DNNGuard: An Elastic Heterogeneous DNN Accelerator Architecture against Adversarial Attacks

Xingbin Wang
State Key Laboratory of Information Security, Institute of Information Engineering, CAS
School of Cyber Security, University of Chinese Academy of Sciences
wangxingbin@iie.ac.cn

Rui Hou*
State Key Laboratory of Information Security, Institute of Information Engineering, CAS
hourui@iie.ac.cn

Boyan Zhao
State Key Laboratory of Information Security, Institute of Information Engineering, CAS
zhaoboyan@iie.ac.cn

Fengkai Yuan
State Key Laboratory of Information Security, Institute of Information Engineering, CAS
yuanfengkai@iie.ac.cn

Jun Zhang
Hubei University of Arts and Science
zhangjun@hbuas.edu.cn

Dan Meng
State Key Laboratory of Information Security, Institute of Information Engineering, CAS
mengdan@iie.ac.cn

Xuehai Qian
University of Southern California
xuehai.qian@usc.edu

ABSTRACT

Recent studies show that Deep Neural Networks (DNN) are vulnerable to adversarial samples that are generated by perturbing correctly classified inputs to cause the misclassification of DNN models. This can potentially lead to disastrous consequences, especially in security-sensitive applications such as unmanned vehicles, finance and healthcare. Existing adversarial defense methods require a variety of computing units to effectively detect the adversarial samples. However, deploying adversary sample defense methods in existing DNN accelerators leads to many key issues in terms of cost, computational efficiency and information security. Moreover, existing DNN accelerators cannot provide effective support for special computation required in the defense methods.

To address these new challenges, this paper proposes *DNNGuard*, an elastic heterogeneous DNN accelerator architecture that can efficiently orchestrate the simultaneous execution of original (*target*) DNN networks and the *detect* algorithm or network that detects adversary sample attacks. The architecture tightly couples the DNN accelerator with the CPU core into one chip for efficient data transfer and information protection. An elastic DNN accelerator is designed to run the target network and detection network simultaneously. Besides the capability to execute two networks at the same time, DNNGuard also supports the non-DNN computing and allows the special layer of the neural network to be effectively supported by the CPU core. To reduce off-chip traffic and improve resources utilization, we propose a dynamical resource scheduling mechanism. To build a general implementation framework, we propose an *extended AI instruction set* for neural networks synchronization, task scheduling and efficient data interaction. We implement DNNGuard based on RISC-V and NVDLA, and evaluate its performance impacts with six target networks and three typical detection networks. Experiment results show that DNNGuard can effectively validate the legitimacy of the input samples in parallel with the target DNN model, achieving an average 1.42× speedup compared with the state-of-the-art accelerators.

CCS CONCEPTS

• **Computer systems organization → Neural networks**; **Heterogeneous (hybrid) systems**; • **Security and privacy → Intrusion detection systems**.

KEYWORDS

DNN accelerator, Heterogeneous architecture, Adversarial sample, Detection network

ACM Reference Format:
Xingbin Wang, Rui Hou, Boyan Zhao, Fengkai Yuan, Jun Zhang, Dan Meng, and Xuehai Qian. 2020. DNNGuard: An Elastic Heterogeneous DNN Accelerator Architecture against Adversarial Attacks. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20), March 16–20, 2020, Lausanne, Switzerland.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3373376.3378532

1 INTRODUCTION

Modern artificial intelligence algorithms represented by deep neural networks (DNNs) have greatly promoted the development of

*Corresponding author: Rui Hou(hourui@iie.ac.cn)

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ASPLOS '20, March 16–20, 2020, Lausanne, Switzerland
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7102-5/20/03…$15.00
https://doi.org/10.1145/3373376.3378532

technologies such as image and video comprehension [1, 2], speech recognition [3, 4], and natural language processing [5, 6], enabling important application in critical domains including finance, healthcare and autonomous driving. However, recent studies have shown that by carefully performing minor perturbation to the normal samples, attackers can construct adversary samples that are able to deceive DNN models to make false predictions [7–15]. It can potentially lead to disastrous consequences in security-sensitive application. For example, in autonomous driving, an attacker can interfere the vehicle traffic sign recognition system by constructing an adversary images, so that the normal stop sign can be recognized as as an acceleration or speed limit sign [16]. Attackers can also design adversary samples against speech recognition models to produce voice command to control Apple Siri [17] and Microsoft Cortana [18].

To defend these attacks, the current practice is to use traditional machine learning denoising methods or DNN models, or a combination of both. For example, Feature Squeezing [19] detects adversary samples by median smoothing filter and data bit width compression. MagNet [20] evaluates manifold measurements of images by training a DNN model to classify an input image as adversarial or clean. It includes one or more detectors, the manifold measurements contain two DNN models and a reformer. SafeNet [21] combines two methods to detect adversary samples by applying an additional DNN model and SVM algorithm.

Currently, DNN models are typically deployed in GPUs and domain specific DNN accelerators. Comparing the two, GPU-based solutions provide better performance and versatility, while DNN accelerators achieve better energy efficiency. Thus, GPUs are widely used in parallel model training with high throughput. In contrast, for inference tasks that execute in resource-constraint environments, DNN accelerators are widely and commercially deployed [22, 23]. With the current accelerator architecture, deploying target network with detect algorithm or network poses to key problems.

First, it is *difficult to achieve cost and execution efficiency*. Existing DNN accelerators cannot effectively support the defense methods such as machine learning denoising. Thus, a separate accelerator is typically employed to run detection network model, which significantly increases hardware cost. An alternative option is to reuse the same accelerator and run both target and detection network. However, the current accelerator architecture only allows the serial execution of the two, degrading system performance.

More importantly, deploying detection network naïvely will introduce *security issues*. Deploying the two networks in loosely coupled two accelerators will lead to potential information leakage [24]. Specifically, the data transmission between two accelerators is performed in plain text, which allows the attacker to obtain intermediate data that enable more attacks.

To tackle these challenges, the paper proposes *DNNGuard*, an elastic heterogeneous DNN accelerator architecture that can efficiently orchestrate the *simultaneous execution* of original (target) DNN networks and the defense algorithm or network that detects adversary sample attacks. The DNNGuard architecture is designed with a thorough analysis of the major adversary sample defense methods and deep understanding of the essential requirements on computing resources, on-chip data sharing and task synchronization. To satisfy the requirements, DNNGuard adopts a tightly-coupled heterogeneous architecture with a CPU core and an elastic DNN accelerator to support *diverse* adversary sample defense methods. The elasticity enables target and detection network to run simultaneously in one accelerator; and the CPU core efficiently performs non-DNN computing and the special layer of the neural network. In summary, the paper makes several key contributions as follows.

- An elastic on-chip buffer management mechanism is proposed to fully exploit the data locality—the detection network can efficiently access intermediate data of the target network without off-chip accesses—and guarantee the access order required by the detection mechanism.
- An elastic PE computing resource management is proposed to ensure that detection network executes faster than target network to avoid false prediction by identifying possible attacks timely. Moreover, this management mechanism also maximizes the utilization of computing resources.
- An extended AI instruction set is proposed for the first time (to the best of our knowledge) to support: (1) the synchronization of two networks and the efficient data interaction; (2) the inter-core communication mechanisms that facilitate efficient communication and task scheduling.
- We implement DNNGuard architecture based on RISC-V and NVDLA, and evaluate its performance impacts, hardware cost as well as parameter sensitivity.

The rest of the paper is organized as follows. In Section 2, we analyze existing adversarial sample defense methods and summarize the requirements for accelerator architecture. Section 3 presents the overview of DNNGuard with implementation details explained in Section 4. We evaluate the performance impacts and parameter sensitivity in Section 5 and discuss various design issues in Section 6. The related work is reviewed in Section 7 before concluding the paper in Section 8.

2 BACKGROUND AND MOTIVATION

2.1 Adversarial Sample Detection Mechanisms

Table 1: Summary of Adversarial Sample Defense Methods

Type	Compute Mode	Architecture
DNN	Intermediate feature maps [25–31]	DNN
	Special layers or operators [27–34]	Hash layer, PGD, observed values etc
	Small network [25–42]	DNN (3-10 layers)
	Large network [21]	VGG19/GoogleNet
Non-DNN	Feature squeezing Logistic regression Denoising etc. [42–47]	Decision tree, Random forest, Gaussian etc.
Hybrid	DNN and ML [21] [20, 27–34, 48–56]	DNN + decision tree, SVM, PCA. etc

Table 1 summarizes existing adversarial sample defense methods. In terms of the computational requirements for the deployment of adversarial sample defense methods, the methods are classified into three categories. (1) *Non-DNN (including machine learning) based methods* that detect adversarial samples by denoising or

transforming the input data [19, 42–47]; (2) *DNN based methods* that use DNN network as sub-network model (detection network model) to verify whether the input image is adversarial sample or restore the original image [20, 21, 25–42]. (3) *Hybrid methods* that combine detection network with traditional machine learning algorithm to detect adversarial sample [20, 21, 26, 27, 29–31, 34–37, 49–56]. Besides them, some works propose to retrained target network by generating adversarial samples to improve the robustness of the model [57–59]. Due to its importance, the methods of adversarial sample attacks are continuously evolving.

2.2 Workflow of Detection Mechanisms

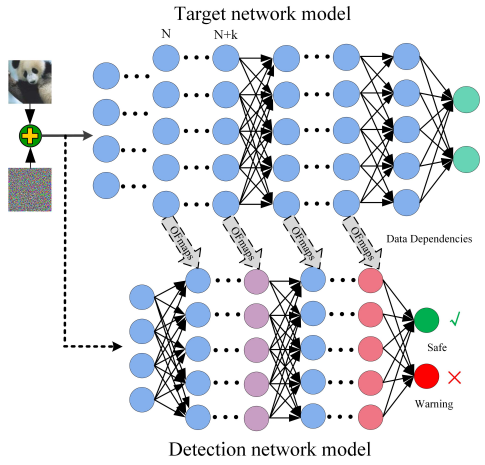

Figure 1: Workflow of Target and Detection Network

Workflow 1: The workflow of target network and detection mechanism.

Input: Adversarial sample, the total layers of detection network (M)
Output: Safe or Warning

```
while current layer L <= M Or detection mechanism running do
    Target network first performs N (N>=1) layer to process input data, and computes output feature maps of the current layer;
    The detection mechanism reads these feature maps as input feature maps to compute;
    Meanwhile, target network computes N+k (k>=1) layer in parallel;
end
return Detection results;
```

Figure 1 shows the typical interactive workflow of target and detection network. The specific implementation process is described in Workflow 1. The detection mechanism can be implemented by the DNN network and/or non-DNN algorithm and should run faster than target network. The detection network might perform one or more layers of calculations. With a few exceptions, the detection mechanism does not access the intermediate data of the target network, only read the input image [37, 39, 40].

2.3 Requirements of Detection Network

Based on the various adversarial sample defense methods summarized in Table 1 and the current accelerator architectures, we believe that the design of detection network has three requirements.

- *High-bandwidth data sharing.* The size of detection networks is usually small [25–42] and they need to frequently access the Output Feature maps (OFmaps) of target network. This implies that a high-bandwidth data sharing should be supported between target and detection network.
- *Elasticity.* When executing target and detection network simultaneously on the same accelerator, the PE computing resources and on-chip buffer resources need to be elastically allocated to meet its performance requirements of the detection network.
- *The need of CPU.* Among various detection mechanisms, some computations are not suitable to execute on existing DNN accelerators, they includes *special computing units* [27–34] such as atypical custom layers (e.g., hash layer) and new activation functions (e.g., convex relaxation); and *traditional machine learning algorithms* (non-DNN) such as SVM, KNN, PCA, decision tree, random forest, image filtering and denoising techniques [42–47], etc. It is best to perform these computations on CPU, which also requires the efficient communication mechanisms between accelerator and CPU core.

In summary, we believe that it is challenging to deploy adversarial sample defense methods in existing DNN accelerators. It is important to develop a new DNN accelerator architecture that can simultaneously execute target network and detect mechanisms. The architecture should be general enough to be deployed with various detect methods, specifically, it should: (1) support data communication and synchronization of target and detection network; (2) effectively perform the special computational operations required by the defense methods; and (3) manage and schedule the computation and on-chip buffer resources needed by target and detection network.

3 DNNGUARD OVERVIEW

3.1 Tightly coupled heterogeneous architecture

To address the challenges discussed above, we propose *DNNGuard*, a heterogeneous architecture which is composed of a CPU core and an elastic DNN accelerator, to efficiently support a variety of adversarial defense algorithms. The overall architecture is shown in Figure 2. Specifically, the CPU core is mainly used for executing the special computing units. To run both target and detection network simultaneously, the elastic DNN accelerator increases the number of processing element (PE) and global on-chip buffer, as well as the nonlinear unit that is dedicated for detection network.

The reason to global buffer is the key to provide efficient data communication. With DNN accelerator and the CPU integrated in one chip, a straightforward way supporting communication between the two is to use the shared off-chip DRAM accessed via bus.

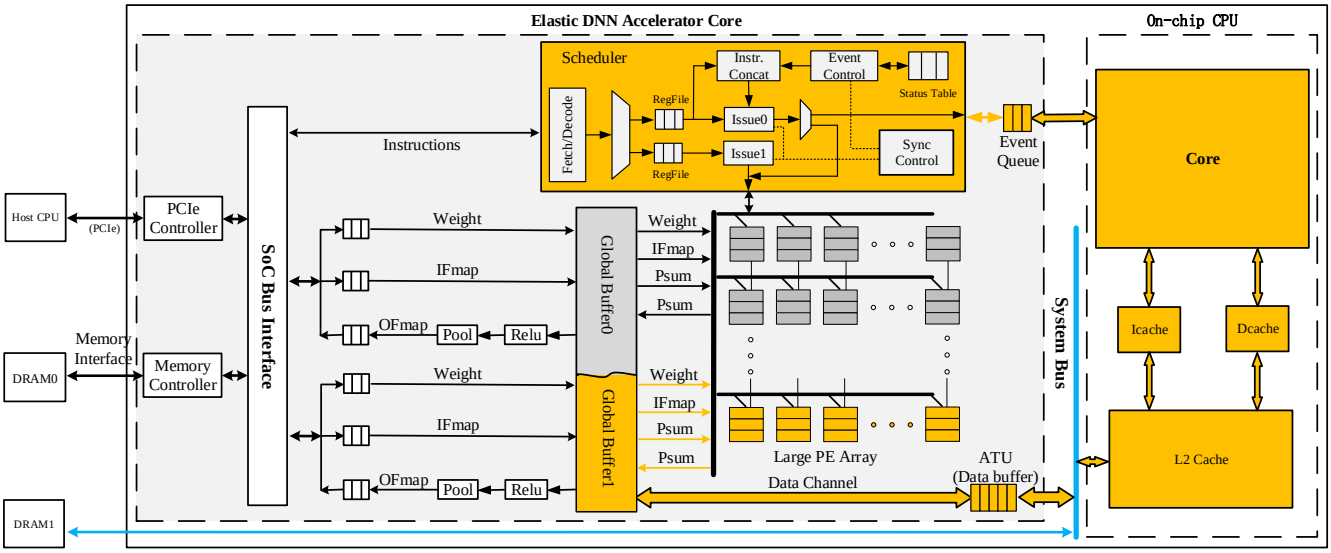

Figure 2: The DNNGuard Architecture Based on Elastic DNN Accelerator and CPU Core

However, it results in significant performance overhead and energy consumption. The read/write performance of global on-chip buffer is over 80× higher than off-chip DRAM. Therefore, in DNNGuard, a global on-chip buffer tightly couples the accelerator and the CPU which enables fast communication between them and within the accelerator.

Based on DNNGuard architecture, we have three network deployment scenarios. (1) Target and detection network execute on DNN accelerator together while the on-chip CPU is idle. Two networks exchange data through the global on-chip buffer inside DNN accelerator. (2) The detection network is divided into two parts, one executing on the DNN accelerator and the other on the on-chip CPU. (3) The target network runs on the DNN accelerator, and the Non-DNN defense methods run on the on-chip CPU. For the second and third case, the accelerator and the CPU communicate with global buffer and Address Translation Unit (ATU).

3.2 Task Data Communication and sharing

Existing DNN accelerators typically adopt the design of *static and one-way* on-chip buffer. For example, input neurons buffers (NBin) and synaptic weights buffers (SB) are only used for PE inputs. The output neurons buffers (NBout) are only used for PE outputs. Unfortunately, such static one-way buffer is inefficient to support data communication between the detection mechanism and target network because the accelerator must put the OFmaps to off-chip DRAM, which will consume more off-chip memory bandwidth.

To address the problem, we propose the *elastic on-chip buffer management mechanism*, which enables the detection mechanism to reuse the shared OFmaps, effectively reducing the data transfer and improving the data communication efficiency between the target network and the detection mechanism. The elastic buffer management mechanism has the following three characteristics.

Improved local data reuse. Compared with the static one-way buffer structure, the elastic buffer management mechanism allows the conversion of the output buffer into an input buffer through the buffer switch to avoid the data movement or copy. It enables detection mechanism to access the shared OFmaps timely with better communication efficiency, which affects the calculation time or delay of DRAM access.

Support for concurrent tasks. Based on the elastic on-chip buffer management mechanism, we design a new on-chip buffer access pattern which can efficiently support two new scenarios. When the on-chip buffer stores the entire OFmaps of target network, the target network and detection mechanism can simultaneously access the shared OFmaps. If not, the detect mechanism takes the priority — target network is suspended until detection mechanism finishes reading its intermediate data.

Enforcing Read-After-Write dependence between tasks. As the shared buffer between target network and detection mechanism, we need to ensure that OFmaps is read by detection mechanism before it is updated by target network. We enforce the RAW dependence by adding empty/full buffer status registers. The hardware scheduler monitors these status registers continuously at runtime and switches the buffer read and write operations to schedule the task execution, thus achieving a correct data access sequence.

3.3 Task-level Synchronization and Scheduling

During the execution, the detection and target network need data coordination and communication, and the former is required to run faster. Moreover, the two networks compete for computing resources and on-chip buffer of the DNN accelerator. Thus, it is necessary to ensure that the detection network can obtain sufficient resources to process the data of the target network in a timely manner. In fact, different target networks have different PE utilization, and even PE utilization of different layers in the same network are not the same.

The target and detection network are usually running on a DNN accelerator. If a layer of the target network over-utilizes PEs, an

additional synchronization and scheduling mechanism is required to ensure that the detection network obtains sufficient resources to process the data of the target network. Otherwise, data will be lost and the detection of the adversarial sample cannot be completed.

Due to the deterministic and sequential nature of the target network and the detection network's processing flow, we do not use a complex handshake mechanism to synchronize and schedule the tasks. Instead, we introduce a *scheduler* within the DNN accelerator and propose an *extended AI instruction set* to dynamically configure the PE and on-chip buffer resources. The instruction set dynamically schedules the PE computing and global on-chip buffer resources. It also cooperates with the event queue to achieve efficient communication and task scheduling between elastic DNN accelerator and the CPU core.

4 IMPLEMENTATION

4.1 Elastic on-chip buffer

For the DNNGuard architecture, the target and detection network need to access the on-chip buffer simultaneously, which leads to new requirements for the design of the on-chip buffer. On the one hand, on-chip buffer needs to support highly concurrent access. On the other hand, it should support efficient data reuse between two networks by dynamic switching. Therefore, we propose an elastic on-chip buffer management mechanism.

4.1.1 Microarchitecture Implementation. As shown in Figure 3, elastic on-chip buffer is essentially a large global buffer, composed of multiple SRAM banks and their corresponding addressing routing networks. It is mainly used to cache feature maps and weights of the two networks. Next we describe each component in detail.

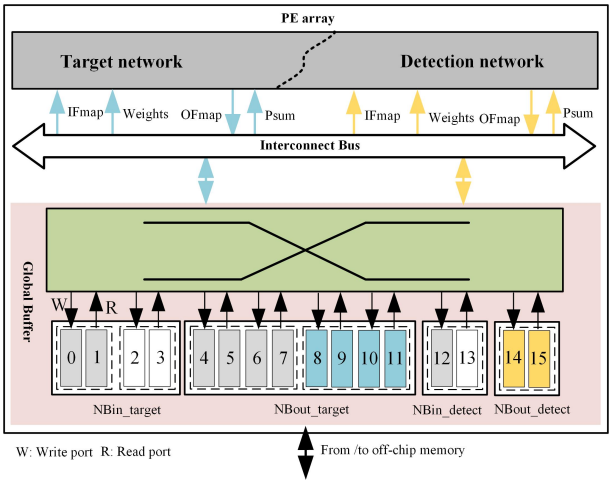

Figure 3: Elastic On-chip Buffer

Multi-bank SRAM routing Each physical bank is uniquely identified by an index (the high address of an SRAM bank). A buffer is formed by having an index array that holds the indices of the physical banks. Multiple SRAM banks are dynamically grouped together as a set to form the buffers (e.g., NBin_Target, NBout_target, SB_target of target network etc.). The granularity of static one-way buffer allocation is at the set level. Since the best bank utilization at the set-level is merely 51% for AlexNet in DNN accelerator [60], the granularity of elastic buffer allocation by DNN complier is at bank level rather than at set level.

The scheduler receives instructions from the host CPU to form the configuration table for on-chip buffer allocation and the routing table for index array switching. According to configuration table and routing table, elastic on-chip buffer management mechanism can set the status of these banks switch dynamically without physically moving/copying data. It also increases bank utilization.

During execution, the sizes of the buffers are dynamically allocated by the target and detection network according to instructions based on the following policies: (1) The capacity of the target network to store the OFmaps should be guaranteed first. (2) The allocation is performed according to the size of OFmaps of target and detection network. For example, input buffer (NBin_detect) of detection network can be small or even omitted. (3) The spatial structure and usage mode of the buffer for target and detection network should remain unchanged to allow the efficient management of the elastic on-chip buffer. The weight buffers (SB_target and SB_detect) work in similar manner, we omit them in Figure 3.

Multi-bank SRAM management Each physical bank contains four status registers (empty, full, write completion, read completion) to show its usage status. The status of the input and output buffer of the target and detection network includes BankWriteFinish, BankWriteDoing, NBinUsing, NBinReadComplete, NBinUnusing etc. The scheduler controls the execution of the two networks by polling these status registers. When the status register shows that target network has completed the calculation of the current layer and written all OFmaps to the NBout_target, the scheduler informs both networks to simultaneously read the shared feature maps as input data for the next layer.

Scheduler The scheduler is responsible for parsing and sending the extended AI instruction set, monitoring the processing status registers and buffer status registers. Furthermore, it handles event queue communication mechanism and task queue management between the accelerator and CPU core (e.g., receiving, decoding, sending and address translation unit, etc.).

4.1.2 Typical scenarios. In this example, layer i-1 of target network produces four OFmaps which become the four IFmaps of layer i ❶. Then layer i produces eight OFmaps. Assume that NBin_target and NBout_target have two banks and four banks respectively. If each bank can store one feature map, this setting means that, in one iteration, the accelerator reads two IFmaps and computes the convolution results for four partial OFmaps 0~3. Two iterations are needed, the accelerator needs to load all the four IFmaps to compute the four final OFmaps of layer i. In this way, another two iterations are needed to produce OFmaps 4~7 ❷. The detection network is omitted since it works in a similar manner.

On-chip buffer stores all OFmaps of the target network As NBout_target buffer can store all IFmaps of target network, target network and detection network can read the shared feature maps at the same time. There is no need to guarantee the sequence between the two, which does not affect the execution of target network. However, it is necessary to ensure that the detection network can consume these feature maps in time.

On-chip buffer only stores partial OFmaps of the target network Suppose that the target network completes OFmaps 0~3 of layer i and writes them into the output buffer NBout_target. At this time, the target network needs to wait for the detection network to read these feature maps. The scheduler senses whether the detection network has read OFmaps 0~3 by polling status register. Then it informs target network to move the data to the off-chip DRAM. Meanwhile, IFmaps are read from off-chip DRAM into the input buffer NBin_target. The remaining OFmaps 4~7 will be processed in a similar way.

4.2 Elastic PE Resource Management

Existing PE design is composed of Multiply-and-Accumulate (MAC) and Convolution Accumulator (CACC) with a single-channel design. Naturally, this structure is unable to support dual-model input-output or multi-model input-output. Also, existing PE structure does not support dynamical allocation for target and detection network. Therefore, we add three modules to the existing PE structure to support elastic allocation of computing resources as shown in Figure 4. The specific descriptions of each module are as follows:

- MAC Switch (MS) unit: An MS unit is added to the input port of PE to select the activation value or weight of the input of a certain network;
- Adder Switch (AS) unit: An AS unit is added to the output port of PE to select the computational output or partial sum of a certain network;
- Routing Logic unit: It guarantees the consistency of data input and output routes belonging to the same network, so that the target and detection network can be executed as two threads.

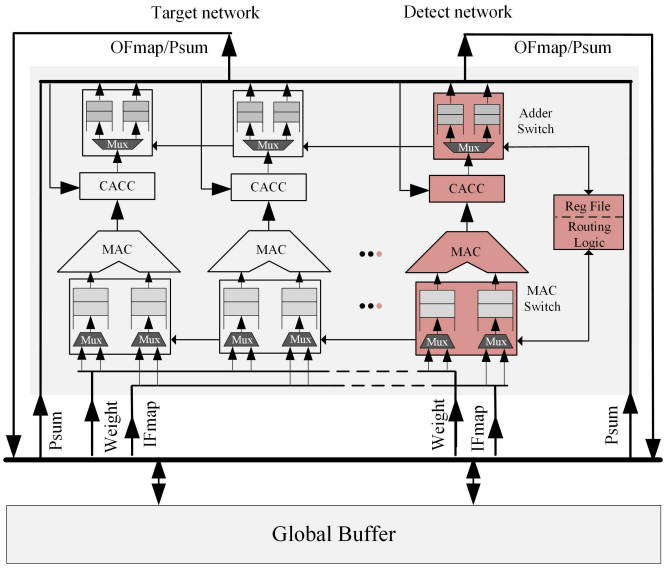

Figure 4: DNN Accelerator with Elastic PE Structure

We also implement a private buffer in the MS unit and the AS unit, which can be used to efficiently forward data. There is a corresponding register for each PE unit set by the configure instruction. The input path selection of the partial sum is also controlled by the Routing Logic unit.

A number of factors need to be considered when allocating PE resources, including the overall PE utilization of the different networks and layers, the size of the on-chip buffer required by the middle layer feature map (the size of OFM, IFM, Weight), and the calculation and communication ratio. To optimize execution performance of the two networks, we need to exploit the trade-off between the number of PEs and the on-chip buffer size of the target and detection network. We use a greedy algorithm to allocate the PE and buffer resources, and achieve the data synchronization between the two networks through dynamic scheduling of the shared PE and the global on-chip buffer.

4.3 Compiler Design

Analytical model: In order to minimize off-chip accesses and maximize performance of target and detection network, the compiler performs co-optimization of both accelerator architecture and corresponding scheduling to determine the hardware parameters of the two networks. We design a heuristic resource optimization search algorithm to optimize the scheduling and resource partitioning. Its inputs include: the dataflow graph, data dependency of the two DNN models and DNNGuand hardware parameters (i.e., PE number, on-chip cache size, DRAM bandwidth, etc.). The algorithm outputs the number of PEs, and the on-chip buffer size required by each layer of the target network and detection network by taking the four steps:

- **Initialization:** initialize the number of PEs and the on-chip buffer size for each layer of the target and detection network, the estimated execution cycles of two models, etc;
- **Increment:** a greedy algorithm is used to search the proper on-chip buffer size by varying the number of PEs iteratively starting from proper value to the maximum number of PEs on the DNNGuard;
- **Calculation and estimation:** the execution cycle is evaluated in the simulator based on the allocated resources;
- **Reiterate/termination:** if the estimated execution cycle is less than the initial value or execution cycle, record the choices; terminate if the number of PEs has reached the maximum value, otherwise, reiterate from step **Increment**.

In addition, in order to reduce the search space of the number of PE and the on-chip buffer size, we set empirical values proportionally according to the size of target and detection network.

Programming model: We use the standard DNN framework format (Caffe in the current implementation) to describe the target and detection network, and the data dependencies between them are represented by the data link structures. Each node of the list contains the following information: Dataflow_direction, Target_layer_number, Target_FP_channel_start, Target_FP_channel_end, Detection_layer_number, Detection_FP_channel_start, Detection_FP_channel_end. DNNGuard uses the compiler to generate execution instructions of the two networks based on the extended instruction set, as well as the configuration instructions of the computing resources and on-chip buffer. At runtime, the host CPU configures the instruction registers on the accelerator through the instruction interface (CSB interface).

Table 2: Extended AI Instruction Set

Type	Operand Specification	Description
Configuration Panel	Cfg_Resource_Allocate	*Scheduler dynamically allocates PE and on-chip buffer for target network and detection network.*
	Cfg_Trig	*The instruction triggers detection task after scheduler prepares the relevant instructions (Event Dispatch).*
	Cfg_Clear	*Scheduler resets the process status register and buffer status register of target and detection network.*
Data Panel	GBuffer_To_PE_Load	*The DMA loads OFmaps of target network into the local buffer of PE from global buffer.*
	PE_To_GBuffer_Store	*The DMA stores PE processing result into global buffer from the local buffer of PE.*
	GBuffer_To_CPU_Load	*The DMA loads OFmaps of detection network into data buffer from global buffer.*
	CPU_To_GBuffer_Store	*The DMA stores CPU processing result into data buffer from the cache of CPU.*
Control Panel	Control_Event_Dispatch	*Detection mechanism processes special operations as events. Event type: hash layer (0x01), SVM (0x02), PCA (0x03) etc. The instruction packages event command (Event ID etc.)*
	Control_Sync	*The instruction is not only used for the synchronization between DNN accelerators and CPU but also the synchronization between the target and detection network. It can be generated by the compiler and the scheduler.*
	Control_Polling_Check	*Scheduler checks process status register and buffer status register of target and detection network. If the value is 1, detection mechanism executes and resets status register. Otherwise scheduler always polls these status registers.*

4.4 Orchestrating Communication with Extended AI Instruction Set

4.4.1 Communication Between Target and Detection Network. During their execution, the target and detection need to schedule and allocate the computation and on-chip buffer resources. The detection network requires a large amount of intermediate data produced by target network. Moreover, the execution processes of the two networks need to be synchronously controlled. The extended AI instruction set includes three categories of instructions, as shown in Table 2, to meet diverse communication and data processing requirements. The *Control_Event_Dispatch* instruction and *Control_Sync* instruction are designed to coordinate the communication between the two networks, as well as the communication between elastic DNN accelerator and CPU. During data communication, the scheduler controls the computation flow by monitoring the status register. We add a *Control_Polling_Check* instruction to query the status register on the DNN accelerator to obtain the status of event processing (including TaskStart, TaskDoing, TaskSuspension, TaskDone), target and detection network processing status (including ChannelProcessStart, ChannelProcessDoing, ChannelProcessDone, etc.), as well as the read and write status of on-chip buffer. Furthermore, CPU core which is processing special computation needs to set a separate thread to query the event queue, parse the received command packets accordingly, and send the results and status of the processing to the scheduler.

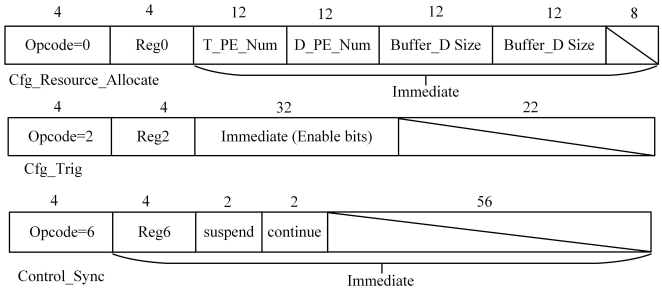

Figure 5: Extended AI Instruction Format

The typical format of extended AI instruction set is shown in Figure 5. It is a 64-bit instruction consisting of the operating code, the destination register, the source register, and immediate value.

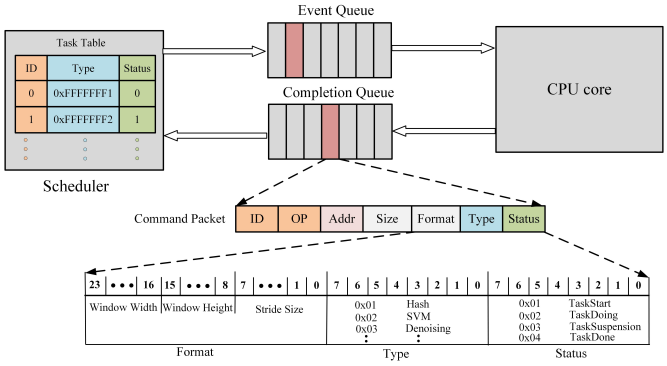

Figure 6: Event Queue Communication Mechanism

The sixteen 64-bit general registers are designed to support extended AI instruction sets. They are similar to other typical types of instruction sets for neural network[61, 62]. Bitfusion [63] and NVDLA is 32-bit instruction which is the subset of the extended AI instruction.

4.4.2 Communication Within Detection Network. In order to enable efficient communication between elastic DNN accelerator and CPU core, we implement efficient data communication and task scheduling based on global on-chip buffer and event queue communication mechanism. Meanwhile, address translation unit is used to support CPU to read and write on the on-chip buffer of elastic DNN accelerator.

Event queue communication mechanism. It is designed to immediately respond to the accelerator request. As shown in Figure 6, the scheduler takes and packages the relevant instructions for event communication (including the task start and event ID) and sends the command package to the event queue (FIFO). CPU core processes special computation and sends the results and status of the processing to the scheduler.

4.5 Putting All Together: A Running Example

Figure 7 shows the execution process of a typical detection network [31] on the DNN accelerator and CPU. Two typical timing windows are selected to demonstrate the usage of the extended AI instruction set.

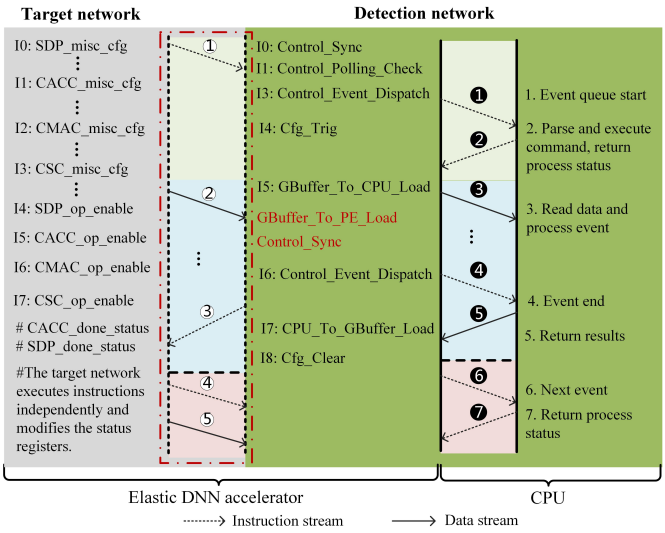

Figure 7: A Running Example of DNNGuard

Timing window 1: The target and detection network conduct data transmission and synchronization on the elastic DNN accelerator, as shown in the red box of Figure 7. The scheduler queries the status registers of target and detection network through *Control_Polling_Check* instruction. It uses the *GBuffer_To_PE_Load* instruction to read the required feature maps, and uses *Control_Sync* to achieve data synchronization between the two networks.

Timing window 2: The detection network executes *Control_Sync* instruction after the calculation of current layer. The scheduler transfers the output data to CPU core for special computation processing through the *Control_Event_Dispatch* instruction and event queue communication mechanism. The CPU reads the corresponding data according to the command packet instructions to complete a specific computation (state information needs to be returned to the scheduler during CPU processing, such as TaskDoing), and writes the processed data back to the on-chip buffer of the accelerator.

5 EVALUATION

5.1 Methodology

Accelerator Implementation. NVDLA [64–67] and RISC-V [68] are employed to implement the accelerator of DNNGuard in Figure 8. The scheduler and logic fabric are designed in Verilog language and the corresponding hardware is synthesized by using Synopsis Design Compiler with SMIC 65nm standard-cell library. The elastic NVDLA is simulated and verified with Synopsys Verilog Compile Simulator (VCS), and the power consumption is estimated with Synopsys Prime-Time PX according to the simulated Value Change Dump (VCD) file.

Simulator. We implement a customized cycle-accurate timing simulator to evaluate the accelerator fabric, integrated with the CACTI [69] to model DRAM access latency. The simulator's input file contains the micro-architectural parameters such as the number of PE, on-chip buffer size, maximum memory bandwidth

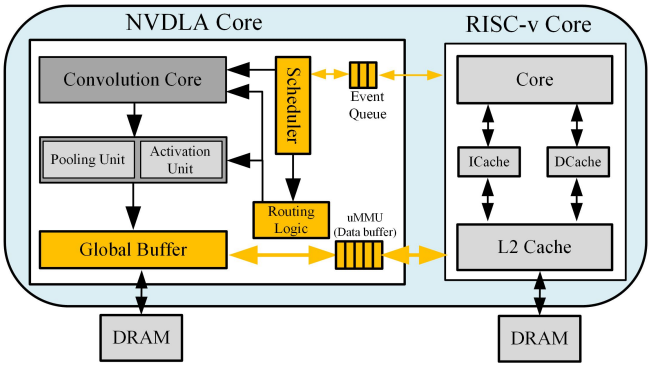

Figure 8: Elastic NVDLA and RISC-V implement DNNGuard architecture

Table 3: Evaluation Platform Configuration

NVDLA	
Frequency	1.0 GHz
MACs	**1024** + 128
Buffer (KB)	**256** + 128
Memory BW	10 GB/s
RISC-V	
Frequency	1.0 GHz
Processor type	4-way out-of-order
Pipeline	15 stages
Commit	Up to 4 instructions/cycle
L1 DCache	32KB, 8-way, 64B line
LLC Cache	Shared 2MB, 16-way, 64B block
DRAM	8 GB DDR3, 192 cycle latency

and the description of each layer of DNN networks. The output results include SRAM accesses, DRAM accesses, DRAM bandwidth requirement, the size of OFmapIFmap, weight of target network and adversarial network. It also generates computation and communication behaviors, and the PE utilization of each layer of DNN network. Furthermore, we design a compiler to generate extended AI instructions for target network and detection network based on these parameters. The average NVDLA's PE utilization for each target network is 64.3%, and the size of the detection network model is usually small (3~10 layers). According to the performance parameters generated by the simulator, PE computing resources and on-chip buffer resources increase by 12.5% and 50% respectively, compared to NVDLA.

Workloads. We use several state-of-the-art target DNNs as the benchmarks to evaluate DNNGuard, as it is summarized in Table 4. The selected DNNs are used for various applications ranging from image comprehension, object recognition and natural language processing, and they are popular medium to large scale with dense DNN workloads with several hundreds of Mbytes memory footprints. We also select three typical detection networks that

Table 4: Target and Detection DNN Workloads

Network	Type	Layers	Max IFM	Max OFM	Total MAC	Total Weights
TextCNN [5]	Target	6	0.12MB	0.16MB	9.26M	0.6M
TinyYOLO-v2 [70]	Target	9	0.66MB	2.64MB	3.13G	8.35M
AlexNet [71]	Target	8	0.31MB	0.57MB	0.724B	61M
VGG16 [72]	Target	16	6.24MB	6.27MB	15.5B	138M
GoogleNet [73]	Target	22	0.39MB	1.52MB	1.43B	7M
ResNet50 [74]	Target	50	1.57MB	1.57MB	3.9B	25.6M
AN [25]	Detect	6	0.17MB	0.34MB	531M	0.91M
NIC [31]	Detect	8	0.26MB	0.86MB	761M	1.3M
PuVAE [40]	Detect	10	0.26MB	0.86MB	1.1G	1.8M

B: Billion, M: Million

cover different data flows (Serial and Parallel) and computing platforms (Elastic DNN accelerator and CPU). Furthermore, experimental evaluation on the quantization of detection network demonstrates that the detection rate of adversarial sample is insignificant.

Baselines. To make fair comparison, we re-implement NVDLA called Source-NVDLA (SNVDLA) which combines the large NVDLA and the small NVDLA together to run the target network and the adversarial network respectively. The elastic NVDLA (ENVDLA) has an equal amount of PE and on-chip buffer as the SNVDLA. Table 3 shows the evaluation platform configuration.

5.2 Results on DNNGuard Architecture

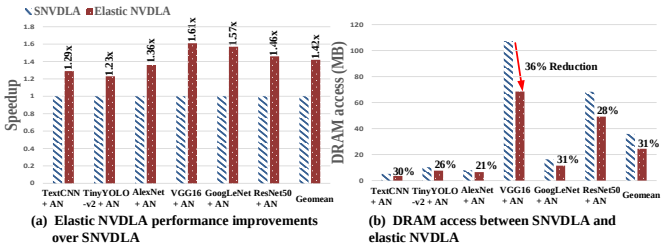

Figure 9: The Performance Comparison of Elastic NVDLA and SNVDLA

Performance. We compare elastic NVDLA against the SNVDLA on typical detection network (AN [25]) and six target networks listed in Table 4. Figure 9 (a) shows the performance speedup of the elastic NVDLA. Compared with the fixed allocation of the target and detection network, the elastic NVDLA use synchronization and configuration instructions to dynamically adjust the resource allocation for better performance. On average, the elastic NVDLA achieves around 1.42× speedup over SNVDLA. Specifically, dynamic resource allocation results in the performance improvement of about 18.4%. Meanwhile, data communication improves the performance of the elastic NVDLA by 32.1%. The results make sense because the elastic NVDLA integrates dedicated functional units and elastic resources management optimized for DNN techniques. The extension AI instruction also improves the utilization of on-chip buffer and PE resources, which is the major reason for the elastic NVDLA performance improvement.

As shown in Figure 9 (b), the intermediate data may be stored in the off-chip DRAM due to the large feature maps of VGG16 (the largest feature map is 6.27MB). The elastic NVDLA uses the elastic on-chip buffer management mechanism, in combination with data communication and control instructions, to reduce DRAM traffic by 36.2%, thus improving VGG16 processing performance by 1.61×. Moreover, to enable detection network to consume the intermediate data in a timely manner, the scheduler must allocate more resources to solve the storage bottleneck caused by the low computing power of the detection network.

We also analyze on the usage of the extended AI instruction set in each network, as shown in Figure 11(a). Although communication instruction and synchronization instruction take up a small proportion, they play a key role in the cooperative execution of target and detection network. The *Cfg_Clear* instruction and *Control_Polling_Check* instruction have the highest use frequency, mainly because the scheduler needs to control the process state register and on-chip buffer state register. Since there is no special computing operation on this detection network, the *Control_Event_Dispatch* instruction is not used. The extended AI instruction set accounts for average 2.3% of the total instructions. For a fair comparison, we set the execution cycle of extended AI instruction same as the instruction execution cycle of NVDLA which is 18 cycles.

Table 5: Source NVDLA and Elastic NVDLA Design Parameters

Accelerator	Area (um^2)	Power (mW)	Combinational Cell	Sequential Cell
Source NVDLA	5118978	737.305	947405	71638
Elastic NVDLA	5361105	861.909	1024144	75454
Overhead	4.7%	16.9%	8.1%	5.3%

Area and power. We list the area and power parameters in Table 5. The overall area of the elastic NVDLA is 5.1 mm^2, which is about 4.7% larger than that of SNVDLA. The combinational logic (mainly scheduler, complex interconnect and control unit, routing logic .etc) consumes 1.02% area of elastic NVDLA. The nonlinear activation unit and pool unit also consume about 3.7% of the area. The power consumption of the ENVDLA increases 17% compared to SNVDLA. More specifically, the scheduler, elastic on-chip buffer management mechanism and elastic PE resource management mechanism consume about 9.9% of the power. The nonlinear activation unit and pool unit also consume about 7% of the power.

5.3 Detection Mechanism on DNNGuard

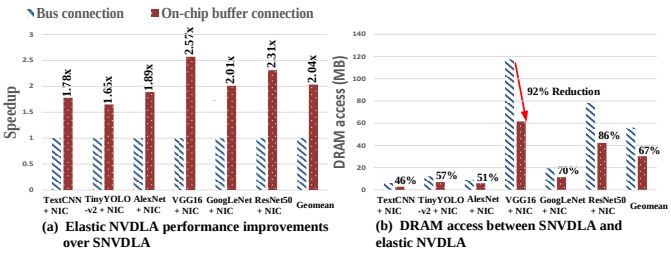

Figure 10: The performance improvement of tightly coupled DNN accelerator architecture

We connect SNVDLA and RISC-V via system bus. Figure 10(a) shows the performance improvement of the tightly coupled architecture. Compared with the traditional bus connection's data transfer, the on-chip buffer connection has better communication efficiency. The latter has an average performance improvement of about 2.04×. Figure 10 (b) shows that the average reduction of the off-chip data traffic is 67%.

We find that detection network NIC (including SVM algorithm) running on elastic DNN accelerator and CPU can cause performance loss for smaller target network models. More specifically, the execution time of the SVM algorithm on RISC-V is about 3.8ms. However, the running time of Alexnet, GoogLeNet with the detection network without considering data dependence of SVM executed by CPU is 0.97ms and 1.17ms respectively. Thus DNN accelerator must execute *Control_Sync* instruction to wait until CPU finishes its computation. In order to improve the performance of the detection network, the scheduler allocates more computing and storage resources to detection network which will cause the performance degradation of the target network. Due to the target network waiting time's reduction, the execution time of the NIC can be reduced by 6.3% and 8.5% when co-located with AlexNet and GoogLeNet. Similarly, the longest running time of ResNet50 is about 9.05ms. At this point, the NIC runtime (1.12ms + 3.8ms) can meet the performance requirements of the target network. Certainly, the performance of the target network can be maintained by using sufficient computational power to accelerate SVM.

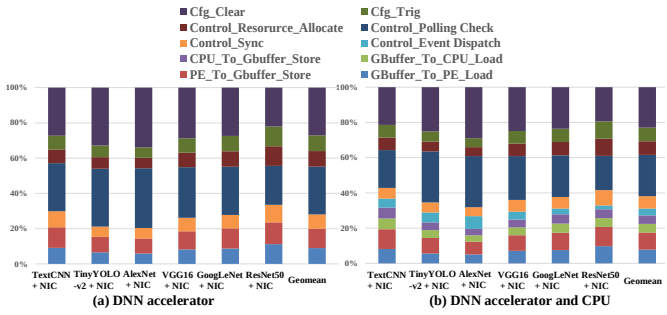

Figure 11: The Percentage of Different Instructions

Moreover, we collect the percentage breakdown of extension AI instruction types in the six benchmarks in Figure 11(b). On average, 33.8% instructions are configuration instructions, 23.7% instructions are data instructions, 42.5% instructions are control instructions. This observation clearly shows that control instructions play a critical role in the communication between the elastic accelerator and CPU. Thus efficient implementations of these instructions are essential to improve the performance of the DNNGuard.

Meanwhile, the computational complexity of the detection functionality is closely related to the scheduling and management of on-chip buffer and PE resources. The detection functionality (NIC) of Figure 11(b) has a higher computational complexity than the detection network (AN) of Figure 11(a), therefore the executing of NIC requires more control instructions and data instructions.

5.4 Non-DNN Defence Methods on CPU

We evaluated several typical non-DNN adversarial sample defense methods on RISC-V. Typical non-DNN adversarial sample defense methods are shown below: PCA [27, 75], SVM [21, 31, 76], Logistic Regression [35, 42], Decision Tree [77], Random Forest [47], Feature Squeezing [19]. Figure 12 shows that these defense methods can meet the performance requirements of larger target network models such as ResNet50 and VGG16, but they cannot meet performance requirements of smaller target network models such as AlexNet. In addition, the elastic DNN accelerator uses all the compute resources and on-chip buffer to accelerate the target network to improve their performance.

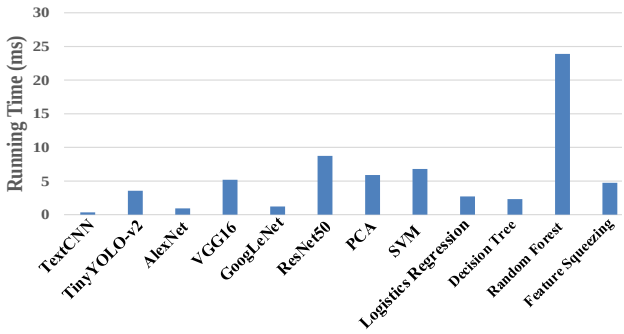

Figure 12: The Performance of Non-DNN Defence on CPU

For small target networks such as AlexNet and TextCNN, these defense methods cannot meet the requirements for maximum performance. However, the single-frame running time of RseNet50 is 8.73ms, which meets the requirement for adversarial sample detection. For Random Forest, execution time is too long to meet the performance requirements of all target networks (the maximum performance that accelerator can achieve).

5.5 Sensitivity Analysis

Performance impacts of the number of PE. Figure 13(a) shows the sensitivity, using the ratio of frame per second (FPS). While the number of PE increases from 64 to 2048, the computational performance does not increase linearly. Furthermore, PE utilization decreases with the increase of PE number. The increase in area, power consumption and cost is disproportionate to the increase in performance, so the elastic DNN accelerator does not need too many PEs. For example, compared with an accelerator with 64 PEs, the ResNet50 has a maximum performance increase of 6.33x, while the computing resource increases by 32×. According to our experiment results, PE utilization is close to 100% when the number of PEs is in the range of 64~256, and gradually decreases since then. In addition, we tested SafeNet [21], the largest detection network (implemented primarily with VGG16), and we increased the DNN accelerator's PE by 43%, achieving the same performance as the target network.

Performance Impacts of Buffer Capacity. As shown in Figure 13 (b), increasing the size of the on-chip buffer does not significantly improve the performance of the detection network, especially for small computation-intensive networks. The performance

improvement is mainly limited by the DRAM bandwidth. AlexNet and GoogLeNet achieved a maximum performance increase of 14.3% compared to 64KB on-chip buffer. However, VGG16 and ResNet50 have a maximum performance improvement of 2.36×, mainly due to the increased on-chip buffer reducing data traffic between the DNN accelerator and off-chip DRAM.

Performance Impacts of DRAM Bandwidth. The relationship between DRAM bandwidth and the performance of the two networks is shown in Figure 13(c). Experimental results show that AlexNet and VGG16 have higher sensitivity to DRAM memory due to heavy-weight full connection layers. After the DRAM bandwidth is increased to 15 GB/s, the increase in bandwidth does not significantly improve performance.

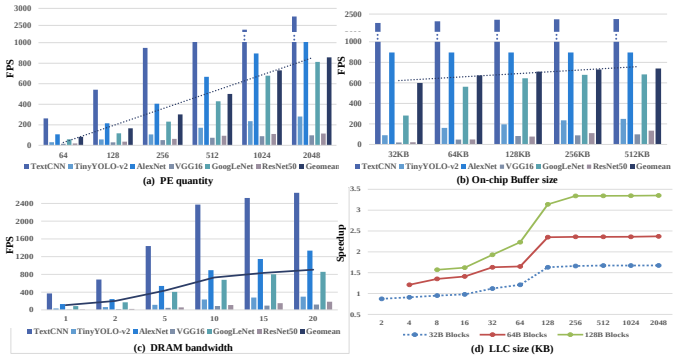

Figure 13: Sensitivity Analysis

Performance Impacts of LLC size of CPU. The performance of typical adversarial sample defense methods (SVM [21], Logistic Regression [42, 43], Decision Tree [77], Random Forest [47]) on RISC-V is measured by changing the size of LLC, and the geometric average value of their performance improvement is calculated. We set the LLC size to 0 as the baseline. Figure 13 (d) shows the performance improvement of these defense methods. Larger cache blocks and larger LLC sizes can result in significant performance improvements. For a cache block size of 128 bytes, increasing the size of LLC can significantly improve performance (up to 3.34x). However, when LLC size is larger than 256KB, the performance of these defense methods does not increase correspondingly. As shown in the figure, the performance of these defense methods is most sensitive to LLC at 64KB. It is mainly because the L1 cache size is set to 64KB, which allows the processor to prefetch the data used by these defense methods to improve performance.

6 DISCUSSION

Robust target network. The robust target network model as list in Table 4, which resists the attacks of adversarial samples, could be divided into two parts. Thus, the elastic DNN accelerator can process the target network in a similar way using two threads in parallel. In this case, the global on-chip buffer is used for the data exchange between the two parts. The experiment results show that the average performance is improved by 1.54×.

Serial relationship between the target and detection network. PuVAE [40] is a typical serial processing method. As the execution time of detection network is much smaller than that of target networks with large scale, such as VGG16 and ResNet50, the performance overhead is 4%~14%. The execution time of some target network, such as AlexNet and GoogLeNet, is close to the detection network because they have similar network scale. Consequently, the serial processing method incurs 34%~61% performance overhead for the small scale target network.

In addition, there is no accelerator architecture that can convert the serial execution of two networks into parallel execution. Therefore, we perform the two networks in a serial manner for experimental evaluation. Compared with SNVDLA, ENVDLA (in serial) improves the performance of AlexNet and VGG16 by 1.18× and 1.27× with the detection network (PuVAE).

Adaptability to future algorithms. Our architecture can support all the current adversarial sample defense methods, such as GAN [78]. The performance overhead is mainly attributed to the deconvolution operation, which is executed on the CPU. Fortunately, the NVDLA supports deconvolution operation. This could significantly improve the performance for GAN's defense method. Our architecture can also adapt to the evolution of defense algorithm, and support their various computing types as well as multiple work flows.

Compatibility with the current DNN accelerator. The architecture of DNN accelerator includes sparse architecture, data flow architecture, prediction architecture etc. The difference between these architectures is the design of the PE structure and on-chip buffer. Although extended AI instruction set can be well integrated into their architecture to support the execution of target network and detection network, unified interfaces for PE and on-chip buffer are still needed to guarantee our architecture's compatibility.

For typical DNN accelerator, we implemented a "dual-Eyeriss" setting where two Eyeriss [23] architectures are used to deploy the target network and the detection network (AN [25]), respectively. The performance of dual-Eyeriss accelerators is evaluated on Systolic Array Simulator (SCALE-SIM) [79]. Under the premise that ENVDLA and dual-Eyeriss accelerators have the same performance, the dual-Eyeriss accelerators consume an average of 43.1% more computing resources and 35.7% more storage resources than ENVDLA.

For sparse DNN accelerators, a more complex scheduling mechanism is needed. The uncertainty of network execution time will result in more handshake signals. To solve this problem, a complex handshake mechanism needs to be be adopted for data communication between the two models. These will be our future works.

Security Analysis. In order to cope with the various attacks of DNN accelerators [24, 80], We design architecture support for security chip to reduce the attack surfaces [81]. Our DNNGuard architecture enables the target network and detection network to effectively reuse OFmaps and reduce the memory side-channel information leakage [80]. Compared with the deployment scheme of connecting two DNN accelerators through PCIe interface, tightly coupled design of DNNGuard architecture can effectively avoid side-channel information leakage of data interface.

7 RELATED WORK

Instruction sets of DNN accelerator. There is an increasing interests on data flow and architecture design of DNN accelerators. One trend is ASIC or FPGA acceleration method without instruction set architecture [23, 60, 82, 83], and another is the implementation of AI instruction set used for accelerating various DNNs [61–63, 84]. However, the design of these instruction sets is aimed at the operational efficiency or flexibility for DNN. Our work is mainly focused on optimizing the task and data communication between the two networks. Our DNNGuard architecture equipped with extended AI instruction set allows a hardware accelerator to detect adversarial samples.

Adversarial sample attack and defense algorithms. Recently, Cheng *et al.* [85] propose a prior-guided random gradient-free method to improve black-box adversarial attacks. Duan , et al. [86] is able to craft "real" adversarial images through derivative-free search on the discrete integer domain. They make many adversarial sample defense methods invalid. Meanwhile, there are many adversarial sample defense methods to deal with adversarial attack [51, 55, 87]. These adversarial defense methods can be deployed by our architecture. Specifically, Li *et al.* [51] can detect and defend adversarial attacks by correcting logits which need to be computed with DNN accelerator and CPU.

Hardware supports for detecting adversarial sample. DeepFense [75] is the first end-to-end automated framework that has the ability to detect malicious inputs online and validate the legitimacy of input samples in parallel with the victim DNN model. Moreover, it leverages hardware/software/algorithm co-design and customized acceleration to achieve just-in-time performance in resource constrained settings. Our work provides a hardware accelerated architecture for online defense against adversarial inputs. Moreover, it has the ability to adapt to the evolution of adversarial sample defense methods in future.

In order to make a fair comparison, we implement the DeepFense with eight defenders in parallel on DNNGuard (The running clock is 150MHz). Experimental results demonstrate that DNNGuard delivers 1.21× and 1.13× improvement over DeepFense for MNIST and SVHN, respectively. DeepFense consists of multiple parallel neural networks and a non-DNN model, which is a typical deployment scenario for DNNGuard. Our architecture can make full use of intermediate data of the target network for DeepFense to reduce data movement. Although the overall performance of DNNGuard for online defense against adversarial sample is slightly better than that of DeepFense, the dedicated hardware architecture for DeepFense with multi-models defenders (more than ten models) performs better than DNNGuard. This is mainly due to DNNGuard's inability to provide the parallelism of a dedicated architecture, as well as the sufficient Non-DNN computing power.

8 CONCLUSION

This paper proposes *DNNGuard*, an elastic heterogeneous DNN accelerator architecture that can efficiently orchestrate the simultaneous execution of original (*target*) DNN networks and the *detect* algorithm or network that detects adversary sample attacks. The architecture tightly couples the DNN accelerator with the CPU core into one chip for efficient data transfer and information protection. An elastic DNN accelerator is designed to run the target network and detection network simultaneously. Besides the capability to execute two networks at the same time, DNNGuard also supports the non-DNN computing and allows the special layer of the neural network to be effectively supported by the CPU core. To reduce off-chip traffic and improve resources utilization, we propose a dynamical resource scheduling mechanism. To build a general implementation framework, we propose an *extended AI instruction set* for neural networks synchronization, task scheduling and efficient data interaction.

9 ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments and suggestions. This work was supported by the Strategic Priority Research Program of Chinese Academy of Sciences under grant No. XDC02010000. This work is also supported by National Science Foundation (Grant No. CCF-1750656, CCF-1919289).

REFERENCES

[1] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6077–6086, 2018.

[2] Will Norcliffe-Brown, Stathis Vafeias, and Sarah Parisot. Learning conditioned graph structures for interpretable visual question answering. In *Advances in Neural Information Processing Systems*, pages 8334–8343, 2018.

[3] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.

[4] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

[5] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[7] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[8] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[9] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[10] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387. IEEE, 2016.

[11] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 2019.

[12] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.

[13] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[14] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.

[15] Shuyu Cheng, Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. Improving black-box adversarial attacks with a transfer-based prior. *arXiv preprint arXiv:1906.06919*, 2019.

[16] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[17] Apple. ios-siri-apple. https://www.apple.com/ios/siri/, 2016.

[18] Microsoft. Cortana-your intelligent virtual and personal assistant - microsoft. https://www.microsoft.com/en-us/windows/cortana, 2016.

[19] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *network and distributed system security symposium*, 2018.

[20] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 135–147. ACM, 2017.

[21] Jiajun Lu, Theerasit Issaranon, and David Forsyth. Safetynet: Detecting and rejecting adversarial examples robustly. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 446–454, 2017.

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

[23] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2016.

[24] Xing Hu, Ling Liang, Lei Deng, Shuangchen Li, Xinfeng Xie, Yu Ji, Yufei Ding, Chang Liu, Timothy Sherwood, and Yuan Xie. Neural network model extraction attacks in edge devices by hearing architectural hints. *arXiv preprint arXiv:1903.03916*, 2019.

[25] Akram Erraqabi, Aristide Baratin, Yoshua Bengio, and Simon Lacoste-Julien. A3t: Adversarially augmented adversarial training. *arXiv preprint arXiv:1801.04055*, 2018.

[26] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267*, 2017.

[27] Xin Li and Fuxin Li. Adversarial examples detection in deep networks with convolutional filter statistics. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5764–5772, 2017.

[28] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. *International Conference on Learning Representations*, 2018.

[29] Fangzhou Liao, Ming Liang, Yinpeng Dong, Tianyu Pang, Xiaolin Hu, and Jun Zhu. Defense against adversarial attacks using high-level representation guided denoiser. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1778–1787, 2018.

[30] Li Chen, Hailun Ding, Qi Li, Jiawei Zhu, Haozhe Huang, Yifan Chang, and Haifeng Li. Adversarial feature genome: a data driven adversarial examples recognition method. *arXiv preprint arXiv:1812.10085*, 2018.

[31] Shiqing Ma, Yingqi Liu, Guanhong Tao, Wen-Chuan Lee, and Xiangyu Zhang. Nic: Detecting adversarial samples with neural network invariant checking. In *NDSS*, 2019.

[32] Deqiang Li, Ramesh Baral, Tao Li, Han Wang, Qianmu Li, and Shouhuai Xu. Hashtran-dnn: A framework for enhancing robustness of deep neural networks against adversarial malware samples. *arXiv preprint arXiv:1809.06498*, 2018.

[33] Jingyi Wang, Guoliang Dong, Jun Sun, Xinyu Wang, and Peixin Zhang. Adversarial sample detection for deep neural network through model mutation testing. In *Proceedings of the 41st International Conference on Software Engineering*, pages 1245–1256. IEEE Press, 2019.

[34] Eric Wong and J Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[35] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[36] Shangxi Wu, Jitao Sang, Kaiyuan Xu, Jiaming Zhang, Yanfeng Sun, Liping Jing, and Jian Yu. Attention, please! adversarial defense via attention rectification and preservation. *arXiv preprint arXiv:1811.09831*, 2018.

[37] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. *International Conference on Learning Representations*, 2018.

[38] Guanhong Tao, Shiqing Ma, Yingqi Liu, and Xiangyu Zhang. Attacks meet interpretability: Attribute-steered detection of adversarial samples. In *Advances in Neural Information Processing Systems*, pages 7717–7728, 2018.

[39] John Bradshaw, Alexander G de G Matthews, and Zoubin Ghahramani. Adversarial examples, uncertainty, and transfer testing robustness in gaussian process hybrid deep networks. *arXiv preprint arXiv:1707.02476*, 2017.

[40] Uiwon Hwang, Jaewoo Park, Hyemi Jang, Sungroh Yoon, and Nam Ik Cho. Puvae: A variational autoencoder to purify adversarial examples. *arXiv preprint arXiv:1903.00585*, 2019.

[41] Xiaojun Jia, Xingxing Wei, Xiaochun Cao, and Hassan Foroosh. Comdefend: An efficient image compression model to defend adversarial examples. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6084–6092, 2019.

[42] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280*, 2017.

[43] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[44] Rajeev Sahay, Rehana Mahfuz, and Aly El Gamal. A computationally efficient method for defending adversarial deep learning attacks. *arXiv preprint arXiv:1906.05599*, 2019.

[45] Chawin Sitawarin and David Wagner. Defending against adversarial examples with k-nearest neighbor. *arXiv preprint arXiv:1906.09525*, 2019.

[46] Morgane Goibert and Elvis Dohmatob. Adversarial robustness via adversarial label-smoothing. *arXiv preprint arXiv:1906.11567*, 2019.

[47] Yifan Ding, Liqiang Wang, Huan Zhang, Jinfeng Yi, Deliang Fan, and Boqing Gong. Defending against adversarial attacks using random forest. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[48] Aamir Mustafa, Salman H Khan, Munawar Hayat, Jianbing Shen, and Ling Shao. Image super-resolution as a defense against adversarial attacks. *arXiv preprint arXiv:1901.01677*, 2019.

[49] Connie Kou, Hwee Kuan Lee, Teck Khim Ng, and Ee-Chien Chang. Enhancing transformation-based defenses using a distribution classifier. *arXiv preprint arXiv:1906.00258*, 2019.

[50] Jiayang Liu, Weiming Zhang, Yiwei Zhang, Dongdong Hou, Yujia Liu, Hongyue Zha, and Nenghai Yu. Detection based defense against adversarial examples from the steganalysis point of view. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4825–4834, 2019.

[51] Yifeng Li, Lingxi Xie, Ya Zhang, Rui Zhang, Yanfeng Wang, and Qi Tian. Defending adversarial attacks by correcting logits. *arXiv preprint arXiv:1906.10973*, 2019.

[52] Tejas Borkar, Felix Heide, and Lina Karam. Defending against adversarial attacks through resilient feature regeneration. *arXiv preprint arXiv:1906.03444*, 2019.

[53] Jonathan Aigrain and Marcin Detyniecki. Detecting adversarial examples and other misclassifications in neural networks by introspection. *arXiv preprint arXiv:1905.09186*, 2019.

[54] Yao Qin, Nicholas Frosst, Sara Sabour, Colin Raffel, Garrison Cottrell, and Geoffrey Hinton. Detecting and diagnosing adversarial images with class-conditional capsule reconstructions. *arXiv preprint arXiv:1907.02957*, 2019.

[55] Rajkumar Theagarajan, Ming Chen, Bir Bhanu, and Jing Zhang. Shieldnets: Defending against adversarial attacks using probabilistic adversarial robustness. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6988–6996, 2019.

[56] Uiwon Hwang, Jaewoo Park, Hyemi Jang, Sungroh Yoon, and Nam Ik Cho. Puvae: A variational autoencoder to purify adversarial examples. *arXiv preprint arXiv:1903.00585*, 2019.

[57] Pengcheng Li, Jinfeng Yi, Bowen Zhou, and Lijun Zhang. Improving the robustness of deep neural networks via adversarial training with triplet loss. *arXiv preprint arXiv:1905.11713*, 2019.

[58] Zhezhi He, Adnan Siraj Rakin, and Deliang Fan. Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 588–597, 2019.

[59] Haichao Zhang and Jianyu Wang. Defense against adversarial attacks using feature scattering-based adversarial training. *arXiv preprint arXiv:1907.10764*, 2019.

[60] Arash Azizimazreah and Lizhong Chen. Shortcut mining: Exploiting cross-layer shortcut reuse in dcnn accelerators. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 94–105. IEEE, 2019.

[61] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh. From high-level deep neural models to fpgas. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, page 17. IEEE Press, 2016.

[62] Shaoli Liu, Zidong Du, Jinhua Tao, Dong Han, Tao Luo, Yuan Xie, Yunji Chen, and Tianshi Chen. Cambricon: An instruction set architecture for neural networks. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 393–405. IEEE Press, 2016.

[63] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pages 764–775. IEEE Press, 2018.

[64] NVIDIA. Hardware architectural specification. http://nvdla.org/hw/v1/hwarch.html, 2018.

[65] NVIDIA. Unit description. http://nvdla.org/hw/v1/ias/unit_description.html#tab-sdp-supported-use-scenarios, 2018.

[66] NVIDIA. Nvdla primer. http://nvdla.org/primer.html, 2018.

[67] NVIDIA. Nvdla-hw. https://github.com/nvdla/hw, 2018.

[68] Andrew Waterman, Yunsup Lee, David A Patterson, and Krste Asanovi. The risc-v instruction set manual. volume 1: User-level isa, version 2.0. Technical report, CALIFORNIA UNIV BERKELEY DEPT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCES, 2014.

[69] Naveen Muralimanohar, Rajeev Balasubramonian, and Norm Jouppi. Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 3–14. IEEE Computer Society, 2007.

[70] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[71] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[72] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[73] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[74] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[75] Bita Darvish Rouhani, Mohammad Samragh, Mojan Javaheripi, Tara Javidi, and Farinaz Koushanfar. Deepfense: Online accelerated defense against adversarial deep learning. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2018.

[76] Angelo Sotgiu, Ambra Demontis, Marco Melis, Battista Biggio, Giorgio Fumera, Xiaoyi Feng, and Fabio Roli. Deep neural rejection against adversarial examples. *arXiv preprint arXiv:1910.00470*, 2019.

[77] Yingying Hua, Shiming Ge, Xindi Gao, Xin Jin, and Dan Zeng. Defending against adversarial examples via soft decision trees embedding. In *Proceedings of the 27th ACM International Conference on Multimedia*, pages 2106–2114, 2019.

[78] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models. *arXiv preprint arXiv:1805.06605*, 2018.

[79] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. Scale-sim: Systolic cnn accelerator. *arXiv preprint arXiv:1811.02883*, 2018.

[80] Weizhe Hua, Zhiru Zhang, and G Edward Suh. Reverse engineering convolutional neural networks through side-channel information leaks. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[81] Dan Meng, Rui Hou, Gang Shi, Bibo Tu, Aimin Yu, Ziyuan Zhu, Xiaoqi Jia, and Peng Liu. Security-first architecture: deploying physically isolated active security processors for safeguarding the future of computing. *Cybersecurity*, 1(1):2, 2018.

[82] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. In *ACM SIGPLAN Notices*, volume 53, pages 461–475. ACM, 2018.

[83] Caiwen Ding, Siyu Liao, Yanzhi Wang, Zhe Li, Ning Liu, Youwei Zhuo, Chao Wang, Xuehai Qian, Yu Bai, Geng Yuan, et al. C ir cnn: accelerating and compressing deep neural networks using block-circulant weight matrices. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 395–408. ACM, 2017.

[84] Yongwei Zhao, Zidong Du, Qi Guo, Shaoli Liu, Ling Li, Zhiwei Xu, Tianshi Chen, and Yunji Chen. Cambricon-f: machine learning computers with fractal von neumann architecture. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 788–801. ACM, 2019.

[85] Shuyu Cheng, Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. Improving black-box adversarial attacks with a transfer-based prior. *arXiv preprint arXiv:1906.06919*, 2019.

[86] Yuchao Duan, Zhe Zhao, Lei Bu, and Fu Song. Things you may not know about adversarial example: A black-box adversarial image attack. *arXiv preprint arXiv:1905.07672*, 2019.

[87] Tejas Borkar, Felix Heide, and Lina Karam. Defending against adversarial attacks through resilient feature regeneration. *arXiv preprint arXiv:1906.03444*, 2019.