which expedite the design process by leveraging high-level algorithmic descriptions and utilizing predefined high-quality hardware IPs. However, these tools still require considerable expert input,

Manuscript received 20 September 2023; revised 16 February 2024; accepted 1 April 2024. Date of publication 24 April 2024; date of current version 20 September 2024. This work was supported in part by the NSF RTML Grant under Award 1937592; in part by the NSF CAREER Award under Grant 2345577; and in part by the AMD Center of Excellence at UIUC. The work of Xiaofan Zhang was supported by the Google Ph.D. Fellowship. This article was recommended by Associate Editor L.-C. Wang. *(Corresponding authors: Deming Chen; Yingyan Lin.)*

Yongan Zhang, Cong Hao, and Yingyan Lin are with the Department of Computer Science, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: celine.lin@gatech.edu).

Xiaofan Zhang is with the ML Systems and Cloud AI, Google, Mountain View, CA 94043 USA.

Pengfei Xu is with the Memory Solution Lab, Samsung, San Jose, CA, USA.

Yang Zhao is with the Department of Electrical and Computer Engineering, University of Minnesota, Twin Cities, MN, USA.

Deming Chen is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Champaign, IL 61801 USA (e-mail: dchen@illinois.edu).

Digital Object Identifier 10.1109/TCAD.2024.3393428

as they either depend on domain experts to narrow down the design space using pre-existing architecture standards [7], [8] or only offer limited design exploration and optimization. This constraint has hampered the development of DNN accelerators with competitive performance.

For the above challenges, we propose *AutoAI2C*, a design automation framework that can automatically generate optimized FPGA- and ASIC-based accelerator implementations for DNN models defined in popular machine learning frameworks, such as PyTorch [9], and enables quick and reliable performance estimation for hardware accelerators ranging from customized accelerators [10], [11], [12] to commercial edge-devices [13], [14]. The main contributions of this article are as follows.

1) *AutoAI2C:* By incorporating the two proposed enablers, i.e., *Chip Predictor* and *Chip Builder*, we construct an automated DNN accelerator design framework dubbed *AutoAI2C*, which can generate the optimized DNN accelerator implementation in an end-to-end manner (i.e., synthesizable hardware and deployment code), by directly taking the inputs of DNN model definitions from popular machine learning frameworks (e.g., PyTorch [9]) and application-driven specifications (e.g., performance objectives and hardware resource budgets). Experiments conducted under various settings show that *AutoAI2C* generated FPGA- and ASIC-based DNN accelerators outperform state-of-the-art (SOTA) accelerators with up to a 2.12× throughput improvements or 2.4× latency reduction with the same level of hardware resource usage, or reducing energy consumption by up to 1.6×, when running the same DNN workloads.
2) *One-for-All Design Space Description:* We introduce an object-oriented graph-based representation for DNN accelerators that unifies design factors across three abstraction levels: IP, architecture, and hardware-mapping. This approach enables highly flexible hardware configurations, scalable architecture and mapping co-optimization, and adaptive accelerator designs that can adapt to different algorithms.
3) *Chip Predictor:* Building on the unified design space description, we propose *Chip Predictor*, a multigrained performance estimation tool for DNN accelerators. This tool offers a coarse-grained and analytical-model-based mode for quick performance estimation, as well as a fine-grained and run-time-simulation-based mode for more accurate estimation, leading to less than a 10% prediction error as compared to real-measured performance.
4) *Chip Builder:* To create DNN accelerator designs with competitive hardware performance, we further propose *Chip Builder*, which optimizes designs within the proposed unified design space using a two-stage design space exploration (DSE) methodology to balance the exploration efficiency and the generated design quality. Specifically, our *Chip Builder* features: 1) an architecture/IP design process that utilizes the quick performance feedback from *Chip Predictor*'s coarse-grained mode, enabling a fast 1st-stage exploration and optimization and 2) an IP/pipeline design process that relies on *Chip Predictor*'s fine-grained mode, for a detailed 2nd-stage IP-pipeline co-optimization and resource reallocation.

This work is a continuation of our *AutoDNNchip* work published in [15]. Specifically, *AutoAI2C* further enhances *AutoDNNchip* [15] by making the following three new contributions. First, we further expand the accelerator search space to include more accelerator templates inspired by the versatile accelerator [16]. This enables *AutoAI2C* to generate more efficient accelerators for recent DNNs with diverse structures, catering to various application requirements. The newly added templates, however, increase the accelerator search space size during DSE from 4.6 million to 40 million, raising the exhaustive search time (8.75×) from 0.8 to 7 h. Thus, a more efficient search algorithm is necessary to ensure *AutoAI2C*'s search efficiency. To this end, second, we propose a new search method based on a bio-inspired evolutionary algorithm (EA) [17] for *Chip Builder* (see Section V) to more efficiently navigate over the large hardware accelerator search spaces, leading to further optimized accelerator designs and thus better accelerator performance. Third, we conduct comprehensive experiments to validate the effectiveness and advantages of *AutoAI2C*, integrating the expanded accelerator design space and the newly proposed search algorithm. Specifically, the automatically generated accelerators from our *AutoAI2C* outperform those generated using existing SOTA design automation frameworks [15], [18], [19], [20] by 3.72× on throughput (Section VI-E). Moreover, compared with commonly used random search or reinforcement learning (RL)-based search algorithms [18], our evolutionary-based search algorithm achieves a 4.12× improvement in search efficiency (Section VI-D).

II. Background and Related Works

DNN Accelerators: In recent years, DNN accelerators have become key for accelerating DNN workloads due to their performance and energy efficiency [8], [10], [14], [21], [22], [23]. Innovations like DNNBuilder [8] enhance FPGA throughput with specialized designs, while new approaches [16] efficiently map DNN models to accelerators by partitioning hardware resources. ASIC accelerators also see advancements with optimizations tailored to specific applications, such as Eyeriss's [10] row-stationary dataflow for convolutional layers. However, the complexity of designing these accelerators, requiring expertise in both DNN algorithms and hardware design and taking extensive time, underscores the value of our AutoAI2C proposal.

DNN Accelerator Performance Prediction: When estimating DNN accelerator performance, roofline models are commonly used, with customized analytical tools [8] also proposed to capture the various attributes of each accelerator. For more reliable estimation, loop-based descriptions have been adopted by tools, such as Timeloop [24] and DNN-Chip Predictor [25], which consider hardware design configurations, memory hierarchies, and dataflows. These tools analyze data

movements and memory accesses for energy and latency estimation. Interstellar [26] extends loop-based descriptions by using Halide's scheduling language to represent hardware designs for design exploration. MAESTRO [27] uses a data-centric approach to depict the accelerator's data movement and reuse behavior. However, these estimation tools often fail to flexibly adjust for the different tradeoffs between estimation efficiency and accuracy as does our proposed multigrained *Chip Predictor*, which enables an effective and efficient DSE.

DNN Accelerator Generation: To bridge the gap between fast DNN model development and slow implementation of hardware accelerators, various accelerator generation tools have been proposed. MAGNet [7] proposes and tunes a configurable accelerator template with HLS tools. ConfuciuX [18] leverages RL to optimize the accelerator design. DNNBuilder [8] take the high-level DNN description in the Caffe/TensorFlow framework and then automatically generate optimized FPGA-based accelerators. More recent works [19], [20], [28], [29] utilize differentiable co-search methods to automate the generation of both DNN model structures and paired DNN accelerator design to achieve an improved tradeoff between task accuracy and hardware efficiency. Previous works, however, either necessitate substantial expert input to narrow down the design space based on established architectural standards [7], [8], or they fail to offer a design space description that can be readily adapted to various hardware architecture styles [19], [20], [28], [29]. Our proposed *AutoAI2C* utilizes an one-for-all graph-based design space description and a two-stage DSE methodology. These two combined approaches can not only easily adapt to a wide variety of hardware architecture styles, but can also automatically narrow down the expansive design space.

III. One-for-All Design Space Description

A. Overview

The design space size of DNN accelerators can be both large and diverse, and consist of various design components. For instance, the number of choices can explode to over 10^{10} and span areas from the network-on-chip (NoC) design style to buffer sizes [20], [29]. To ensure the proposed *AutoAI2C* framework's generality and, at the same time, minimize the human effort in the design automation loop, it is crucial to create a design space that is both clearly formulated and comprehensively applicable to all levels of the design. Thus, we first summarize the DNN accelerator's design abstraction levels into the following three types: 1) architecture level; 2) IP level; and 3) hardware-mapping level. Table I lists the major design parameters which we deem to be sufficient for most cases, with the last column showing the design abstraction level that each design parameter resides in. Therefore, a capable design space should cover all the aforementioned levels and consistently maintain the same representation format for ease of automation. To this end, we adopt a *One-for-all Design Space Description* that utilizes a directed graph structure to unify the design factors from the three abstraction levels. Upon analysis of Table I, we can observe that: 1) most of the design factors are linked to design optimization

TABLE I
Summary of DNN Accelerators' Design Factors

Design factor	Description	Back-end	Opt. level
B_W, B_A, B_{Acc} [a]	Bit precision	F, A [b]	IP, Accuracy req.
$Freq.$	Clock frequency	F, A	Arch., IP
$Arch_{mem}$	Memory tech/hierarchy/volume	A	Arch., IP
$Arch_{pe}$	PE array architecture	F, A	Arch.
Bw	Port/Bus width for data transfer	A	IP
$Malloc$	Memory allocation	F, A	Arch., IP, Mapping
$Data\ Schedule$	DNN to accelerator mapping	F, A	Mapping

[a] B_W, B_A, B_{Acc}: Bit precision for weights, activations, accumulations
[b] A: ASIC design; F: FPGA design

TABLE II
Summary of Hardware IP Types and Applicable Attributes in the Graph-Based Design Space Description

Compo.[a]	Hardware meaning	Attributes
Node	Memory IPs	Impl., Freq., Vol., Prec., Dt., StM., E, L[b]
	Computation IPs	Impl., Freq., Prec., StM., E, L
	Data Path IPs	Impl., Freq., Bw.[c] Prec., Dt., StM., E, L
Edge	IP inter-connections (IP dependency)	Start, End [d]

[a] Compo.: components within the graph denoted as nodes and directed edges;
[b] Impl.: arch. implementation style , e.g., DSP48E, 28nm SRAM, sync FIFO, etc.; Vol.: volume/capacity (bits); Freq.: clock frequency (MHz); Prec.: bit precision; Dt.: data type for model weights and activations.; E/L: energy&latency overhead; StM.: the state machine (including needed inputs and generated outputs) through the whole execution process;
[c] Bw.: port/bus width; [d] Start & End: the start or end node for an edge.

from multiple levels and 2) the variation in design factors covers a wide range, from clock frequency to algorithm-to-hardware mapping. We thus conjecture that the overall system performance is governed by optimization from different hardware components in different design levels. Therefore, for the design space description of the DNN accelerator, we adopt an object-oriented directed graph which enables cross-level optimization for the instantiated hardware components. An illustrative example of the representation format is shown in Fig. 2. Specifically, a basic directed graph is first constructed using the design factors for the PE array architecture, memory architecture and mapping/dataflow. Within the graph, each node denotes a hardware IP which we categorize into three types: 1) computation IP; 2) datapath IP; and 3) memory IP, as listed in Table II. Each directed edge in the graph denotes an interconnection between two nodes and the edge direction is governed by the corresponding data movement's direction, i.e., the data dependency between the nodes. To cover the design possibilities across multiple abstraction levels, the directed graph's nodes and edges are appended with various applicable attributes, as listed in Table II, in an object-oriented fashion. We organize the following sections such that in Section III-B, based on the five SOTA DNN accelerators, we introduce five graph-based accelerator templates, which are further used in the *Hardware IP Pool* (see Fig. 1 under the user specified inputs) of *AutoAI2C*. Together with other customized auxiliary templates, the available IPs can be combined into a defined directed graph to provide a number of design candidates with various performance tradeoffs. In Section III-C, we discuss in detail the applicable attributes for the IP nodes and edges.

B. Graph-Based Accelerator Templates

We construct four graph-based accelerator templates in Fig. 3 to illustrate DNN accelerators. These templates can be compiled into real accelerator implementation by applying

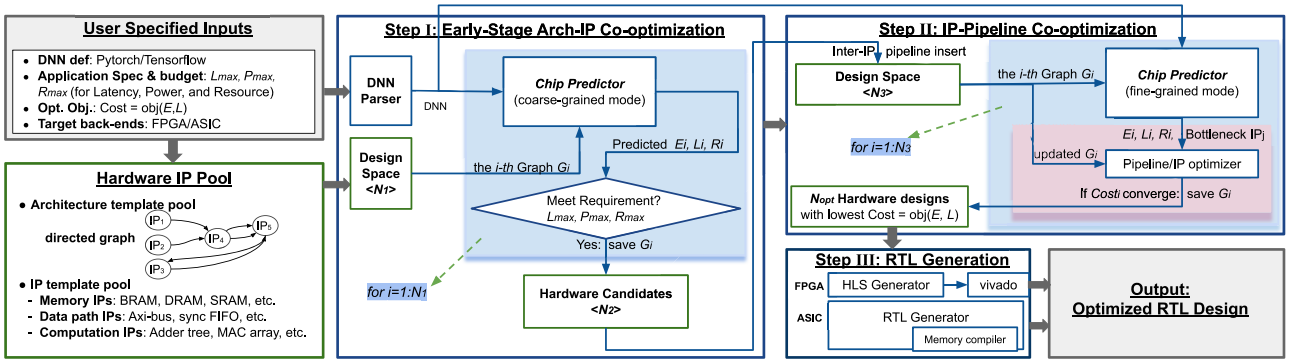


Fig. 1. *AutoAI2C*'s design flow for the DSE, optimization and DNN-to-RTL generation.

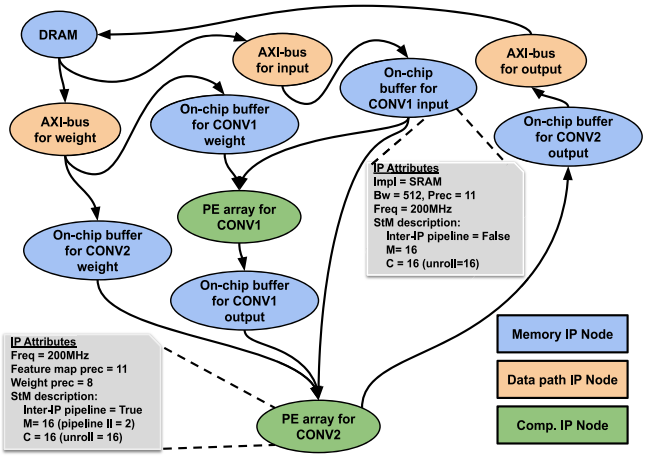


Fig. 2. Illustrating the graph-based design space description based on a heterogeneous accelerator architecture to accelerate the residual block in ResNet [30], with M and C denoting the output and input channels, respectively.

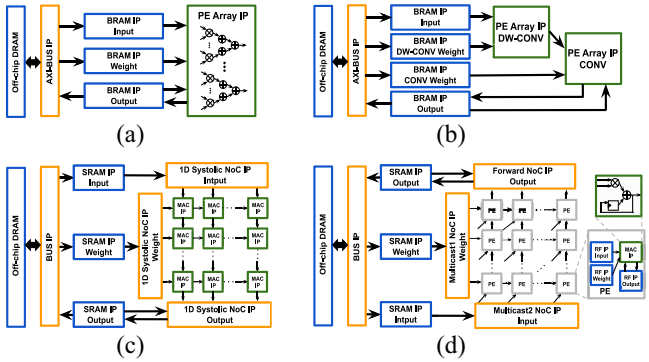


Fig. 3. Overview of the 4 architecture templates in our *Hardware IP Pool*, which are adopted from SOTA FPGA- and ASIC-based DNN accelerators, where (a) illustrates a common architecture template where a sequence of multipliers is followed by an adder-tree reduction network, (b) shows a template comprising PE array IP bundle, with a depth-wise convolutional IP (DW CONV) and a normal convolutional IP (CONV), (c) demonstrates a systolic array style architecture template, and (d) shows an architecture template similar to a well-known DNN accelerator Eyeriss [10].

appropriate IP and edge attributes (Section III-C). Specifically, Fig. 3(a) shows an architecture template where a sequence of multipliers is followed by an adder-tree reduction network to compose the major PE array IP, which is a commonly used implementation in FPGA-based accelerators. Fig. 3(b) shows a template comprising two different PE array IPs in a bundle, including a depth-wise convolutional IP (denoted as DW_CONV) and a normal convolutional IP (denoted as CONV). The DW_CONV and CONV operator combination is commonly adopted in compact vision-based accelerator systems such as [31]. Two dedicated BRAM IPs are included to handle the memory configuration and access behavior for the two computation IPs, respectively. Fig. 3(c) utilizes a systolic array style architecture template which is adopted by TPU [14] type DNN accelerators. Fig. 3(d) demonstrates an architecture template where the data path IPs (i.e., NoC IPs) define the data reuse and propagation of the activation and weights within the PE array, and each PE IP has a more complicated structure in terms of the register file (RF) size and logics to feed the data to each PE. This architecture style is adopted by the well-known DNN accelerator Eyeriss [10].

In addition to the four aforementioned monolithic accelerator templates, we propose the fifth one inspired by [8], [16], and [32]. This new template is designed to more effectively manage the complexities of recently developed DNN models, which feature a diverse set of layers and execution patterns. As highlighted in [32], emerging applications like augmented/virtual reality (AR/VR) and natural language processing (NLP) often require the deployment of multiple DNN models for a single task. Moreover, even within a single DNN model, there can be a heterogeneous mix of operation types and data shapes, such as the combination of multilayer perceptrons (MLPs), convolutional layers, and attention mechanisms in many recent NN models. These varied workloads pose a challenge for existing accelerators that rely on a single, fixed architecture, making it difficult to maintain high hardware resource utilization throughout the execution process. To address this, our newly proposed architecture template employs a parameterized multichunk micro-architecture. This design balances communication bandwidth and maximizes resource utilization among different hardware modules, particularly when running a diverse set of operations. As a result, it significantly enhances the algorithmic throughput of accelerators generated by AutoAI2C, especially for more complex tasks. Specifically, as shown in Fig. 4, each chunk of

the micro-architecture is associated with: 1) one subaccelerator, equipped with multiple memory hierarchies (e.g., on-chip SRAM buffer and off-chip DRAM); 2) a PE array which is parameterized by searchable design factors such as style of PE interconnection (i.e., NoC); 3) allocated capacity for each of the buffers; and 4) the mapping and scheduling strategies for each operation (i.e., dataflows) to effectively leverage data reuse and parallelism. By doing this, each subaccelerator can flexibly process multiple but not necessarily consecutive layers, while different subaccelerators can be pipelined. For more efficient mapping, layers with similar structures tend to be assigned to the same subaccelerator.

While the accelerator template in Fig. 3 features single-accelerator micro-architectures, the one in Fig. 4 adopts a multiaccelerator micro-architecture. As each subaccelerator in the multiaccelerator micro-architecture has its unique design parameters, the design space increases exponentially with the number of subaccelerators. Specifically, each subaccelerator in the multiaccelerator template consists of: 1) secondary on-chip buffer IPs to encourage more local data reuse and reduce expensive off-chip accesses and 2) a PE array, with each PE including a multiply and accumulate (MAC) IP and local RF IPs for the inputs, weights, and outputs; for each subaccelerator, the NoC IPs dedicate the given DNN's temporal and spatial mapping into the PE array and thus the data movement behavior within/across different memories, which can lead to drastically different levels of hardware performance [7]. For example, the size of the design space for VGG16 will increase from 4.6 million to nearly 40 million, when switching from solely considering the original single-accelerator templates in our *AutoDNNChip* [15] to including the new multiaccelerator one. As we will later show, exhaustively exploring the expanded space takes almost 7 h, while the space containing only the single-accelerator templates requires only 0.8 h. As such, the new accelerator template enlarges the graph-based accelerator template pool, and calls for efficient DSE search algorithms to more efficiently navigate through the expanded design space.

C. IP Attributes

This section presents the IP attributes used to characterize each constructed IP in the aforementioned graph-based hardware design space representation. Table II lists the possible attributes for three different node IP types, including memory IPs (e.g., BRAM and off-chip DRAM), data path (e.g., AXI bus), and computation hardware that parameterizes the corresponding designs. Specifically, the attributes are elaborated as follows:

1) The *Implementation or Impl.* attribute represents the necessary hardware resources for each implementation of the IPs' components, e.g., DRAM and BRAM for implementing memory IPs.
2) The *state machine (StM).* attribute describes each IP's transition timestamp and conditions between states of computation and loading/unloading data. The StM attribute also defines both the needed input addresses and generated output addresses for resolving the data dependency. For instance, in Fig. 5, the IPs' StM attribute can be used to construct different pipeline design styles: Fig. 5(b) and (c) illustrate two kinds of designs w/o and w/ inter-IP pipelines along with their corresponding StM definition, respectively. In Fig. 5(c), more states are necessary to represent and implement the inter-IP pipeline between data path and computation IPs.
3) The *data precision or Prec.* attribute represents the bit precision for each data type within the IPs.
4) The *clock Frequency or Freq.* and *energy/latency or E/L* attributes define the achievable clock frequency and required unit energy/latency for the IPs, respectively.
5) The *memory volume or Vol.* and *port/bus width or Bw* attributes represent the memory volume for memory IPs and port/bus width of data path IPs, respectively.

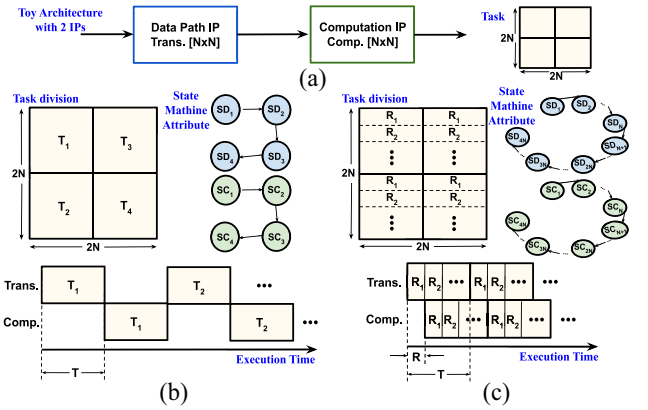


Fig. 5. Toy example demonstrating the generated StM w/o and w/ inserting inter-IP pipeline optimization, where (a) shows a simple architecture with 2 IPs, including a data path IP and a computation IP, and the task division, (b) and (c) illustrate the StM and the time diagram w/o and w/ inserting inter-IP pipelines; SD and SC denote the state for data path IP and computation IP, respectively.

IV. Proposed *Chip Predictor*

A. Overview

To enable automated DSE, as later described in Section V, we develop a reliable and fast accelerator performance estimator called *Chip Predictor*, based on the design space representation proposed in Section III. As shown in Fig. 6, the proposed *Chip Predictor* takes the inputs of DNN model specs (e.g., the layer structure and activation/weight bit precision), hardware architecture specs (e.g., number of PEs and NoC design), algorithm-to-hardware mapping (e.g., the correspondence between IPs and operators), and IP design factors (e.g., unit energy/delay cost of a MAC operation and memory accesses), and then outputs the estimated energy consumption, latency, and resource consumption to execute the specific DNN models on the target accelerator. Under the scope of the whole proposed *AutoAI2C*, *Chip Predictor* plays a crucial role in connecting the framework's different components. Specifically, first, we construct a graph-based description that considers all design abstraction levels and covers a large search space. The space representation is then used as one

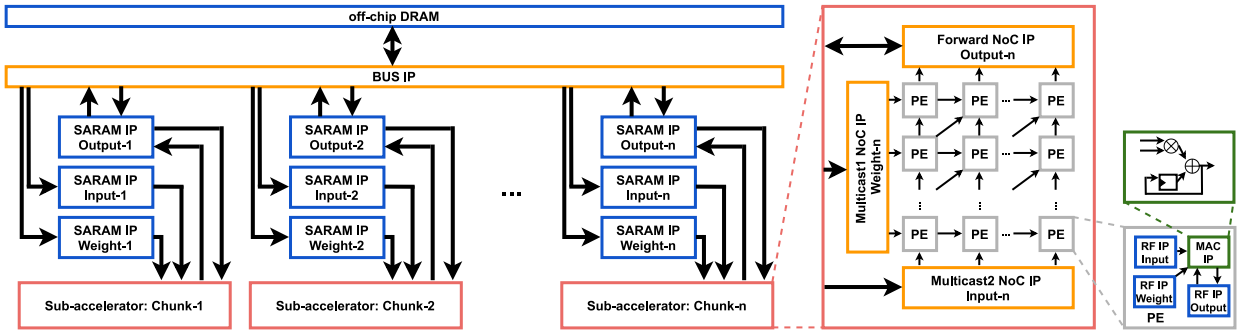


Fig. 4. Illustration of the chunk-based pipeline architecture template, newly incorporated into our AutoAI2C.

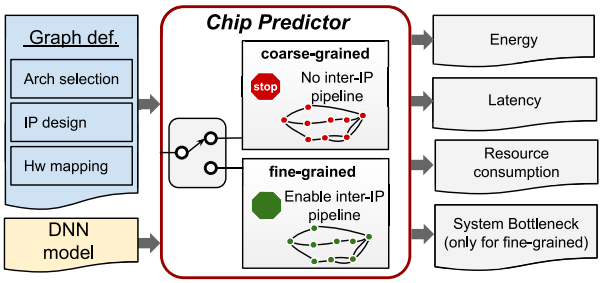


Fig. 6. Overview of the proposed *Chip Predictor*.

of the inputs to *Chip Predictor*. Second, to accommodate the different simulation accuracy and time requirements of *Chip Builder*'s two-stage DSE, *Chip Predictor* adopts a mixed-granularity prediction approach: 1) a coarse-grained mode that provides quick analytical IP performance estimation, enabling efficient design space reduction during *Chip Builder*'s 1st-stage architecture and IP exploration and 2) a fine-grained mode that offers accurate but slightly slower performance prediction through run-time simulations, by modeling inter-IP pipeline dependency between IPs. This mode is used for *Chip Builder*'s 2nd-stage DSE, targeting IP-pipeline co-optimization for reduced idle cycles.

B. Chip Predictor's Coarse-Grained Mode

1) Analytical-Model-Based Intra-IP Modeling: Chip Predictor's coarse-grained mode is based on analytical models, i.e., utilizing a set of formulations for estimation. For symbol consistency, throughout the article, E, L, and R represent the modeled energy, latency, and resource consumption, respectively. ip_{comp}, ip_{dp}, and ip_{mem} denote the computation IP, data path IP, and the memory IP, respectively. The high-level modeling concept utilizes the unit energy, latency, and overhead costs profiled beforehand in simulation and scales the cost accordingly with the design attribute values to obtain the estimated consumption. Specifically, the energy and latency of the computation IPs are formulated by

$$E_{ip_{\text{comp}}} = e_1 + (\#\text{states}) \times (e_2 + e_{\text{mac}} \times U) \tag{1}$$

$$L_{ip_{\text{comp}}} = l_1 + (\#\text{states}) \times l_{\text{mac}} \tag{2}$$

where (#states) denotes the total number of states as defined by the IP StM attributes; U denotes the unrolling factor which dedicates the number of parallel PEs; e_{mac} and l_{mac} denote the unit energy and latency costs for a MAC operation, respectively; e_1 and l_1 are the energy and latency overhead for starting up, i.e., latency on configuring the data path switches and preloading data, respectively; and e_2 denotes energy or logic overhead for the controlling units. All the unit costs are profiled beforehand for plugging in, based on individual modules' RTL implementation and simulations under different clock frequencies, unroll factors, bit precisions, etc. Meanwhile, the energy and latency of the data path IPs can be modeled in a similar fashion

$$E_{ip_{dp}} = e_3 + (\#\text{states}) \times (e_4 + V \times e_{\text{bit}}) \tag{3}$$

$$L_{ip_{dp}} = l_2 + (\#\text{states}) \times \left(l_3 + \frac{V}{P_w} \times l_{\text{bit}}\right) \tag{4}$$

where V denotes the total inter-IP communication data volume (bits); P_w denotes the data path's port width; e_{bit} and l_{bit} denote, for each bit of data access, the unit energy and latency costs, respectively; e_3 and l_2 are the energy and latency overhead for starting up the pipeline, respectively; and e_4 and l_3 denote the CPU and auxiliary logics controlling overhead for the energy and latency, respectively.

2) Analytical-Model-Based Inter-IP Modeling: The system performance, as indicated by the total latency, energy, and resource consumption of a DNN building block (e.g., the bundle in [21] and [33]), is modeled such that the total energy consumption is obtained from summing up the consumption of all the instantiated IPs. The total latency is obtained from summing up all the IPs' latencies on the critical path of the graph. Specifically, it is formulated as

$$R_{\text{mem}} = \sum_{ip_{\text{mem}} \in G} \text{Vol}_{ip_{\text{mem}}} \tag{5}$$

$$R_{\text{mul}} = \sum_{ip_{\text{comp}} \in G} U_{ip_{\text{comp}}} + R_{\text{mul}_{\text{dec}}} \tag{6}$$

$$E = \sum_{ip \in G} E_{ip}; \; L = \max_{\text{path} \in G} \sum_{ip \in \text{path}} L_{ip} \tag{7}$$

where G denotes the whole graph representing the targeted accelerator; R_{mem} denotes the total consumed memory volume for each memory type as defined by the *Impl* attributes; R_{mul} denotes the total number of consumed multipliers used in both the computation IPs and the necessary memory address decoding logic, with the latter specially denoted as $R_{\text{mul}_{\text{dec}}}$.

Regarding the latency estimation, the system latency for the represented accelerator is taken from the maximum of all the data paths' latencies, i.e., the critical path latency. For faster modeling, the inter-IP pipeline effects are excluded in the coarse-grained mode but further included in the fine-grained mode of *Chip Predictor* (see Section IV-C). More cycles can be saved in the fine grained mode thanks to the inserted inter-IP pipelines.

C. Chip Predictor's Fine-Grained Mode

In the fine-grained mode of *Chip Predictor*, we utilize Algorithm 1 to perform the run-time simulations by considering the inter-IP pipeline to obtain the corresponding inter-IP latency. Overall, Algorithm 1 greedily schedules the targeted algorithm's operations on their corresponding hardware IPs at a much finer granularity (as fine as each input feature point). The finer-grained scheduling enables the maximum possible overlap in execution time among the targeted hardware IPs for implementing the inter-IP pipeline.

The detailed run-time simulation algorithm is described in Algorithm 1, where each IP (denoted as *ip*) has: 1) its neighbor IPs on the graph specified as *ip.prev* and *ip.next* according to the operation dependency and algorithm mapping, respectively, and each *ip* receives its input data from *ip.prev* and passes its outputs to *ip.next*; and 2) a StM for each instantiated IP based on the previously defined *StM* attribute to store different states during the whole execution process, including each IP's busy/idle status and its required inputs and produced outputs. The algorithm iteratively checks whether a mapped operation can be scheduled to run and the IP can move to the next state. Specifically, for each clock cycle during the simulation, *ip* can transition to the next state when: 1) all the output data in the current states is produced (i.e., *ip* completes computing and reaches the idle state) and 2) *ip.prev* has produced all of the input data which *ip* requires for the next state. The amount of required input and output data is based on the granularity of the mapped operation. If *ip* is in the idle state but the required inputs are not ready from *ip.prev*, it will continue waiting in the idle state, causing an increase in the idle cycles associated with this IP and blocking the pipeline. An *ip* is designated as busy when generating its outputs and will transition to the idle state when it finishes generating all the outputs in this state.

Algorithm 1 Run-Time Simulation for *Chip Predictor*'s the Fine-Grained Mode

Input: An accelerator design specified by graph G;
For each *edge* in G
  $ip_{start} \longleftarrow edge'$s starting node;
  $ip_{end} \longleftarrow edge'$s ending node;
  Add ip_{start} to $ip_{end}.prev$;
  Add ip_{end} to $ip_{start}.next$;
Initialize energy and latency: $E = 0$, $cycles = 0$;
While not all inference outputs are stored back
  $cycles \longleftarrow cycles + 1$;
  For each ip in G
    If (ip is *idle*) & (all needed inputs $\in$ outputs of $ip.prev$)
      $ip \longleftarrow busy$;
      ip jumps to the next state;
    If (ip is *idle*) & (not all needed inputs $\in$ outputs of $ip.prev$)
      $ip.idle\_cycles \longleftarrow ip.idle\_cycles + 1$;
    If (ip is *busy*) & (not all outputs for ip is ready)
      Update the ready outputs for ip;
    If (ip is *busy*) & (all outputs for ip is ready)
      $ip \longleftarrow idle$;
      $E \longleftarrow E + E_{ip}$;
$L \longleftarrow \frac{cycles}{global\ clk\ freq}$;
$ip_{bottleneck} \longleftarrow ip$ with minimum idle cycles.

Algorithm 2 IP-Pipeline Co-Optimization Stage for the *Chip Builder*

Input: Design space D_G containing N_2 graphs;
For each G in D_G
  For each *edge* in G
    $ip_{start} \longleftarrow edge'$s starting node;
    $ip_{end} \longleftarrow edge'$s ending node;
    Add ip_{start} to $ip_{end}.prev$;
    Add ip_{end} to $ip_{start}.next$;
  While simulated latency L_G does not converge (Algorithm 1)
    $ip \longleftarrow ip_{bottleneck}$ from Algorithm 1;
    If inter-IP pipeline is inserted between ip and $ip.next$
      allocate more resource to ip;
    Else
      insert inter-IP pipeline between ip and $ip.next$;
      update the state machine of ip;
      update the state machine of $ip.next$;
Select top N_{opt} candidates in D_G

V. Proposed *Chip Builder*

A. Overview

To automate accelerator design in complex and practical implementations, we introduce *Chip Builder*, which features a two-stage DSE engine (Section V-B). This engine can: 1) efficiently rule out infeasible designs via the 1st-stage optimization, which focuses on architecture/IP designs using *Chip Predictor*'s coarse-grained mode and 2) effectively boost the performance of the remaining designs (e.g., those left from the previous stage) via the 2nd-stage IP-pipeline co-optimization using *Chip Predictor*'s fine-grained mode. To improve the DSE efficiency, which could otherwise render the DSE infeasible for certain complicated accelerator designs, we also introduce a tailored EA for the fast exploration stage, enabling more efficient scanning of the design space (Section V-C).

B. Two Stage DSE

As illustrated in Fig. 1, the design optimization flow of *AutoAI2C* utilizes *Chip Builder*'s two-stage DSE engine based on the feedback from the proposed *Chip Predictor*. To effectively and efficiently explore the design space, summarized as design factors in Table I, *AutoAI2C* comprises three major procedures as outlined in Fig. 1.

1) *First-Stage DSE:* A fast exploration process for the micro-architecture and dataflow space to efficiently prune away infeasible designs with the help of *Chip Predictor*'s coarse-grained mode.
2) *Second-Stage DSE:* An IP resource optimization and inter-IP pipeline exploration process to effectively improve the hardware performance of the top performing design candidates generated from the 1st-stage DSE and to more accurately pick out the best designs.

3) A system-level design validation using automated RTL generation and execution.

1) Step I (Early Stage Architecture and IP Configuration Exploration): As shown in the middle section of Fig. 1, the fast exploration stage primarily focuses on parsing the input algorithm specs, matching the algorithm with suitable hardware templates and optimizing the hardware design factors. Specifically, first, a DNN parser is implemented to extract the DNN layer information (e.g., layer types such as CONV, ReLU and Pooling), given the definition from a mainstream machine learning framework (e.g., PyTorch [9]). Second, based on the parsed DNN model definition, the user-defined performance metric (e.g., latency) and hardware budgets (e.g., max available BRAM size of the FPGAs), an accelerator design space containing N_1 potential candidates is generated by producing commonly-used and applicable hardware architecture templates and hardware IP templates from the *Hardware IP pool*. For example, a folded and more reused hardware architecture is favored over a flattened one when the resource budget is limited, while flattened structures that facilitate IP pipelines and have shorter initiation intervals are preferred when the budget is sufficient and throughput is the primary performance goal. Third, an exhaustive search is conducted on the possible design choices for the architecture and IP configuration to eliminate most infeasible designs and narrow down the design space to N_2 ($N_2 < N_1$) promising candidates, such as designs with lower latency or higher throughput based on specific design requirements. This quick early exploration leverages the analytical nature of the *Chip Predictor*'s coarse-grained mode, allowing for efficient navigation through the vast space. However, in situations where the accelerator architecture is more complex and involves numerous design choices, this exhaustive approach to reducing the design space may be restrictive. As a result, in Section V-C, we introduce a more efficient evolutionary search algorithm to enhance the speed of rapid exploration.

2) Step II (Inter-IP Pipeline Exploration and IP Optimization): To enhance performance, this step takes the N_2 designs obtained from the fast exploration stage as inputs and carries out further IP optimization using Algorithm 2. In actual implementation, N_2 is empirically tuned such that the top N_2 designs from the 1st-stage DSE stand out the most from the rest of the designs, i.e., the centroid of the top N_2 designs' performance has the maximum distance to the centroid of the rest of the designs' performance. The 2nd-stage DSE primarily involves inserting inter-IP pipelines for more precise estimation, identifying bottlenecks, reallocating resources, and exporting the best designs. Specifically, first, a new design space of size N_3 (automatically inferred from previous N_2 designs) is created by inserting inter-IP pipelines into different positions within the associated graph-based space representation. This leads to N_3 new graphs with varying inter-IP pipeline designs and potentially different hardware performance. Second, for each of the graphs with inter-IP pipelines inserted, bottleneck IPs are identified during Algorithm 1's run-time simulations and then optimized via deeper (if not already inserted) inter-IP pipeline designs or provided with more resources via reallocation, until the performance converges based on the estimation from *Chip Predictor*'s fine-grained mode (The performance optimization process can also terminate based on the designated total number of optimization iterations), as shown in Algorithm 2. Third, after the final optimization iteration, the top N_{opt} design candidates are selected based on the *Chip Predictor*'s estimated hardware performance. RTL validation is then performed on these candidates to ensure their practicality for implementation.

Although the designs before the 1st-stage DSE can also benefit from the above *Inter-IP Pipeline Exploration and IP Optimization*, the performance gap between the top N_2 designs and the rest of the designs is usually too large for the rest of the designs to catch up. Therefore, we only focus on the top N_2 designs in this step for better efficiency.

3) Step III (Design Validation Through RTL Generation and Execution): Upon obtaining the top N_{opt} optimized designs from the two-stage DSE, an automated code generation procedure is executed to generate the RTL or HLS C source code for FPGA and ASIC back-ends, respectively.

1) For the FPGA back-end, we use the standard Vivado [5] design flow to generate the bitstream with the predesigned system diagram templates. Concurrently, designs that fail the place and route (PnR) process are removed to ensure the functional validity of the *AutoAI2C*'s generated accelerators. The final generated design files include the board-level deployment C code, the quantized and reordered weights' binaries, and the HLS C code for the HLS IP back-end.
2) For the ASIC back-end, the generated design files include the synthesizable RTL code and testbench, the quantized and reordered weight binaries, and the memory specifications. To generate the gate-level or layout netlist, the output RTL can be passed to an EDA tool like Design Compiler. Simultaneously, Memory Compilers use the on-chip memory specifications to generate the memory design. As a result, the correct functionalities of all output designs are ensured after completing this process.

C. Evolutionary Search Algorithm as Efficient Plugin

In this section, we present the evolutionary search algorithm that serves as a plugin for the vanilla *Chip Builder*'s fast exploration stage. The vanilla *Chip Builder* denotes the *Chip Builder* variant that uses exhaustive search to conduct DSE. The motivation is that the accelerator design space can easily explode with the more diverse accelerator templates that are required by the most recent DNN models, which have increased complexity and structure diversity, making the exhaustive search algorithm adopted in the vanilla *Chip Builder* less efficient or even infeasible. Therefore, our objective is to equip the vanilla *Chip Builder* with a more efficient evolutionary search algorithm capable of handling a much larger design space, ultimately leading to superior accelerators with enhanced hardware performance.

Evolutionary search algorithms surpass conventional random search in efficiency by strategically exploring promising

Algorithm 3 Proposed Evolutionary Search Algorithm as a Plugin for *Chip Builder*'s Fast Exploration Stage

Inputs: Termination condition T; the maximum population $p_{\max}$; perturbation rate r; birth/dying rate s; the number of output genomes N_2;
genome_pool ={}
While T not met
  If size of *genome_pool* $\leq p_{\max}$
    If *genome_pool* is empty
      Generate $s * p_{\max}$ new genomes by randomly picking genes
      Add new genomes to *genome_pool*
    Else
      Generate new genomes by randomly change r of the genes from the top $s * p_{\max}$ performing genomes in *genome_pool*
      Add the new genomes to *genome_pool*
  Else
    Remove the bottom $s * p_{\max}$ genomes from *genome_pool*
Return N_2 top performing genomes from *genome_pool*

areas and using information from previous iterations to guide the search toward optimal solutions [34]. The flexibility and reduced sensitivity to hyperparameters as compared with algorithms like Bayesian Optimization also make EAs particularly suitable for optimizing new hardware architectures or algorithmic applications.

As detailed in Algorithm 3, our proposed evolutionary search algorithm enables efficient exploration by maintaining constant perturbation around favorable design points and filtering out undesired ones. Following the convention in [17], during each search iteration, the candidate accelerators under evaluation can be regarded as genomes. The properties of these genomes are characterized by genes, which are designed to denote connections among different hardware IPs and IPs' attributes within a parameterized graph-based accelerator design description, as introduced in Section III-B. During implementation, each genome is represented as a vector with each element as a gene. In Algorithm 3, T represents the termination condition of the search process and can be designed to either achieve the target accelerator performance (e.g., energy or latency) or complete a specific number of search iterations. The maximum population, $p_{\max}$, indicates the highest number of accelerator choices considered during each search iteration. The perturbation rate, r, represents the fraction of genes modified based on the top-ranking designs to generate new designs or genomes. The birth or dying rate, s, denotes the fraction of newly created designs or removed designs due to poor performance. Finally, N_2 refers to the number of the ultimately selected top-performing designs, as described in Section V-B.

With the proposed EA-based accelerator search algorithm, our *Chip Builder*, and thus *AutoAI2C*, can more efficiently explore the large design spaces and handle more diverse accelerator template pools, promising to generate accelerators with more competitive performance, as validated in our experiments (see Section VI-E).

VI. Experiment Results

A. Methodology and Setup

Hardware Setup: We evaluate AutoAI2C's effectiveness using 20 DNN models across six platforms, including three edge devices [12], [13], [14], one cloud-level FPGA [36], and two ASIC-based accelerators [10], [11]. Table III summarizes the hardware settings for different platforms, including resource constraints and application-driven specifications, which are tailored for real-time visual recognition tasks like image classification and object detection. For the high-performance ZC706 FPGA board [36], we use frames per second (FPS) as the performance metric to align with cloud computing needs. Table I outlines the architecture and dataflow search space for our proposed accelerator DSE. Throughout the experiments, we validate the top 10 designs after the two-stage DSE, i.e., N_{opt} is set to 10.

TABLE III
Application-Driven Inputs, Such as the Objectives (e.g., Latency) and Hardware Constraints (e.g., Resource Budget), When *Chip Builder*'s Generated FPGA- and ASIC-Based DNN Accelerators Are Evaluated

Target Back-end	Application	Opt. Obj.	Th./P. Req.	Res. Budget
Ultra96 FPGA	Object Detection	E, L	20FPS 10W	DSP=360, FF=141120 LUT=70560, BRAM=432
ZC706 FPGA	Image Classification	FPS	/ 15W	DSP=900, FF=437200 LUT=218600, BRAM=1090
ASIC	Vision Tasks [11]	E, L	15FPS 600mW	On-chip SRAM=128KB # of MAC units=64

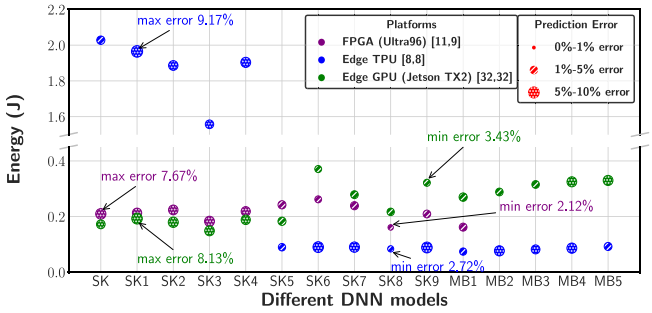


Fig. 7. *Chip Predictor*'s prediction error for the energy consumption targeting 15 DNN model variants from SkyNet(SK) [21] and MobileNetV2(MB) [35] on 3 edge devices: an edge FPGA, Edge TPU, and edge GPU.

Benchmarked DNN Models and Datasets: To fairly compare with two academic ASIC accelerators [10], [11], we use the DNN models specified in their original papers. For the three edge devices, we evaluate 15 representative compact DNN model variants from MobileNetV2 [35] and SK [21], with different model settings such as channel scaling factors and input resolutions. To assess AutoAI2C's scalability, we also benchmark additional models of varying complexity [37], [38], as illustrated in Fig. 10.

Chip Predictor Setup: The unit energy and latency factors are extracted either through real-device measurement, synthesized simulation, or paper-reported data [10].

B. Validation of Chip Predictor

In this section, we evaluate the prediction accuracy of the proposed *Chip Predictor*, where the prediction errors, as we later show, are all under 10%, enabling the predictor to reliably guide the proposed *Chip Builder* during the DSE.

1) Validation of the Predicted Energy Consumption: As shown in Fig. 7, as compared with the real-measured ones from 3 edge devices, under the same settings, *Chip Predictor*'s maximum prediction error is under 9.17%, where the major

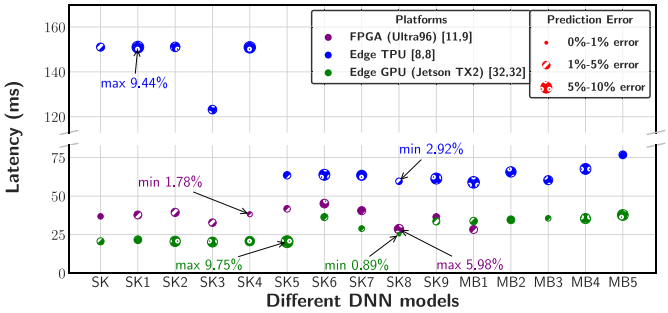


Fig. 8. *Chip Predictor*'s prediction error on the latency consumption targeting 15 DNN model variants from SK [21] and MobileNetV2(MB) [35] on 3 edge devices: an edge FPGA, Edge TPU, and edge GPU.

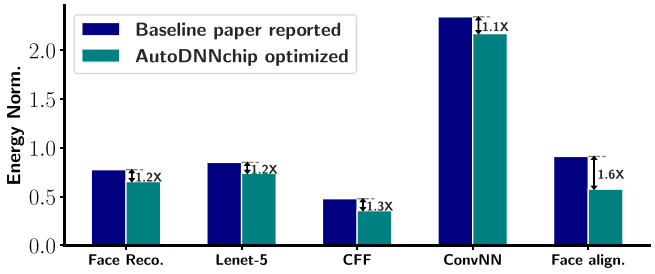


Fig. 9. Benchmarking the normalized energy consumption of ASIC-based accelerators from *AutoAI2C* and the ones in [11], on five shallow neural networks under the same configurations and throughput requirement.

source of error comes from the not modeled power cost of peripheral modules, e.g., board IOs, and routine processes of the operating systems running on the boards.

2) Validation of the Predicted Latency: Benchmarked against measured results of the 15 DNN models on the 3 edge devices, *Chip Predictor*'s latency prediction results are shown in Fig. 8. The average prediction error is 4.85%, 3.73%, and 6.57% on the edge GPU, the Ultra96 FPGA board, and the edge TPU, respectively, where the maximum latency prediction error is 9.75%.

C. Evaluation of Our New Chip Builder and AutoAI2C

In this section, we evaluate the performance of the proposed *Chip Builder* and *AutoAI2C* as a whole, making use of our time-efficient and accurate *Chip Predictor* and the EA search algorithm to perform an effective two-stage DSE. Specifically, we analyze and benchmark the hardware performance of *AutoAI2C* generated DNN accelerators against that of the baselines by comparing the hardware performance of *AutoAI2C* generated accelerators against that of SOTA designs under the same conditions.

1) Evaluating AutoAI2C's Generated FPGA-Based Accelerators: We benchmark *AutoAI2C*-generated accelerators with four SOTA FPGA DNN accelerators [8], [39], [40], [41] in terms of the throughput (i.e., FPS). As shown in Table IV, accelerators generated by the enhanced *AutoAI2C* consistently outperform those of SOTA baselines. In particular, on ImageNet, *AutoAI2C* generated accelerators achieve a 1.11× to 2.12× FPS improvement on VGG16 and 1.10× to 1.73× on AlexNet. When optimizing the accelerator designs for the throughput objective, the pipelined template illustrated in Fig. 4 is eventually picked due to its higher hardware utilization on DNN tasks with decent depth and drastically different layer shapes. In other cases, single accelerator templates, as shown in Fig. 3, tend to be favored for their lower latency and resource consumption. The consistently better performance of *AutoAI2C* generated accelerators validates its effectiveness in automatically navigating over the large accelerator space to generate optimized accelerators for a given DNN model. Note that when using *AutoAI2C* to generate optimized accelerators, for a fair comparison, we adopt the same precision and FPGA resource budgets as the baselines.

2) Evaluating AutoAI2C's Generated ASIC-Based Accelerators: In these experiments, we evaluate the optimized performance tradeoff between latency and energy consumption for *AutoAI2C* generated ASIC-based accelerators. Through benchmarking the energy consumption against the SOTA ASIC-based accelerator [11] on five shallow neural networks, we demonstrate that the proposed *AutoAI2C* can generate ASIC-based accelerators with comparable or even superior energy efficiency under the same throughput constraint. Specifically, as shown in Fig. 9, *AutoAI2C* generated ASIC-based accelerators consistently outperform [11] in all five networks, with a 7.9% to 58.3% energy consumption reduction, highlighting the SOTA acceleration efficiency of *AutoAI2C* generated accelerators.

D. Evaluating AutoAI2C's EA-Based Search Algorithm

In this section, we evaluate the effectiveness of the enhanced *AutoAI2C*'s EA-based search algorithm by benchmarking its search efficiency over that of both random search and RL-based search baselines, which are utilized in existing design automation frameworks such as [18]. Specifically, we optimize the accelerator parameters for VGG16 and AlexNet models using the same hardware settings as in Table IV, based on the random search, RL-based, and EA-based search algorithms. For the RL-based search, the number of layers and hidden units in the RNN are tuned to maximize the search efficiency, where we adopt a 2-layer RNN with 1024 and 512 hidden units for the VGG16 and AlexNet cases, respectively. Furthermore, we empirically tune the hyper-parameters for our EA-based search and find that the searched accelerators' performance is not sensitive to the adopted hyper-parameter values. In all experiments, the hyper-parameters are as follows: the maximum population $p_{\max}$ is set to 1E+4, the perturbation rate r is set to 0.25, the birth/dying rate s is set to 0.2, and the termination condition T is defined as the point at which an accelerator design is identified that can achieve performance metrics of 300 FPS for AlexNet and 30 FPS for VGG16, which are competitive with the SOTA FPGA accelerators as shown in Table IV. Search efficiency is quantified by the number of sampled design points to discover an accelerator design that meets these performance criteria. For consistent comparison, the number of sampled design points are averaged over 50 runs for each search algorithm. Table V shows that the EA-based search consistently achieves the best search

TABLE IV
AutoAI2C GENERATED FPGA ACCELERATORS VERSUS SOTA FPGA ACCELERATORS BASED ON A ZC706 FPGA BOARD [36], WHEN ACCELERATING VGG16 AND ALEXNET ON IMAGENET AT A FREQUENCY OF 200MHZ

Techniques	[8]	[39]	[40]	***AutoAI2C***	[8]	[41]	***AutoAI2C***
DNN Models	VGG16	VGG16	VGG16	VGG16	AlexNet	AlexNet	AlexNet
Resource Utilization	680/900 DSP	824/900 DSP	780/900 DSP	723/900 DSP	808/900 DSP	693/900 DSP	704/900 DSP
Data Precision	16-bit	16-bit	16-bit	16-bit	8-bit	8-bit	8-bit
Throughput (FPS)	27.72	24.34	14.50	**30.79** (↑**1.11**× - ↑**2.12**×)	340	217	**375.17** (↑**1.10**×- ↑**1.73**×)

TABLE V
EVALUATING THE PROPOSED EA-BASED SEARCH ALGORITHM AGAINST RANDOM AND RL-BASED SEARCH USING THE ACCELERATOR DESIGN SPACE FOR ALEXNET AND VGG16. THE DESIGN SPACE SIZE IS 4E+7. EFFICIENCY METRIC IS THE AVERAGE NUMBER OF SAMPLED DESIGN POINTS OUT OF 50 RUNS TO FIND AN ACCELERATOR DESIGN THAT CAN ACHIEVE 300 FPS FOR ALEXNET AND 30 FPS FOR VGG16

	Random Search	RL	EA (**Proposed**)
Sampled Points (#) AlexNet	1838073	1602551	**497701** (↓3.22× - ↓3.69×)
Sampled Points (#) VGG16	2067556	1763490	**501283** (↓3.52× - ↓4.12×)

efficiency across the tested tasks. Specifically, it improves the search efficiency by 2.62× and 3.25× on AlexNet and 2.19× and 2.85× on VGG16, as compared to the RL-based and random search baselines, respectively. Apart from the difference in the number of sampled design points, RL requires a costly training and inference computation for the RNN, leading to a much longer search time than EA and random search.

E. Evaluating AutoAI2C Against Existing Design Automation Frameworks

In this section, we compare the performance of accelerators created using our *AutoAI2C* framework with four SOTA DNN accelerator design automation tools: 1) *AutoDNNchip* [15]; 2) *EDD* [19]; 3) *DIAN* [20]; and 4) *ConfuciuX* [18]. The assessment of the baselines is achieved by analyzing and reproducing this article presented hardware architecture templates, search spaces, and DSE methods, which are essential enablers for the effectiveness of DSE and the resulting accelerator performance. Each framework's performance is benchmarked using the same set of DNN structures (shown in Fig. 10) and dataset (ImageNet), with data precision set to 16-bit on the ZC706 hardware platform. Comparisons are made under hard constraints of the total number of design points sampled by each framework. The constraints are set as 0.125E+7, 0.25E+7, 0.5E+7, 1E+7, and 2E+7, where 2E+7 is the rough number of design points sampled by random search to reach a competitive performance level. For consistency, the performance is averaged over 50 runs for each framework and each constraint.

Based on our comparison, we identify four key observations. First, as shown in Fig. 10, accelerators crafted with our *AutoAI2C* framework consistently outperform those from the four baseline tools, showing an up to 3.72× increase in FPS across different DNN models. The results highlight *AutoAI2C*'s ability to deliver high-performance DNN accelerators. The reasons for this performance edge, which we will detail further, include: 1) a versatile and extensive design space facilitated by a graph-based approach and a broader selection of micro-architecture templates and 2) an efficient two-stage DSE.

Second, our analysis shows that while tools like *EDD* [19] and *DIAN* [20] demonstrate promising initial performance improvements in settings with a limited number of sampled design points, their gains are notably capped. Specifically, *EDD* [19] and *DIAN* [20] use differentiable optimization techniques for hardware configurations, which allow for quick performance boosts in a short optimization period. However, the accelerators designed by these methods often reach a performance ceiling that is relatively low. This limitation appears to stem from the frameworks' tradeoff between design space flexibility and compatibility with differentiable optimization, relying on rigid design space templates that limit potential improvements. In contrast, our method employs a flexible, graph-based design space that enables greater performance potential. Yet, this flexibility complicates the application of baseline differentiable search techniques within our design framework.

Third, our comparison with *AutoDNNchip* [15] and *ConfuciuX* [18] revealed a more significant performance boost in structurally complex networks, e.g., 2.56× for simpler VGG16 [43] as opposed to 3.72× for more complex FBNet-C [37]. This improvement largely stems from our use of a heterogeneous multiaccelerator micro-architecture template (chunk-based pipeline architecture) as shown in Fig. 4, enabling the deployment of specialized subaccelerators for layer clusters with similar structures. By tailoring the hardware design to each layer's optimal computation patterns, e.g., adjusting hardware parallelism to fit the dimensions of larger network weights, we enhance computing resource utilization and achieve higher throughput than the single-accelerator designs of *AutoDNNchip* [15] and *ConfuciuX* [18]. The benefits of aligning computation patterns with hardware design become increasingly apparent as network structures grow more complex, underscoring the value of our multiaccelerator approach.

Fourth, we expanded our analysis by applying the DSE methods of *AutoDNNchip* [15] and *ConfuciuX* [18] within our design space to evaluate their optimization efficiency. We found that despite our design space's greater flexibility and breadth, these frameworks were less effective at leveraging and navigating it, often resulting in suboptimal hardware performance, as shown in Fig. 10. Specifically, *AutoDNNchip* [15]'s exhaustive search and *ConfuciuX* [18]'s RL method struggled to efficiently explore the vast design space for the best configurations. Although the performance difference between our approach and these baselines decreases when the number of sampled design points is limited, our

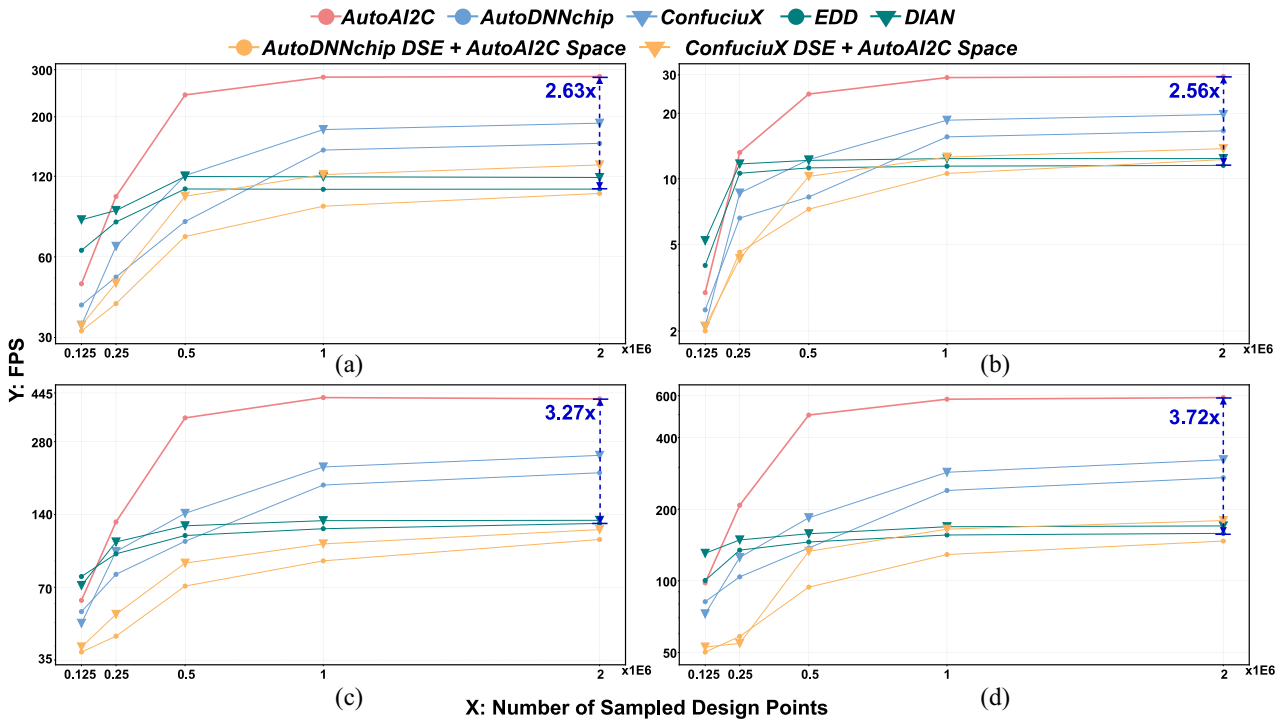


Fig. 10. Performance of generated accelerators by *AutoAI2C* and four baselines [15], [18], [19], [20] when accelerating (a) AlexNet [42], (b) VGG16 [43], (c) EfficientNet-B0 [38], and (d) FBNet-C [37] using different numbers of sampled design points.

AutoAI2C framework more effectively uses an increased number of sampled design points to secure a more substantial performance gain.

VII. Framework Extendability and Scalability

Extendability: Our *AutoAI2C* framewor can be flexibly applied across various application domains beyond its initial focus on DNN acceleration. The core of our approach, a graph-based design space representation coupled with a strategy that integrates multiple heterogeneous accelerators within a single template, is inherently domain-agnostic. This approach is particularly effective for computational tasks that can be decomposed into heterogeneous subtasks. For example, graph processing tasks often involve partitioning a large graph into multiple subgraphs to enhance processing efficiency [44]. Different subgraphs may exhibit unique data sparsity patterns, necessitating specialized accelerators for each to achieve optimal processing efficiency [44]. Our framework accommodates this need by allowing for the deployment of distinct subaccelerators, each dedicated to processing a specific subgraph.

The adaptation of our heterogeneous multiaccelerator architecture to graph processing or other domains indeed requires the design of each subaccelerator to be tailored to the specific application requirements. While the architecture for a DNN accelerator might differ significantly from that needed for graph processing, the versatility of our graph-based design space representation enables this format to be reused across various submodule design levels, so our proposed DSE methods and performance modeling can still be compatible and utilized.

Scalability: With more competitive hardware templates incorporated into the *Hardware IP Pool* as subaccelerators, our proposed *AutoAI2C* can be additionally augmented to ensure both the scalability of hardware performance and the efficiency of DSE. From an architectural standpoint, increasing the number of subaccelerators in a design can meet specific application needs, e.g., distributed processing, but also can introduce potential challenges. Specifically, the communication among subaccelerators could emerge as a performance bottleneck. Integrating more advanced and efficient interconnect IPs can be a strategic priority to solve this issue. Furthermore, with more subaccelerators, interconnect topologies among subaccelerators becomes increasingly critical to achieving optimal performance. Subaccelerator level topology exploration can be added to our DSE process, which is currently empirically fixed for each template. Thanks to our graph-based design space representation, the new exploration can leverage techniques like graph processing to find the optimal topology for enhancing the hardware performance.

The inclusion of additional subaccelerators naturally expands the design space, potentially making it exponentially larger. However, it is important to note that not all subaccelerator designs need to be interdependent. For instance, designs catering to distinct algorithmic operators may be decoupled, allowing us to approach the DSE in a more modular fashion, dividing it into smaller, manageable subspaces. These subspaces can then be explored in parallel for much improved optimization efficiency.

VIII. Conclusion

To bridge the gap between the growing demand for DNN accelerators and the challenging DNN accelerator design process with ever-increasing complexity, we introduce *AutoAI2C*, a framework that automates the generation of

both FPGA- and ASIC-based DNN accelerators. *AutoAI2C* leverages the proposed *One-for-all Design Space Description*, *Chip Predictor*, and *Chip Builder*, enabling a flexible and accessible design process and consistently generating designs with similar or even superior performance as compared with SOTA DNN accelerators in terms of throughput, latency, and energy efficiency. Extensive experiments show that DNN accelerators generated by *AutoAI2C* outperform SOTA designs with a up to 2.12× higher throughput or a 2.4× decrease in latency without increasing hardware resource consumption, or a reduction in energy costs by up to 1.6×.

REFERENCES

[1] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *J. Field Robot.*, vol. 37, no. 3, pp. 362–386, 2020.

[2] A. S. Lundervold and A. Lundervold, "An overview of deep learning in medical imaging focusing on MRI," *Zeitschrift für Medizinische Physik*, vol. 29, no. 2, pp. 102–127, 2019.

[3] "Gpt-4 technical report." openai. 2023. [Online]. Available: https://cdn.openai.com/papers/gpt-4.pdf

[4] Z. Yu et al., "Master-ASR: Achieving multilingual scalability and low-resource adaptation in ASR with modular learning," in *Proc. ICML*, 2023, pp. 40475–40487.

[5] (Xilinx Semicond. Manuf. Commer. Co., San Jose, CA, USA). *Vivado High-Level Synthesis*. Accessed: Apr. 15, 2024. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/high-level-design.html

[6] D. Chen et al., "xPilot: A platform-based behavioral synthesis system," in *Proc. SRC TechCon*, 2005, pp. 1–5.

[7] R. Venkatesan et al., "MAGNet: A modular accelerator generator for neural networks," in *Proc. IEEE/ACM ICCAD*, 2019, pp. 1–8.

[8] X. Zhang et al., "DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs," in *Proc. IEEE/ACM ICCAD*, 2018, pp. 1–8.

[9] "Pytorch." Accessed: Sep. 17, 2022. [Online]. Available: https://pytorch.org/

[10] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proc. ACM/IEEE 43rd ISCA*, 2016, pp. 367–379.

[11] Z. Du et al., "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, 2015, pp. 92–104.

[12] "Avnet Ultra96." xilinx. Accessed: Sep. 1, 2019. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/1-vad4rl.html

[13] "Jetson TX2." nvidia. Accessed: Apr. 15, 2024. [Online]. Available: https://developer.nvidia.com/embedded/jetson-tx2

[14] "Edge TPU." google. Accessed: Sep. 1, 2019. [Online]. Available: https://coral.withgoogle.com/docs/edgetpu/faq/

[15] P. Xu et al., "AutoDNNchip: An automated DNN chip predictor and builder for both FPGAS and ASICs," in *Proc. FPGA*, 2020, pp. 40–50.

[16] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN accelerator efficiency through resource partitioning," in *Proc. ISCA*, 2017, pp. 535–547.

[17] A. Samajdar, P. Mannan, K. Garg, and T. Krishna, "GeneSys: Enabling continuous learning through neural network evolution in hardware," in *Proc. 51st IEEE/ACM MICRO*, 2018, pp. 855–866.

[18] S.-C. Kao, G. Jeong, and T. Krishna, "ConfuciuX: Autonomous hardware resource assignment for DNN accelerators using reinforcement learning," in *Proc. 53rd IEEE/ACM MICRO*, 2020, pp. 622–636.

[19] Y. Li et al., "EDD: Efficient differentiable DNN architecture and implementation co-search for embedded AI solutions," in *Proc. 57th ACM/IEEE DAC*, 2020, pp. 1–6.

[20] Y. Zhang et al., "DIAN: Differentiable accelerator-network co-search towards maximal DNN efficiency," in *Proc. IEEE/ACM ISLPED*, 2021, pp. 1–6.

[21] X. Zhang et al., "SkyNet: A hardware-efficient method for object detection and tracking on embedded systems," 2020, *arXiv:1909.09709*.

[22] Y. Zhao et al., "SmartExchange: Trading higher-cost memory storage/access for lower-cost computation," in *Proc. ACM/IEEE 47th ISCA*, 2020, pp. 954–967.

[23] W. Li, P. Xu, Y. Zhao, H. Li, Y. Xie, and Y. Lin, "Timely: Pushing data movements and interfaces in PIM accelerators towards local and in time domain," in *Proc. ACM/IEEE 47th ISCA*, 2020, pp. 832–845.

[24] A. Parashar et al., "Timeloop: A systematic approach to DNN accelerator evaluation," in *Proc. ISPASS*, 2019, pp. 304–315.

[25] Y. Zhao, C. Li, Y. Wang, P. Xu, Y. Zhang, and Y. Lin, "DNN-chip predictor: An analytical performance predictor for DNN accelerators with various dataflows and hardware architectures," in *Proc. IEEE ICASSP*, 2020, pp. 1593–1597.

[26] X. Yang et al., "Interstellar: Using halide's scheduling language to analyze DNN accelerators," in *Proc. 25th ASPLOS*, 2020, pp. 369–383.

[27] H. Kwon et al., "MAESTRO: An open-source infrastructure for modeling dataflows within deep learning accelerators," 2018, *arXiv:1805.02566*.

[28] Y. Fu, Y. Zhang, Y. Zhang, D. Cox, and Y. Lin, "Auto-NBA: Efficient and effective search over the joint space of networks, bitwidths, and accelerators," in *Proc. ICML*, 2021, pp. 3505–3517.

[29] Y. Zhang, H. You, Y. Fu, T. Geng, A. Li, and Y. Lin, "G-CoS: GNN-accelerator co-search towards both better accuracy and efficiency," in *Proc. IEEE/ACM ICCAD*, 2021, pp. 1–9.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE CVPR*, 2016, pp. 770–778.

[31] Y. Zhao et al., "I-FlatCam: A 253 FPS, 91.49 μj/frame ultra-compact intelligent lensless camera for real-time and efficient eye tracking in vr/ar," in *Proc. VLSI*, 2022, pp. 108–109.

[32] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous dataflow accelerators for multi-DNN workloads," in *Proc. HPCA*, 2021, pp. 71–83.

[33] C. Hao et al., "FPGA/DNN Co-design: An efficient design methodology for IoT intelligence on the edge," in *Proc. 56th Annu. Design Autom. Conf.*, 2019, pp. 1–6.

[34] R. C. Correa, A. Ferreira, and P. Rebreyend, "Scheduling multiprocessor tasks with genetic algorithms," *Trans. Parallel Distrib. Syst.*, vol. 10, no. 8, pp. 825–837, Aug. 1999.

[35] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. CVPR*, 2018, pp. 4510–4520.

[36] (Xilinx Semicond. Manuf. Commer. Co., San Jose, CA, USA). *Xilinx ZC706 Evaluation Kit*. Accessed: Sep. 30, 2020. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html

[37] B. Wu et al., "FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search," in *Proc. IEEE/CVF CVPR*, 2019, pp. 10734–10742.

[38] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," 2020, *arXiv:1905.11946*.

[39] Q. Xiao, Y. Liang, L. Lu, S. Yan, and Y. W. Tai, "Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs," in *Proc. 54th DAC*, 2017, pp. 1–6.

[40] J. Qiu et al., "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. ACM/SIGDA FPGA*, 2016, pp. 26–35.

[41] (Xilinx Semicond. Manuf. Commer. Co., San Jose, CA, USA). *Chaidnnv2: Hls Based Deep Neural Network Accelerator Library for Xilinx Ultrascale+ MPSoCs*. Accessed: Dec. 1, 2020. [Online]. Available: https://github.com/Xilinx/CHaiDNN

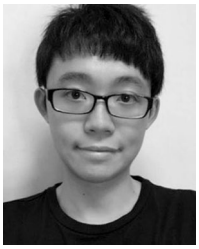
[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[43] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015, *arXiv:1409.1556*.

[44] H. You, T. Geng, Y. Zhang, A. Li, and Y. Lin, "GCoD: Graph convolutional network acceleration via dedicated algorithm and accelerator co-design," in *Proc. HPCA*, 2022, pp. 460–474.

Yongan Zhang received the B.S. and M.S. degrees from Rice University, Houston, TX, USA, in 2019 and 2023, respectively. He is currently pursuing the Ph.D. degree with the School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA.

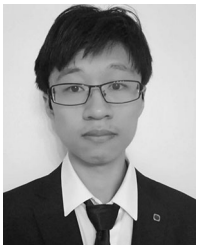
His research interests include AI-assisted hardware design, deep learning accelerator design, and hardware–software co-design.



Xiaofan Zhang (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign, Champaign, IL, USA, in 2022.

He is currently a Software Engineer with Google, Mountain View, CA, USA, with the focus on developing and optimizing large-scale AI systems for large model training and serving. His research interests include AI Systems, energy-efficient computing, and HW/SW co-design.

Dr. Zhang has received the IEEE/ACM William J. Mccalla ICCAD Best Paper Award in 2018, the IEEE/ACM Design Automation Conference System Design Contest Double Championships (2019), the Sundaram Seshu International Student Fellowship in 2020, the Google Ph.D. Fellowship in 2020, the First-Place Winner Award of ACM Student Research Competition at ICCAD in 2021, the Rambus Computer Engineering Fellowship in 2021, and the Mavis Future Faculty Fellowship in 2021.



Pengfei Xu received the B.S. degree from Shanghai Jiao Tong University, Shanghai, China, in 2017, and the M.Eng. degree from Rice University, Houston, TX, USA, in 2019.

He is currently an Engineer with Samsung. His research interests include deep learning accelerator design, hardware–software co-design, and hardware design automation.

Yang (Katie) Zhao received the Ph.D. degree from Rice University, Houston, TX, USA, in 2023.

She is an Assistant Professor with the Department of Electrical and Computer Engineering, University of Minnesota Twin Cities, Minneapolis, MN, USA. Prior to this, she spent 2023 as a Postdoctoral Research Fellow with the Georgia Institute of Technology, Atlanta, GA, USA. Her research interest resides at the intersection of computer architecture, hardware design, and machine learning (ML), with a recent focus on hardware acceleration for emerging ML models, such as large language models.

Dr. Zhao received the ML and Systems Rising Stars in 2023, the Ralph Budd Award for Best Thesis in the School of Engineering Rice University in 2023, the First Place in the University Demo Best Demonstration at DAC in 2022, and the Cadence Women in Technology Scholarship in 2020.

Cong (Callie) Hao (Associate Member, IEEE) received the Ph.D. degree from Waseda University, Tokyo, Japan, in 2017.

She was a Postdoctoral Fellow with the Georgia Institute of Technology (Georgia Tech), Atlanta, GA, USA, from 2020 to 2021 and also worked as a Postdoctoral Researcher of ECE with the University of Illinois at Urbana–Champaign, Champaign, IL, USA, from 2018 to 2020. She is currently an Assistant Professor with the Department of Electrical and Computer Engineering, Georgia Tech. Her current research interests lie in the joint area of efficient hardware design and machine learning algorithms, reconfigurable and high-efficiency computing, and electronic design automation tools.

Dr. Hao was a recipient of the NSF CAREER Award.

Deming Chen (Fellow, IEEE) received the Ph.D. degree from the Computer Science Department, University of California at Los Angeles, Los Angeles, CA, USA, in 2005.

He is the Abel Bliss Professor of the Grainger College of Engineering, University of Illinois at Urbana-Champaign (UIUC), Champaign, IL, USA. He is the Director of the AMD-Xilinx Center of Excellence and the Hybrid-Cloud Thrust CoLead of the IBM-Illinois Discovery Accelerator Institute, UIUC. He has published more than 250 research papers, received ten best paper awards and one ACM/SIGDA TCFPGA Hall-of-Fame Paper Award, and given more than 140 invited talks. His current research interests include reconfigurable computing, hybrid cloud, system-level design methodologies, machine learning and acceleration, and hardware security.

Dr. Chen is an ACM Distinguished Speaker and the Editor-in-Chief of the *ACM Transactions on Reconfigurable Technology and Systems.*

Yingyan (Celine) Lin (Member, IEEE) received the Ph.D. degree in ECE from the University of Illinois at Urbana-Champaign, Champaign, IL, USA, in 2017.

She is currently an Associate Professor with the School of Computer Science, Georgia Institute of Technology, Atlanta, GA, USA. Her research interests include efficient machine learning systems via cross-layer innovations, aiming to develop efficient algorithms, accelerators, and automated tools for enabling ubiquitous on-device intelligence, and promoting green AI.

Dr. Lin was a recipient of the Best Student Paper Award from the 2016 IEEE International Workshop on Signal Processing Systems (SiPS 2016), the 2016 Robert T. Chien Memorial Award from UIUC for Excellence in Research, and was selected as a Rising Star in EECS by the 2017 Academic Career Workshop for Women at Stanford University. She was a recipient of the NSF CAREER Award, the IBM Faculty Award, and the Meta Faculty Research Award, and recently received the ACM SIGDA Outstanding Young Faculty Award. She served as the Program Co-Chair for the 32nd IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP 2021). She is currently an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS.