ICCAD: G: Bridge the Hardware-Software Gap: Exploring End-to-End Design Flows for Building Efficient AI Systems

Xiaofan Zhang (*xiaofan3@illinois.edu*), ACM ID: 1226536, Graduate
ECE, University of Illinois at Urbana-Champaign, Urbana, IL, 61801

1 PROBLEM AND MOTIVATION

The success of artificial intelligence (AI) has been driven in part by the improvement of deep neural networks (DNNs) with deeper and more sophisticated layer interconnections [1–8]. This in turn, puts stringent design requirements on AI systems to deliver not only high inference accuracy but satisfied inference speed, throughput, and energy efficiency. Building AI systems with customized hardware accelerators is one of the most promising approaches, which offloads or partitions workloads for a more efficient process [9–14]. However, building customized hardware accelerators still presents great challenges, such as tedious programming using hardware description languages, intricate hardware verification problems, and time-consuming design space explorations. Also, diverse AI application scenarios further exacerbate the design difficulties and the gap between fast DNN construction in software and slow customized accelerator implementation in hardware is still significant. To address these challenges, we explore end-to-end design flows to bridge the software-hardware gap and deliver efficient AI systems for various applications and hardware setups. The design overview is shown in Figure 1, and our main contributions are as follows:

- We propose ***F-CAD*** [15], the first end-to-end design flow to accommodate the latest virtual reality (VR) application using edge devices. F-CAD introduces a novel elastic accelerator architecture and a dynamic design space exploration to optimize hardware designs. Compared to the state-of-the-art designs, F-CAD achieves 4.0× and 2.8× higher throughput, 62.5% and 21.2% higher efficiency than [16] and [17] by targeting the same task and hardware device.
- We propose ***SkyNet*** [18], the champion design flow of *DAC System Design Contest (DAC-SDC)* to leverage real-time object detection at the edge. SkyNet optimizes both DNN models and customized hardware accelerators by using hardware-software co-design strategies. It significantly outperforms all other 100+ competitors in both CPU-FPGA and CPU-GPU platforms.
- We propose ***EcoSys*** [19], the first end-to-end design flow for CPU-FPGA heterogeneous systems with coherent memory space. EcoSys helps reduce 19.96% host-accelerator data transfer latency and delivers 20.6× and 5.3× higher inference throughput compared to optimized designs using pure server CPU and FPGA, respectively.

2 BACKGROUND AND RELATED WORK

To overcome the complicated and resource-demanding AI workloads, there has been continuous progress in building domain-specific hardware accelerators for more efficient DNN deployment. These accelerators attempt to take advantage of different hardware designs, such as adopting acceleration libraries on CPU [20], exploring kernel optimization on GPUs [21], building customized accelerators on FPGAs [11, 12, 16] and ASICs [9, 10]. Among them, FPGA- and ASIC-based designs can be fully customized to achieve more superior latency, throughput, and energy efficiency. Due to the design flexibility, innovations also happen in the accelerator architecture. For example, a fine-grained layer-based pipeline architecture is proposed in [16] to significantly lower the end-to-end latency, while a new accelerator paradigm is explored in [22] to better balance the design specificity and scalability.

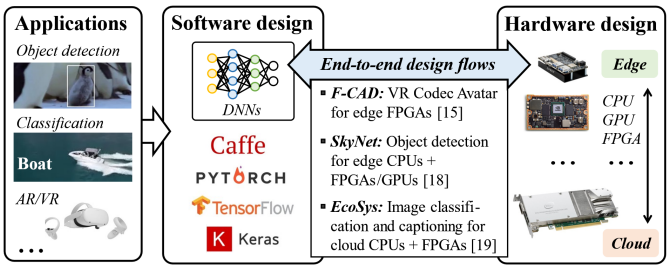

Figure 1: Comprehensive end-to-end design flows are explored in this work to address the challenges of building efficient edge and cloud AI systems.

Researchers also focus on building hardware-efficient DNN models with reduced computation demands and memory footprints. Prior works have demonstrated the possibility of using low bit-width data to represent original floating-point parameters, such as using ternary or even binary network designs [23–25]. These solutions are intended to replace the hardware-intensive floating-point multiplications by logical operations so that DNNs can be more efficient on hardware platforms. Network pruning is proposed to reduce the redundancy of DNN structures [26, 27]. By applying network pruning, the relatively less important connections between DNN layers are discarded, and network retraining is then performed to regain accuracy. Besides, researchers attempt to reduce the network's computation complexity by using depth-wise separable convolutional layers [28]. Such a depth-wise separable structure can effectively reduce the number of operations and provide more compact DNN designs. To further improve the DNN deployment on hardware, layer fusion is proposed in [29] to minimize data movements between on-chip and off-chip memory.

To improve AI system design efficiency, recent works have started focusing on building automation tools to assemble RTL or High-level Synthesis (HLS) building blocks into the full implementation of the customized accelerator [16, 22, 30, 31]. These proposed tools support DNN inputs described by a much higher abstraction level used in popular machine learning frameworks, such as Caffe [32], TensorFlow [33], and PyTorch [34]. These efforts help alleviate slow hardware implementation during AI system design. They also provide opportunities to integrate design space exploration to optimize both hardware accelerator and DNN designs to meet specific performance targets given the available resource budgets [35, 36].

However, existing studies still suffer several drawbacks when handling emerging AI applications, such as the defective scalability for mapping emerging DNNs to resource-constrained devices, the lack of support of heterogeneous computing platforms, and the insufficient co-optimization strategies for hardware and software designs. Therefore, in this work, we propose more comprehensive end-to-end design flows to address these drawbacks and facilitate demanding AI applications.

3 APPROACH AND UNIQUENESS

In this section, we propose three end-to-end design flows, including F-CAD [15], SkyNet [18], and EcoSys [19], to deliver high-quality and efficient AI services on various hardware platforms.

3.1 F-CAD: Enabler for Efficient VR Acceleration

VR is a major area AI technologies play essential roles, and it urgently needs efficient hardware acceleration. The emerging VR applications are compute- and memory-intensive and require real-time and high-quality image rendering on edge devices (e.g., VR headsets). Take codec avatar as an example. It is one of the most impressive breakthroughs that enables immersed communications in VR with photo photo-realistic and three-dimensional human appearances and real-time expressions [7]. The latest DNN model to render codec avatar contains multiple DNN branches and complicated layer dependencies with more than 13.6 GOP (Giga operations) and 7.2 million parameters. It is required to serve with real-time response and high throughput (90 or even 120 FPS) for smooth user interactions. However, most hardware accelerators can not fulfill the performance needs with limited computation, memory, and power budgets at the edge. To address these challenges, we propose F-CAD [15], an end-to-end flow for implementing customized edge accelerators for VR codec avatars with high throughput and strong real-time performance. The uniqueness of this work includes the following two parts.

1) The Elastic Accelerator Architecture: Since the targeted VR workloads adopt multi-branch DNNs, we propose an expandable elastic architecture to support these unique network architectures. The proposed elastic accelerator follows the paradigm in Fig. 2 (a), where inputs of each branch are processed in a pipeline manner and passed through all pipeline stages belonging to that branch. It can be flexibly expanded by following two dimensions as the pipeline stage number and the branch number. As shown in Fig. 2 (b), the proposed architecture consists of basic architecture units (BAUs) arranged in a two-dimensional plane. Each unit is responsible for one pipeline stage. For example, the expansion following the X-axis means more stages (three in this case) need to be handled in this branch, while the expansion along the Y-axis represents more branches are used in the targeted DNN. In this example, F-CAD generates an accelerator with three pipelines corresponding to Br. 1 ~ 3. Inside the BAU, F-CAD allocates computation resources (for process engine), on-chip memory (for input and weight buffers), and external memory resources to allow timely data supply and sufficient computational parallelism. It supports three types of parallelism: the first two along the output and input channels (kernel parallelism factor *kpf*,

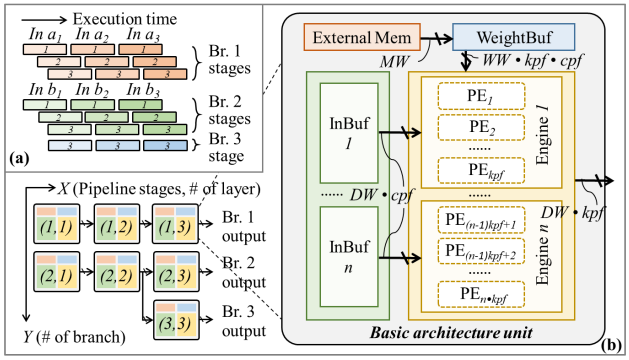

Figure 2: F-CAD proposes (a) a fine-grained layer-based multi-pipeline accelerator with (b) an expansion capability along X- and Y-axis to instantiate multiple basic architecture units (BAUs) according to the targeted DNN.

Table 1: The multi-branch dynamic design space

Br.	Hardware Configurable Parameters
1	$config^1 \leftarrow bsize^1, cpf_1^1 \cdots cpf_l^1, kpf_1^1 \cdots kpf_l^1, h_1^1 \cdots h_l^1$
2	$config^2 \leftarrow bsize^2, cpf_1^2 \cdots cpf_m^2, kpf_1^2 \cdots kpf_m^2, h_1^2 \cdots h_m^2$
...	.....
B	$config^B \leftarrow bsize^B, cpf_1^B \cdots cpf_n^B, kpf_1^B \cdots kpf_n^B, h_1^B \cdots h_n^B$
Customization	$Q, BSize_1 \cdots BSize_B, P_1 \cdots P_B$
Resource budgets	$C_{max}, M_{max}, BW_{max}$

channel parallelism factor *cpf*), and the last one along with the input feature map (*H-partition*).

2) The Multi-Branch Dynamic Design Space Exploration: The proposed elastic architecture introduces a comprehensive and dynamic design space. We summarize it in Table 1. Assuming a DNN with B branches, we pick the first two (with l and m stages) and the last branch (with n stages) as examples. Each of them can be configured in four categories as *kpf*, *cpf*, *H-partition* (corresponding to the BAU configuration), and batch size (*bsize*). User-defined customization includes the data quantization (Q) and the branch priority (P) to indicate the different importance of each branch. The resource budgets specify three major resources as computation C_{max}, on-chip memory M_{max}, and external memory access bandwidth BW_{max}. To effectively search for the best accelerator configuration, we propose a DSE engine following a divide and conquer strategy. It contains two steps: 1) a cross-branch stochastic search to first confirm the resource distribution for every branch; and 2) an in-branch greedy search to explore the optimized accelerator configuration with given resource budgets. Experiments show that the proposed exploration can converge in minutes using a single CPU and deliver optimized configuration guidelines for customized VR accelerators.

3.2 SkyNet: Enabler for Efficient Edge AI with HW/SW Co-Design

Enabling edge AI systems is nontrivial as it requires sophisticated cross-stack optimization to satisfy various real-life requirements. In general, the edge AI system design follows a top-down design flow. It starts with selecting a reference DNN by concentrating on

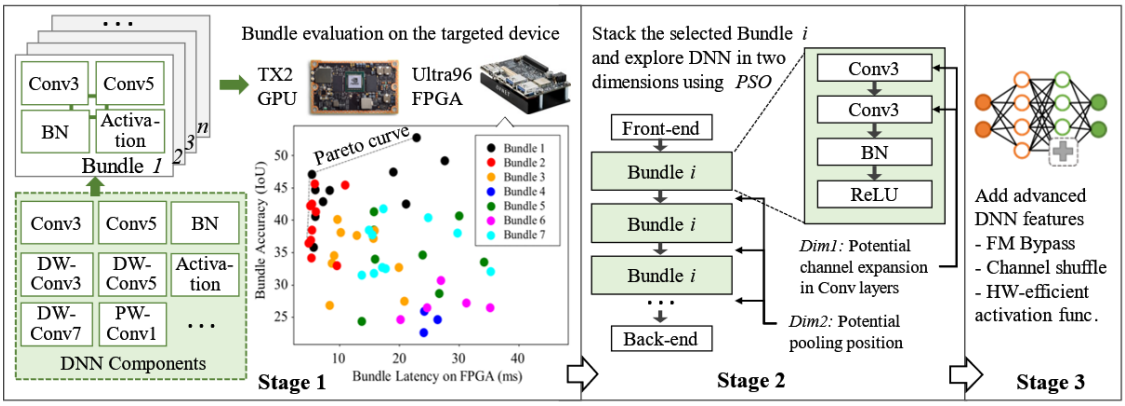

Figure 3: SkyNet features a bottom-up DNN design to capture realistic hardware constraints for edge AI systems.

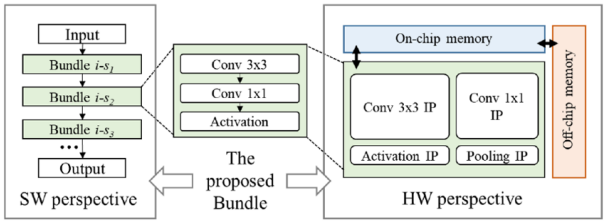

Figure 4: Bundle: a unified intermediate representation.

network inference accuracy. As the DNN is too complicated for embedded systems, it is required to have hardware-related optimization, which helps compress the model size and improve its hardware performance. However, such a process may also cause quality degradation and require optimization on the software side. Eventually, we will have an iterative process between hardware and software optimization until the compressed model is ready to deploy on the targeted device. Following this flow makes it challenging to balance the hardware and software metrics (e.g., inference accuracy vs. throughput performance) and hard to select appropriate reference DNNs at the beginning (as certain benchmark DNNs may fail to handle customized applications). To address these issues, we propose SkyNet, an efficient design flow for building edge AI systems. SkyNet has been demonstrated by winning a competitive system design contest (DAC-SDC) for low-power real-time object detection. It has two significant features to enable a collaborative design and optimization of DNNs and their hardware deployments.

1) The unified HW/SW Intermediate Representation: The most critical task for SkyNet is to find an intermediate representation for effective communication between hardware and software. So, we propose Bundle. As shown in Fig. 4, a Bundle consists of two attributes: From a software perspective, it is a set of sequential DNN layers, which can be repeatedly stacked for constructing DNNs; From a hardware perspective, it is a set of IPs to be implemented on hardware. By constructing DNNs from Bundles, network designs can inherently consider hardware and software metrics to enable efficient edge AI systems.

2) The Bottom-up Hardware-Efficient DNN Design: The other feature proposed by SkyNet is a three-stage approach (Fig. 3) for hardware-efficient DNN designs. In stage 1, we enumerate DNN components and assemble them into Bundle 1 ~ n. Each Bundle is then implemented and evaluated in the targeted hardware for realistic hardware metrics. To get their potential accuracy contribution, we build DNN sketches with fixed front- and back-end structures, and respectively insert one type of Bundle (with replications) in the middle. These DNN sketches are fast trained using targeted datasets to obtain software metrics. In stage 2, we perform a hardware-aware DNN architecture search to discover the most suitable network that satisfies the targeted software and hardware metrics (e.g., DNN accuracy and throughput performance). We propose a group-based particle swarm optimization (PSO) algorithm to solve such a multi-objective optimization process. During the search, we also consider two architecture variations as the channel expansion factor and the pooling spot to facilitate network diversity. From the design methodology perspective, this stage can be extended flexibly to support richer network configuration and can adopt other optimization algorithms with multi-objective support (e.g., random search and Bayesian Optimization). In the last stage, more advanced design features are added if hardware metrics allow, such as a bypass from low-level features to high-level features along with feature map reordering [37] and more hardware-efficient activation.

3.3 EcoSys: Enabler for Efficient Cloud AI with Heterogeneous Computing

Compared to edge applications, cloud AI applications are more heavy-duty and rely more on powerful heterogeneous clusters with general-purpose and domain-specific processors. Among these processors, the customized accelerator, such as the FPGA- or ASIC-based design, is one of the most promising candidates for running AI workloads as they can be fully customized for improved performance and efficiency. However, building customized accelerators requires RTL programming, hardware verification, and precise resource allocation. All of which can be time-consuming and challenging even for seasonal hardware developers. These negative aspects often hinder their adoption by application developers who are used to working on high-level programming and have fewer experiences deploying targeted applications onto heterogeneous systems with customized accelerators. Challenges also come from the massive memory demand caused by AI workloads. As it is impossible to

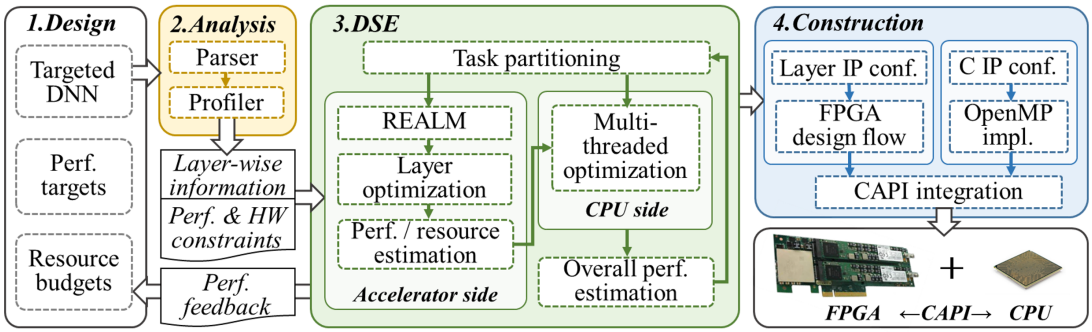

Figure 5: EcoSys proposes a design and optimization flow to deliver optimized heterogeneous systems for cloud AI.

store all the data locally, a sophisticated memory management design is required to diminish extra data transfer overheads. To address these issues, EcoSys substantially raises the accelerator design abstraction level by supporting Python-based DNN descriptions and provides an efficient approach to design, optimize, and integrate customized accelerators into heterogeneous systems for running AI applications. Major features are as follows:

1) The Design Flow for Customized Heterogeneous Systems: EcoSys provides an end-to-end design and optimization flow from DNN design developed by machine learning frameworks to its hardware implementation on a heterogeneous system with CPUs and customized accelerators built by FPGAs (Figure 5). In the *Design* step, the targeted DNN's Python-based definition files and parameters are collected after training on machine learning frameworks. Meanwhile, we can set up a performance target (e.g., throughput performance) and specify available hardware budgets. Model analysis is then processed in the *Analysis* step to understand the input model and capture layer-wise compute and memory access patterns. When all the information is ready, the *DSE* step is launched to explore hardware configurations for accelerating the targeted DNN. Its first task is to partition and distribute input workloads to different devices. On the accelerator side, resource allocation is executed to provide hardware configuration guidelines given the design space and available resources, such as DSPs (compute resources), BRAMs (on-chip memory resources), and external memory access bandwidth. On the CPU side, multi-threaded optimization is applied for optimized system-level performance. Design guidelines are then passed to the *Construction* step for final workload deployment. The workload execution follows a pipeline manner across the accelerator and the CPU to maximize system-level throughput.

2) The Hardware-Software Co-Optimization Methods: We create a coherent memory space between the host CPU and the FPGA accelerator. It intends to provide ample memory space to accommodate complicated DNN models and reduce data transfer and synchronization latency from one device to the other. We also propose diverse optimization strategies to enable precise workload partition considering compute and memory demands, available resource budgets, and performance targets. For customized accelerator design, we include a layer-based pipeline to boost throughput performance, a feature map partition scheme to minimize on-chip memory utilization, highly configurable IPs for layer dedicated hardware designs, and an efficient memory hierarchical design for timely data supply. We introduce OpenMP templates for multi-threading layer implementation on the CPU.

Table 2: Result comparison for VR hardware acceleration using ASIC and ZU9CG FPGA

	865 SoC [38]	[16]	[17]	F-CAD	
Precision	8-bit	8-bit	16-bit	8-bit	16-bit
DSP	-	1820	1024	2229	2213
BRAM	-	1197	1120	1168	1735
FPS	35.8	30.5	22.0	**122.1**	**61.0**
Efficiency	16.9%	28.8%	70.4%	**91.3%**	**91.6%**

4 RESULTS AND CONTRIBUTIONS

4.1 F-CAD

To demonstrate F-CAD's capability for accelerating VR codec avatar, we select three existing accelerators from industry (Snapdragon 865 SoC [38]) and academia (DNNBuilder [16], HybridDNN [17]) for comparison. We collect two metrics, FPS (Frame Per Second) and efficiency (the ratio between actual and theoretical peak throughput), during comparison and present the results in Table 2. The 865 SoC delivers 35.8 FPS, and its major bottleneck is the limited cache size, which causes frequent data transfers and severely restricts performance. So its overall efficiency barely reaches 16.9%. For FPGA designs, the FPS and efficiency of DNNBuilder are limited by insufficient parallelism, so the allocated resources are not fully utilized. HybridDNN fails to scale up when more resources are available because its coarse-grained configuration requires a double-sized accelerator instance to continue scaling. In our design, F-CAD delivers the highest FPS (122.1) and efficiency (91.6%) given the same resource budgets, which can perfectly meet the demanding VR requirements.

4.2 SkyNet

We demonstrate the proposed designs by joining a competitive System Design Contest at the 56th IEEE/ACM Design Automation Conference (DAC-SDC). This competition features a low power object detection challenge using resource-constrained embedded systems, and its evaluation is based on detection accuracy (IoU), inference throughput (FPS), and energy consumption (J). After specifying the targeted applications and hardware resource constraints, SkyNet generates a compact hardware-efficient DNN with 13 Conv layers and 1.82 MB parameters. We compare the top-3 teams in DAC-SDC and present the results in Table 3. SkyNet achieves the highest accuracy and total score by outperforming 100+ competitors in both GPU and FPGA categories. With the state-of-the-art edge AI solutions, SkyNet becomes the first design that wins a double championship in the history of DAC-SDC.

Table 3: Top-3 teams from DAC-SDC'19

Team Name	IoU	FPS	Power(W)	Total Score
TX2 CPU-GPU platform				
SkyNet	**0.731**	**67.33**	13.50	**1.504**
Thinker [39]	0.713	28.79	**8.55**	1.442
DeepZS [40]	0.723	26.37	15.12	1.422
Ultra96 CPU-FPGA platform				
SkyNet	**0.716**	25.05	7.26	**1.526**
XJTU_Tripler [41]	0.615	50.91	9.25	1.394
SystemsETHZ [42]	0.553	**55.13**	6.69	1.318

4.3 EcoSys

We demonstrate the EcoSys framework to accelerate the long-term recurrent convolution network (LRCN), which contains a hybrid DNN structure with convolutional and recurrent neural networks. It works for analyzing the input video and outputs one semantic caption for each frame. EcoSys leverages the CPU-FPGA heterogeneous systems with CAPI-based coherent memory space [43] and delivers 314.7 and 58.1 frames per second (FPS) by targeting the LRCN model with AlexNet and VGG-16 backbone, respectively. Compared to the existing multithreaded CPU (POWER9) and pure FPGA (VU35P FPGA) solutions, EcoSys achieves 20.6× and 5.3× higher throughput performance.

4.4 Research Impacts

This work is one primary focus of my Ph.D. research, and it has led to **eight first- and co-first-authored publications** in premier EDA/ML/System conferences and journals (e.g., TCAD, TPDS, DAC, ICCAD, MLSys) [15, 16, 18, 19, 22, 35, 44, 45]. Among them, DNNBuilder [16] won the ICCAD William J. McCalla **Best Paper Award** in 2018 and the IBM AI Horizons Network **Best Poster Award** in 2018 by delivering state-of-the-art DNN accelerators. The hardware-software co-design method proposed in [46] won the **Best Poster Award** in the ICML Workshop on ODML-CDNNR.

My research achievements have been recognized by multiple fellowships and awards from industry and academia, which include the **Google Ph.D. Fellowship** (2020), the **Sundaram Seshu International Student Fellowship** (2020), the **Rambus Computer Engineering Fellowship** (2021), the **Mavis Future Faculty Fellowship** (2021), and the **Gold Medal in ACM/SIGDA student research competition** (2021). In addition, this work has led to multiple breakthroughs in delivering efficient AI systems and significantly improving AI applications' accessibility, reliability, and sustainability. The novel designs in this work have contributed to IBM Research, Facebook Reality Labs (FRL) Research, and Google Research with hardware and software innovations to solve the most pressing issues facing the new AI computing era.

REFERENCES

[1] Alex Krizhevsky et al. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.
[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
[3] Christian Szegedy et al. Going deeper with convolutions. In *CVPR*, 2015.
[4] Kaiming He et al. Deep residual learning for image recognition. In *CVPR*, 2016.
[5] Barret Zoph et al. Learning transferable architectures for scalable image recognition. In *CVPR*, 2018.
[6] Esteban Real et al. Regularized evolution for image classifier architecture search. In *AAAI*, 2019.
[7] Stephen Lombardi et al. Deep appearance models for face rendering. *ACM Transactions on Graphics (TOG)*, 37(4):1–13, 2018.
[8] Jacob Devlin et al. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL: Human Language Technologies*, 2019.
[9] Norman P Jouppi et al. In-datacenter performance analysis of a tensor processing unit. In *ISCA*, 2017.
[10] Yu-Hsin Chen et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In *ISSCC*, 2016.
[11] Chen Zhang et al. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *FPGA*, 2015.
[12] Jiantao Qiu et al. Going deeper with embedded FPGA platform for convolutional neural network. In *FPGA*, 2016.
[13] Xiaofan Zhang et al. High-performance video content recognition with long-term recurrent convolutional network for FPGA. In *FPL*, 2017.
[14] Yao Chen et al. Cloud-DNN: An open framework for mapping DNN models to cloud FPGAs. In *FPGA*, 2019.
[15] Xiaofan Zhang et al. F-CAD: A framework to explore hardware accelerators for codec avatar decoding. In *DAC*, 2021.
[16] Xiaofan Zhang et al. DNNBuilder: an automated tool for building high-performance DNN hardware accelerators for FPGAs. In *ICCAD*, 2018.
[17] Hanchen Ye et al. HybridDNN: A framework for high-performance hybrid DNN accelerator design and implementation. In *DAC*, 2020.
[18] Xiaofan Zhang et al. SkyNet: a hardware-efficient method for object detection and tracking on embedded systems. In *MLSys*, 2020.
[19] Xiaofan Zhang et al. Exploring HW/SW co-design for video analysis on CPU-FPGA heterogeneous systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.
[20] Bryan Brown. Intel® math kernel library for deep learning networks. 2018.
[21] Dustin Franklin. NVIDIA Jetson AGX Xavier delivers 32 teraops for new era of AI in robotics. *NVIDIA Accelerated Computing| Parallel For all*, 2018.
[22] Xiaofan Zhang et al. DNNExplorer: a framework for modeling and exploring a novel paradigm of FPGA-based DNN accelerator. In *ICCAD*, 2020.
[23] Junsong Wang et al. Design flow of accelerating hybrid extremely low bit-width neural network in embedded FPGA. In *FPL*, 2018.
[24] Dibakar Gope et al. Ternary hybrid neural-tree networks for highly constrained IoT applications. 2019.
[25] Yichi Zhang et al. FracBNN: Accurate and fpga-efficient binary neural networks with fractional activations. In *FPGA*, 2021.
[26] Song Han et al. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015.
[27] Jian-Hao Luo et al. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017.
[28] Andrew G Howard et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.
[29] Manoj Alwani et al. Fused-layer CNN accelerators. In *International Symposium on Microarchitecture*, 2016.
[30] Xuechao. Wei et al. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In *DAC*, 2017.
[31] Yijin Guan et al. FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates. In *FCCM*, 2017.
[32] Yangqing Jia et al. Caffe: Convolutional architecture for fast feature embedding. In *ACM Multimedia*, 2014.
[33] Martín Abadi et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, 2016.
[34] Adam Paszke et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.
[35] Cong Hao et al. FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge. In *DAC*, 2019.
[36] Weiwen Jiang et al. Accuracy vs. efficiency: Achieving both through FPGA-implementation aware neural architecture search. In *DAC*, 2019.
[37] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *CVPR*, 2017.
[38] Qualcomm. Snapdragon 865 5G mobile platform.
[39] Feng Xiong et al. DAC-SDC'19 2nd place winner in GPU track, 2019.
[40] Jianing Deng et al. DAC-SDC'19 3rd place winner in GPU track, 2019.
[41] Boran Zhao et al. DAC-SDC'19 2nd place winner in FPGA track, 2019.
[42] Kaan Kara et al. DAC-SDC'19 3rd place winner in FPGA track, 2019.
[43] Jeffrey Stuecheli et al. CAPI: A coherent accelerator processor interface. *IBM Journal of Research and Development*, 59(1):1–7, 2015.
[44] Qin Li et al. Implementing neural machine translation with bi-directional GRU and attention mechanism on FPGAs using HLS. In *ASP-DAC*, 2019.
[45] Qin Li et al. Efficient methods for mapping neural machine translator on fpgas. *IEEE Transactions on Parallel and Distributed Systems*, 32(7):1866–1877, 2020.
[46] Xiaofan Zhang et al. A bi-directional co-design approach to enable deep learning on iot devices. In *ICML Workshop on On-Device Machine Learning & Compact Deep Neural Network Representations*, 2019.