A System-level Perspective Towards Efficient, Reliable and Secure Neural Network Computing

Tao Liu
Florida International University, tliu023@fiu.edu

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

A SYSTEM-LEVEL PERSPECTIVE TOWARDS EFFICIENT, RELIABLE AND SECURE NEURAL NETWORK COMPUTING

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

ELECTRICAL AND COMPUTER ENGINEERING

by

Tao Liu

2020

To: Dean John L. Volakis
College of Engineering and Computing

This dissertation, written by Tao Liu, and entitled A System-level Perspective Towards Efficient, Reliable and Secure Neural Network Computing, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Arif I. Sarwat

Gang Quan

Jason Liu

A. Selcuk Uluagac

Wujie Wen, Major Professor

Date of Defense: March 20, 2020

The dissertation of Tao Liu is approved.

Dean John L. Volakis
College of Engineering and Computing

Andrés G. Gil
Vice President for Research and Economic Development
and Dean of the University Graduate School

Florida International University, 2020

DEDICATION

To my late grandfather and my family.

ACKNOWLEDGMENTS

First and foremost, I would like to express my gratitude to my advisor, Dr. Wujie Wen, for his excellent guidance, caring, patience, and invaluable insights throughout all the stages of researching and writing this dissertation. His passion and dedication are incomparable and always inspire me.

Next, I wish to express my great appreciation to my committee members, Dr. Arif Sarwat, Dr. Gang Quan, Dr. Jason Liu, and Dr. A. Selcuk Uluagac, for their valuable discussions improving the quality of this dissertation.

Furthermore, I would like to extend my gratitude to all the lab mates, Zihao Liu, Qi Liu, and Nuo Xu, for creating an amazing research environment and greatly helping me achieve my research goals.

Last, I would like to acknowledge the research support provided by the Department of Electrical and Computer Engineering at Florida International University, the support of the National Science Foundation under Grant Numbers: CNS-1423137, CNS-1840813, SPX-1919182, and 2016-2017 Collaborative Seed Award Program of Florida Center for Cybersecurity (FC^2).

ABSTRACT OF THE DISSERTATION

A SYSTEM-LEVEL PERSPECTIVE TOWARDS EFFICIENT, RELIABLE AND SECURE NEURAL NETWORK COMPUTING

by

Tao Liu

Florida International University, 2020

Miami, Florida

Professor Wujie Wen, Major Professor

The Digital Era is now evolving into the Intelligence Era, driven overwhelmingly by the revolution of Deep Neural Network (DNN), which opens the door for intelligent data interpretation, turning the data and information into actions that create new capabilities, richer experiences, and unprecedented economic opportunities, achieving game-changing outcomes spanning from image recognition, natural language processing, self-driving cars to biomedical analysis. Moreover, the emergence of deep learning accelerators and neuromorphic computing further pushes DNN computation from cloud to the edge devices for the low-latency scalable on-device neural network computing. However, such promising embedded neural network computing systems are subject to various technical challenges.

First, performing high-accurate inference for complex DNNs requires massive amounts of computation and memory resources, causing very limited energy efficiency for existing computing platforms. Even the brain-inspired spiking neuromorphic computing architecture which originates from the more bio-plausible spiking neural network (SNN) and relies on the occurrence frequency of a large number of electrical spikes to represent the data and perform the computation, is subject to significant limitations on both energy efficiency and processing speed.

Second, although many memristor-based DNN accelerators and emerging neuromorphic accelerators have been proposed to improve the performance-per-watt of embedded DNN computing with the highly parallelizable Processing-in-Memory (PIM) architecture, one critical challenge faced by these memristor-based designs is their poor reliability. A DNN weight, which is represented as the memristance of a memristor cell, can be easily distorted by the inherent physical limitations of memristor devices, resulting in significant accuracy degradation.

Third, DNN computing systems are also subject to ever-increasing security concerns. Attackers can easily fool a normally trained DNN model by exploiting the algorithmic vulnerabilities of DNN classifiers through adversary examples to mislead the inference results. Moreover, system vulnerabilities in open-sourced DNN computing frameworks such as heap overflow are increasingly exploited to either distort the inference accuracy or corrupt the learning environment.

This dissertation focuses on designing efficient, reliable, and secured neural network computing systems. An architecture and algorithm co-design approach is presented to address the aforementioned design pillars from a system-level perspective, namely efficiency, reliability and security. Three case study examples centered around each design pillar, including Single-spike Neuromorphic Accelerator, Fault-tolerant DNN accelerator, and Mal-DNN: Malicious DNN-powered Stegomalware, are discussed in this dissertation, offering the community an alternative thinking about developing more efficient, reliable and secure deep learning systems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 The Emerging Machine Learning (ML) Accelerator

Deep Neural Networks (DNNs) are nowadays becoming the *de facto* technique to promote the artificial intelligence (AI) industry, as witnessed by the consistent breakthroughs in a myriad of real-world applications spanning from computer vision, speech recognition, object detection, game playing to self-driving vehicles [68, 114, 106]. With the increasing support of DNN programming software and computing hardware, many enterprise giants are starting to offer Machine Learning as a Service (MLaaS) through their cloud infrastructures, such as Amazon AWS [6], Google Cloud Platform [45], and Microsoft Azure [89]. Meanwhile, users can also exchange or purchase the pre-trained "Plug & Play" DNN model on open machine learning marketplace [58], thus to quickly deploy and consume ML services in their private environment.

However, performing the high-accurate DNN computing requires the massive amounts of computation and memory resources, leading to limited energy efficiency. For instance, the recognition implementation of CNN–AlextNet [66] involves not only huge volumes of parameters (61 million) generating intensive off-chip memory accesses but also a large number of computing-intensive high precision floating-point operations (1.5 billion) [35]. Such a weakness makes these solutions less attractive for many emerging applications of mobile autonomous systems like smart device, Internet-of-Things (IoT), wearable device, robotics etc., where very tighten power budget, hardware resource and footprint are enforced [7, 49].

To overcome these challenges in DNN computation, the hardware machine learning accelerators, including DNN accelerator and spiking neuromorphic accelerator, have recently emerged to enable the high-performance DNN computation on embedded devices. To ease the significant computation and data movement overhead of DNN workload, many research efforts have been put on developing high-performance and energy-efficient DNN hardware accelerators, such as FPGA, CMOS and non-CMOS based ASICs [19, 24, 75, 55, 25, 104, 111, 12, 77, 80, 78]. The recently invented resistive random access memory (ReRAM) features as one promising solution among the non-CMOS based DNN accelerators, by providing the computation and storage simultaneously within the memristor crossbar array. The memristive dot product, which naturally represents the key element of DNN computation–multiply-accumulate (MAC) operation [66, 126], can be conducted within the memristor array efficiently by exploiting the relationship between a dot product computation and the currents in a resistive mesh [2, 96, 39, 126]. Such a highly paralleled computing architecture significantly improves the performance-per-watt of DNN accelerators [55, 25, 104, 111], far exceeding that of CMOS-based counterparts.

Moreover, in addition to the DNN accelerator, the emerging spiking-based neuromorphic accelerator, which is inspired from the biological spiking neural network (SNN), has featured as achieving tremendous computing efficiency at much lower power of small footprint platforms, e.g. the famous IBM TrueNorth chip that has total 1 million synapses and an operating power of $\sim$70mW [3]. These low-power, light, and small single-chip solutions leverage the efficient event-driven concept to ease the computational load and enable possible cognitive applications in resource limited platforms, creating a very unique but promising branch of neuromorphic computing research [93, 27].

1.2 Motivation: Challenges in ML Accelerator Design

The challenges in designing the machine learning accelerator can be categorized into three different aspects, including reliability, efficiency, and security.

Efficiency. Although many emerging machine learning (ML) accelerators have been proposed to improve the performance of neural network computation, they still suffer from the limited energy and processing efficiency due to the less-optimized architecture and algorithms. For example, in spiking neuromorphic accelerator design, the information is usually conveyed by the occurrence frequency of spikes (rate coding) or their firing time (time coding). Compared to the rate-based SNN, the more biological plausible time-based SNN may offer better energy efficiency and system throughput [118], since theoretically the information can be flexibly embedded in the time (temporal) domain of short and sparse spikes instead of the spiking count represented by a group of power-hungry dense spikes in rate coding [3, 76]. Meanwhile, the processing efficiency of time-based SNN can be further enhanced by performing an early decision making based on the temporal information extracted from early fired spikes, while in rate coding, the classification cannot be initiated until the last moment, e.g. winner-takes-all rule by sorting the number of spikes fired during the entire period of decoding time for each output neuron [83].

However, the potentials of time-based design are significantly underestimated due to lack of efficient hardware-favorable solutions for time-based information representation and complex spike-timing-dependent (temporal) training of biological synapses towards practical cognitive applications [122]. On one hand, translating the input stimulus (i.e. image pixels) to the delay of the spikes is non-trivial because the coding efficiency can be easily degraded by the biased spike delays distributed in the limited coding intervals. On the other hand, training of the rate-based SNN can be usually performed off-line by directly borrowing the standard back-propagation algorithm from artificial neural network (ANN) [76]. However, this time-independent learning rule does not fit the time-dependent SNN because of a fundamentally different learning mechanism. Developing efficient multilayer learning algorithms to enhance the potentials of time-based SNN is non-trivial due to its fundamentally different processing paradigm – the time-based spiking voltage modulation with a non-differentiable threshold function [41, 11, 90, 124, 129, 79, 105]. Despite of many existing time-based learning rules like "Tempotron" [48] and "SpikeProp" [11], those proof-of-concept algorithms are neither compatible with multilayer extension nor feasible to handle the realistic applications due to theoretical limitations or expensive convergence of learning etc. Thus, an efficient multi-layer time-based learning algorithm that can merge the algorithmic power of deep learning to the efficiency of the time-based SNN architecture will be very crucial.

Reliability. A fundamental challenge faced by these memristor-based emerging DNN accelerators is their poor reliability because the DNN weight, which is represented as the memristance of each memristor cell, can be easily distorted by the inherent physical limitations of memristor device [94, 54, 87, 20]. For example, electrical noise and process variations can limit the programming precision of DNN weights on memristors. Meanwhile, memristance drift can also cause weight perturbations after the DNN weights are programmed. These non-ideal factors can in turn degrade the DNN accuracy, eventually harming the reliability of DNN inference on these accelerators [125].

To overcome this problem, traditional solutions rely on the common circuit techniques such as feedback control [125] and error correction code (ECC) [36, 95], leading to high programming cost and considerable hardware overhead. Although recent works have also investigated errors in ReRAM accelerators [22, 74], their solutions focus on permanent defects (i.e., stuck zero or one fault), overlooking the far more common noise, drifting, and programming errors these devices are likely to encounter. Moreover, these solutions for tolerating defects usually involve non-trivial retraining, which is far from scalable in the envisioned scenario of a neural network trained once in the cloud and deployed to many edge devices each with unique footprint of ReRAM defects and errors.

Security. Deep learning systems are also subject to ever-increasing security concerns. It is common for the non-ML expert to directly consume ML services through these devices from the third-party without understanding the end-to-end DNN process on data, training, and testing etc., which could be untrustworthy and malicious. Prior studies show that adversary can easily fool a normally trained DNN model by exploiting the algorithmic vulnerabilities of DNN classifiers through adversary examples [42, 97, 120, 34] or poisoning attacks [23, 60], therefore to mislead the DNN inference results. Besides, DNN backdoor [47, 81] can be crafted into DNN through poisoned training data for targeted misclassification using any input including a specific trigger. Recent study [110] shows that DNN training algorithm can be also abused to steal the secret training data from user side by slightly modifying the training regularization, significantly compromising user privacy. Also, the DNN output can be further used to encrypt traditional malware to create "DeepLocker" [65] for a highly targeted and evasive attack.

However, adversary settings in prior work mainly address traditional cybersecurity concerns and have not been well explored for the embedded ML accelerators. These devices are typically ultra resource constrained and are operated in a hostile (not physically secure) environment without human intervention. Such distinct characteristics make prior DNN attacks less feasible and evasive. For example, they usually employs highly compressed DNN models with minimal model size budget [50], which could be insufficient to hold the stolen data as expected in [110]. Moreover, DNN trojan trigger [81] is subject to natural noises or input variations in a hostile environment. These specific concerns motivate us to take an initial step to explore the malicious DNN-powered attacks for embedded ML accelerators.

1.3 Contribution in this Dissertation

This dissertation focuses on designing the efficient, reliable, and secured neural network computing system. **In particular, this dissertation presents a comprehensive investigation on embedded machine learning accelerator design from a system-level perspective of the hardware, software, and end-to-end application. The ultimate goal is to illustrate how to leverage an neural network architecture and algorithm co-design approach to address the three design challenges on machine learning accelerators, namely reliability, efficiency and security.** The main contribution of this study is threefold:

First, we developed the time-based single-spike neuromorphic architecture and temporal back-propagation algorithm to facilitate the ultra sparse coding in emerging neuromorphic accelerator, so as to significantly escalate the power and processing efficiency when handling realistic applications. Our design has following features: 1) we developed a precise-temporal encoding approach to efficiently translate the information into the temporal domain of a single spike to dramatically reduce the energy, while offering efficient model size reduction; 2) we developed a supervised temporal learning algorithm with novel average delay response model to enable the temporal error back-propagation, significantly enhancing the learning capacity; 3) we developed a novel asymmetric decoding to relieve the unique and serious weight competition issue and significantly improve the accuracy and processing efficiency.

Second, we developed the scalable fault-tolerance of emerging DNN accelerators by leveraging the algorithmic error-resilience of DNN classifiers. Our investigation shows that the hardware defects induced weight disturbances may occur in any layer of a given DNN model, propagate through the network, and affect the final classification outcome **if and only if** the ranking of different classes on the output layer is altered. While the inherent error resilience of DNNs, which already allows it to handle minor precision loss and data errors [71, 50], can be escalated by wisely redesigning the ensemble learning method such as error-correcting output code (ECOC) [30] for modern DNNs. Based on this observation, our study targets the output layer and enhances DNN stability with a collaborative logistic classifier which leverages asymmetric binary classification coupled with an optimized ECOC to improve the error-correction capability of DNN accelerators.

Third, we demonstrated a new type of threat that synthesizes the DNN with stegomalware [113] in DNN accelerator design. We found that the DNN implies an unprecedented opportunity to mix the data and code in DNN model, to create DNN powered stegomalware. By leveraging the structural complexity and error-resilient property of DNN, adversary can easily replace a small portion of DNN model parameters with malicious code, therefore to turn DNN model into an evasive self-contained stegomalware while still maintaining the service quality as normal. The created malicious DNN can be deployed and survive in user's secured environment, and the embedded malicious code in DNN can be executed with a real-world object selected as a trigger event. We developed a variety of model parameter payload injection techniques to embed malicious payload and protect the payload integrity in the compressed DNN models embedded DNN accelerators, without degrading the original accuracy, therefore to conceal the malicious intent with enhanced evasiveness and scalability. We also developed a bundle of triggering techniques to activate the

DNN powered stegomalware with a selected real-world object as trigger event, to overcome the triggering input variations from the physical world and enhance the triggering performance. Besides, we revisited several different mitigation approaches for this emerging threats.

1.4 Dissertation Organization

The dissertation is structured into five chapters, starting from the current chapter that outlines the research scope, purpose, and contribution.

Chapter 2 introduces the single-spike neuromorphic system design. As a study case in designing the efficient machine learning accelerator, we present how to leverage the spiking neural network (SNN) architecture, temporal neural processing and learning algorithms to significantly improve the power and processing efficiency on emerging neuromorphic computing system.

Chapter 3 introduces the fault-tolerant DNN accelerator. As a case study example for designing the reliable machine learning accelerator, we present how to leverage the DNN classification algorithm and re-architecture on DNN output layer to solve the weight disturbance issue induced by the hardware defects on emerging DNN accelerators.

Chapter 4 introduces the Mal-DNN: Malicious DNN-powered Stegomalware. As a case study example for designing the secure machine learning accelerator, we demonstrate how to turn a deep neural network into an evasive self-contained stegomalware by leveraging the DNN architecture and algorithms. Attack surfaces, approaches and possible mitigation techniques are systematically studied on such an emerging threat on machine learning accelerators.

Chapter 5 concludes this dissertation.

CHAPTER 2
SINGLE-SPIKE NEUROMORPHIC ACCELERATOR

2.1 Preliminary

2.1.1 Spiking Neural Coding

The neural coding in Spiking Neural Network (SNN) can be generally categorized as rate coding, time coding, rank coding and population coding etc. [13]. In particularly, the first two codings are the most attractive, since each piece of coded information is only associated with the spikes generated by a single input neuron, offering simplified encoding/decoding procedures and design complexity.

Fig. 2.1 demonstrates an example of conceptual comparison between rate coding and time coding in SNNs. T_e and T_i (R_e and R_i) denote two types of input neurons: the time-coded (rate-coded) excitatory and inhibitory neurons, respectively. The excitatory neuron can exhibit an active response to the stimulus while the inhibitory neuron intends to keep silent. T_1 and T_2 (R_1 and R_2) denote two time-coded (rate-coded) output neurons for the classification. The rate-based SNN generates far more number of spikes than that of time-based SNN in both types of input neurons. After the input spikes are processed by the two different SNNs, a single

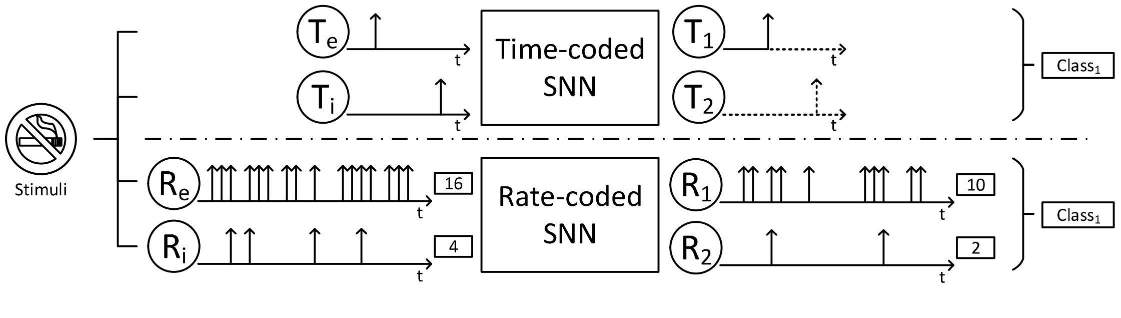


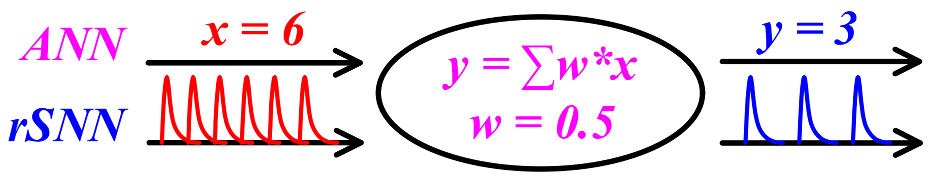
Figure 2.1: The conceptual view of rate-coding and time-coding in SNNs.

spike firing at a specific time interval can perform an inference task in the output layer of the time-based SNN. However, a considerable number of spikes are needed for fulfilling a rate-based classification in the rate-based SNN, indicating a much higher power consumption. Moreover, the rate-based SNN may exhibit a slower processing speed than that of time-based SNN, since the output neuron of the former SNN needs to count the spiking numbers (i.e. through Integrate-and-Fire [14]) in the whole predefined time window, while that of the latter one may quickly suspend its computations once a spike is detected.

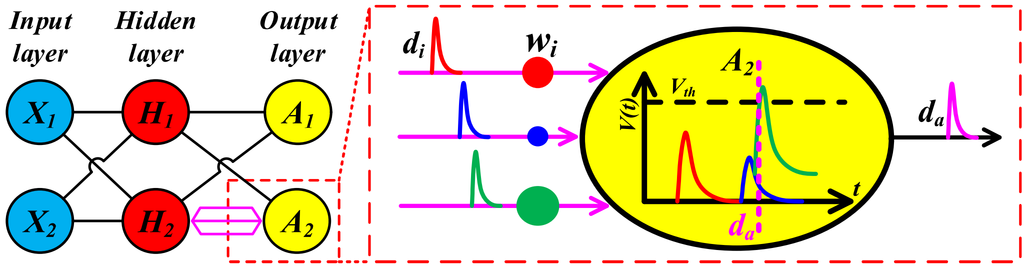
2.1.2 Related Work

SNN based Neuromorphic Accelerator. Many studies have been conducted to facilitate the spiking based Neuromorphic Computing System (NCS) designs in real hardware implementations, including CMOS VLSI circuit [3, 103, 17, 33], reconfigurable FPGA [93], and emerging memristor crossbar [26, 76]. However, these works mainly focus on the rate- or time-based SNN model mapping and hardware implementations, rather than the SNN architecture optimization, i.e. coding, decoding and learning approaches etc.

Temporal Coding. The concept of temporal coding, which relies on the arrival time or delay of a spike train for information representation, has been widely explored and proved in the development of time-based SNN [62, 15]. These theoretical studies, however, mainly emphasize on the biological explanations of time-based SNN models based on simple cognitive benchmarks (i.e. two inputs XOR gate), which are far from the complicated real-world problems such as image recognition. Recently, Zhao et al. [131] proposed an encoding circuit to handle the temporal coding, however, this type of work still concentrates on component-level hardware



(a) Similarity Between ANN and rSNN



(b) Multiple Sub-Synapses and Voltage Thresholding in tSNN

Figure 2.2: Neural processing in ANN, rSNN and tSNN.

implementations with simple case studies, and hence is lack of a holistic architecture-level solution set capable of handling realistic tasks. In [128], a complete time-based SNN design is proposed. However, their solution suffers from limited accuracy fundamentally constrained by existing coding and temporal learning rule, and is not optimized towards hardware-based neuromorphic system designs.

Temporal Learning. As shown in Fig. 2.2(a), the popular learning approaches such as back-propagation [101] can be used in both Artificial Neural Network (ANN) or rate-based SNN. For example, the number of spikes (i.e. "6" here) in rate-based SNN is equivalent to the intensity of input data (ANN-style, $x = 6$). Because the spike rate is closely analogous to information representation of the ANN, many practical multilayer rate-based SNNs are well demonstrated in real-world applications by naturally adopting ANN's backpropogation (BP) algorithm. However, BP is unable to handle precise-time-dependent information due to a fundamentally different neural processing. Many proposals dedicated to the time-based learning have been developed [109, 48, 98]. However, these learning algorithms are neither hardware-favorable nor applicable for realistic tasks due to the expensive convergence and theoretical limitation. For example, in the unsupervised Spike-timing dependent plasticity (STDP) learning rule, the neural network structure and synaptic computation will be exponentially increased due to the expensive convergence and clustering. The proposed "Tempotron" and "Remote Supervised Method (ReSuMe)" can use the teaching spike to adjust desired spiking time for temporal learning, however, are not applicable to handle complicated patterns.

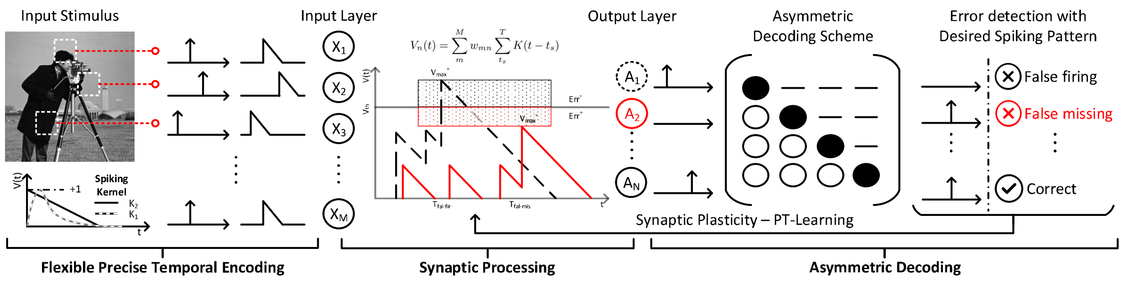
Extending the single-layer time-based SNN to multi-layer time-based SNN can potentially enhance its capability for realistic cognitive tasks. However, designing efficient time-based SNN multi-layer learning algorithms is very challenging due to the fundamentally different training mechanism—the time-based spiking voltage modulation with a non-differentiable thresholding function. We have investigated many existing time-based learning algorithms, i.e. unsupervised spiking-time-dependent plasticity (STDP) [109], theoretical "Tempotron" learning [48] and "SpikeProp" [11]. Those proof-of-concept algorithms are either unable to support multi-layer structure or too bio-plausible to handle the realistic applications because of the cost and difficult convergence of learning etc.

Fig. 2.2(b) illustrates the working principle of the most popular multi-layer supervised temporal learning algorithm- -"SpikeProp" [11] in a two-layer time-based SNN. Here "SpikeProp" can perform complex nonlinear classification in temporal domain by customizing the back-propagation algorithm widely adopted in multi-layer ANNs. Unlike the one-one synaptic connection of two neurons in a standard BP-based multi-layer ANN, the link between any two neurons of two adjacent layers in "SpikeProp" is composed of multiple synaptic terminals (i.e. m), where each terminal serves as a sub-synapse associated with a different spiking delay d_i and weight w_i (see the connection of example neurons H_2–A_2 in Fig. 2.2(b)). A sufficient number of such sub-synapses that can precisely model small delay differences and modulate the spiking voltage kernels between each pre-synaptic and post-synatic neuron pair is needed, leading to significantly enlarged network size. As an example, handling the simple XOR problem with a two-layer architecture (one hidden layer and one output layer) requires $\sim 40\times$ more weights in "SpikeProp" [11] than that of an ANN (240 v.s. 6). Thus, the limited scalability of such a bio-plausible algorithm greatly hinders it from solving more practical and complicated cognitive tasks regardless of the expensive implementation cost, e.g. accurately control the temporal information.

2.2 Single-spike Neuromorphic Architecture

2.2.1 System Architecture

Fig. 2.3 shows a comprehensive data processing flow of proposed single-spike neuromorphic design, namely "*PT-Spike*". First, the stimulus will be captured by the temporal perceptors to generate a sparse spike train (i.e. **single spike**) through "Precise Temporal Encoding". Each spike train will be further modulated in temporal domain by a linear-decayed spiking kernel to form time-dependent voltage pulse. Second, those voltage pulses will be sent to the synaptic network for a weighting



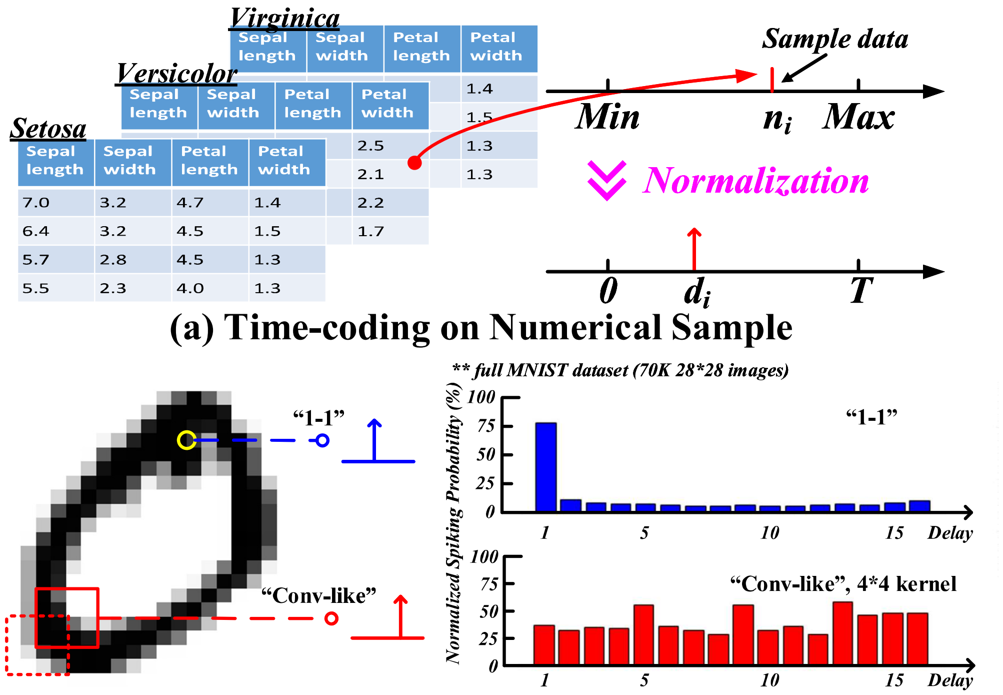
Input Stimulus
Input Layer
Output Layer
Asymmetric Decoding Scheme
Error detection with Desired Spiking Pattern
$V_n(t) = \sum_m^M w_{mn} \sum_{t_s}^T K(t - t_s)$
Spiking Kernel
K_2
K_1
+1
V(t)
X_1
X_2
X_3
X_M
A_1
A_2
A_N
False firing
False missing
Correct
Synaptic Plasticity – PT-Learning
Flexible Precise Temporal Encoding
Synaptic Processing
Asymmetric Decoding

Figure 2.3: The overview of single-spike neuromorphic design.

process, i.e. the memristor crossbar with IFC design can be employed for parallel processing. The output neurons will exhibit time-varying weighting responses due to the time-dependent input information. After that, the output neuron will fire a spike if the weighted post-synaptic voltage crosses a threshold voltage. Then spike trains from the output layer will be transmitted to the "Asymmetric Decoding". Finally, the target pattern will be classified by analyzing the synchronized output spikes with a predefined asymmetric rule. During the learning procedure, desired spike patterns are coded by following the similar asymmetric rule during decoding. The detected errors will be sent-back for synaptic plasticity through "PT-Learning"–a supervised temporal learning algorithm.

2.2.2 Precise Temporal Encoding

In traditional rate coding, a large number of spikes within a proper time window will be needed to precisely indicate the amplitude of an input signal, i.e. the pixel density of visual stimulus. To maximize the power efficiency with minimized number of spikes, the input information will be represented as an extreme sparse train–**single spike** and its occurring delay in aforementioned coding approach. However, such a "one-to-one" mapping between each stimulus and spike train of each input neuron can lead to a significant energy overhead. Meanwhile, the time or temporal information of those spike trains are not fully leveraged by each neuron, resulting in limited coding efficiency thus a dramatical accuracy reduction. As we shall present later, our results on "MNIST" benchmark show that the "one-to-one" mapping achieves very unacceptable training accuracy (($\sim$ 20%) even under a large model size, that is, 784 input neurons for a 28×28 image.



Sepal length	Sepal width	Petal length	Petal width
7.0	3.2	4.7	1.4
6.4	3.2	4.5	1.5
5.7	2.8	4.5	1.3
5.5	2.3	4.0	1.3

(a) Time-coding on Numerical Sample

(b) Time-coding on Visual Sample

Figure 2.4: Comparison on proposed coding scheme.

In "*PT-Spike*", we further propose the "Precise Temporal Encoding". As shown in Fig. 2.3, the "Precise Temporal Encoding" is inspired from human visual cortex and Convolutional Neural Network (CNN), where a Temporal Kernel (i.e. a unit square matrix) will be applied on the full image to capture the spatial information and then translated into a single spike delay in temporal domain as a neuron input by perceiving the localized information from multiple interested pixels, i.e. spiking delay is equal to the average density among several selected pixels. In practice, by selecting a proper stride with which we slide the Temporal Kernel, e.g. smaller than the dimensionality of Temporal Kernel, a portion of localized spatial information will be shared by adjacent kernel sliding. Consequently, the spatial localities can be further transformed into temporal localities, thus to uniformly allocate the spiking delay assigned to each input neuron in time domain, translating into improved coding efficiency and classification accuracy.

To better illustrate the proposed coding techniques, we define following three key parameters: An encoding time window T, a unit time interval τ, and the time encoding resolution $R = \frac{T}{\tau}$. Note τ also denotes the period of a single spike. To make our encoding biological compatible, we also interpret the spike with a short (long) delay as the excitatory (inhibitory) response under strong (weak) stimulus. We explored several possible time-coding schemes on two representative datasets: numerical-style "Iris dataset" (3 classes, 4 attributes) [38] and visual-style "MNIST dataset" (10 handwritten digits) [67], as shown in Fig. 2.4. In Iris dataset, each attribute (i.e. {length, width ...}) can be mapped to a single spike associated with an input neuron. As Fig. 2.4(a) shows, the delay d_i of each single spike generated within T can be calculated as $d_i = T \cdot round\left(1 - \frac{n_i}{max(n_i) - min(n_i)}\right)$, where n_i is the i-th data sample at a selected attribute.

For visual-style "MNIST dataset", we first investigated an existing coding technique adopted in most ANNs and SNNs – the "1-1 coding", i.e. each single pixel is mapped to an input neuron, as shown in Fig. 2.4(b). The delay d_i of the spike generated by the input neuron i is inversely proportional to the associated pixel density p_i: $d_i = T \cdot round\left(1 - \frac{p_i}{max(p_i)}\right)$. Note there will be no spike if $p_i = 0$. However, the coding efficiency of "1-1 coding" is limited because many spikes that should represent different data patterns occur at a common time slot (see the spiking delay distribution of "1-1 coding" in Fig. 2.4(b)). Besides, the number of input neurons is always equal to the image resolution, indicating a large model size. To better leverage the whole encoding time window and reduce the model size, we further develop the "conv-like coding" inspired by human visual cortex (receptive field) and Convolutional Neural Networks (CNNs). By perceiving the localized information from multiple adjacent pixels through a square kernel, spiking delay in "conv-like coding" can be expressed as the number of "0s" within the kernel among the binarized

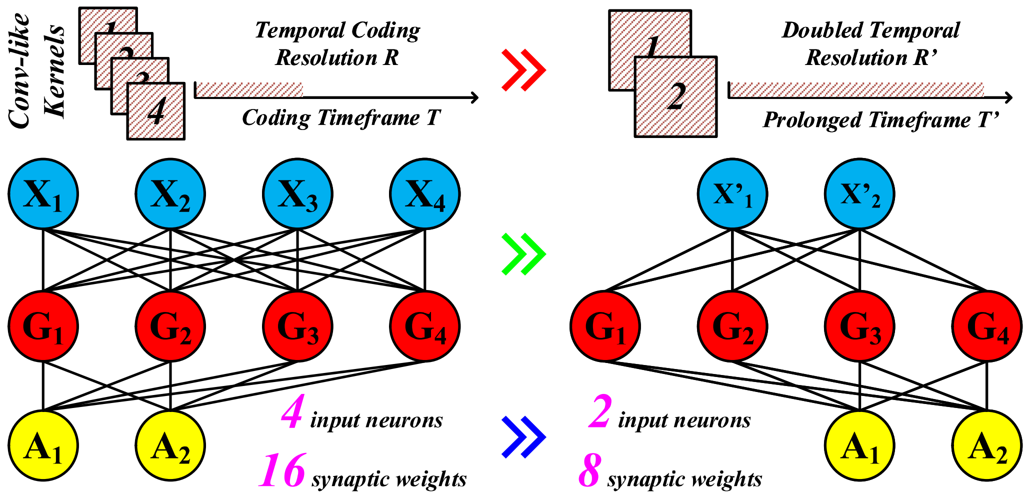


Figure 2.5: Model size reduction through adjustable temporal resolution.

pixels. As Fig. 2.4(b) shows, the spiking delays of "conv-like coding" are almost evenly distributed across the whole time domain, indicating effective utilization of temporal information, thus a potential model size reduction in spatial domain or rather a reduced number of input neurons.

Another unique advantage of the proposed "Precise Temporal Encoding" is to offer a flexible model size reduction. To illustrate the advantage of spatial model size reduction provided by our proposed "conv-like coding", we assume the number of elements covered by the kernel as a square number R. Note $R = \frac{T}{\tau}$ also represents the temporal resolution of encoding. The number of input neurons can be expressed as $M = \lceil \frac{P-\sqrt{R}+1}{S} \rceil^2$, where P and S represent the width of input image and the stride to slide the kernel. "Zero-padding" will be also applied according to the image resolution. Hence the encoding time window T and input neuron number M can be flexibly changed by tunning R without sacrificing the amount of information of an entire image. Fig. 2.5 shows the concept of model size reduction based on "conv-like coding". In this example, the "original design" is configured as $M = 4$ input neurons, 16 synaptic weights for the first layer at a temporal resolution R.

Alternatively, a "size-reduced design" with only $M = 2$ input neurons, 8 synaptic weights (50% less), can be easily achieved by doubling the temporal resolution R or rather the encoding time T (assume τ does not vary). Although the efficiency of model size reduction depends on the percentage of the first-layer weights over the total number of weights, as we shall show later, such a technique is still very effective even without degrading the system accuracy.

2.2.3 Synaptic Processing and Linearized Spiking Kernel

Once the delay for the single spike is determined, as shown in Fig. 2.3, a spiking kernel K will be applied to shape the associated spikes for input neurons. The kernel plays an important role in the following synaptic weighting for the output voltage $V_n(t)$, as shown in Eq (2.1):

$$V_n(t) = \sum_{m}^{M} w_{mn} \sum_{t_s}^{T} K(t - t_s) \tag{2.1}$$

where weight $V_n(t)$ represents the voltage of output neuron n, w_{mn} denotes the synaptic efficacy between input neuron X_m and output neuron A_n. t_s is the decoded spiking delay of X_m. To provide sufficient and accurate temporal information for the classification, the exponential decayed post-synaptic potential in the biological spike response neural model [41] can be expressed as:

$$K_1(t - t_s) = \mu(exp[-(t - t_s)/\tau_1] - exp[-(t - t_s)/\tau_2]) \tag{2.2}$$

where τ (τ_1 and τ_2) denotes decay time constant, and μ is the normalizing constant. However, such an exponential decaying function requires expensive computation and hardware resource. In "*PT-Spike*", we employ a more hardware-favorable kernel function K$_2$–a linear decaying function (see K_1 and K_2 comparison in Fig. 2.3), to

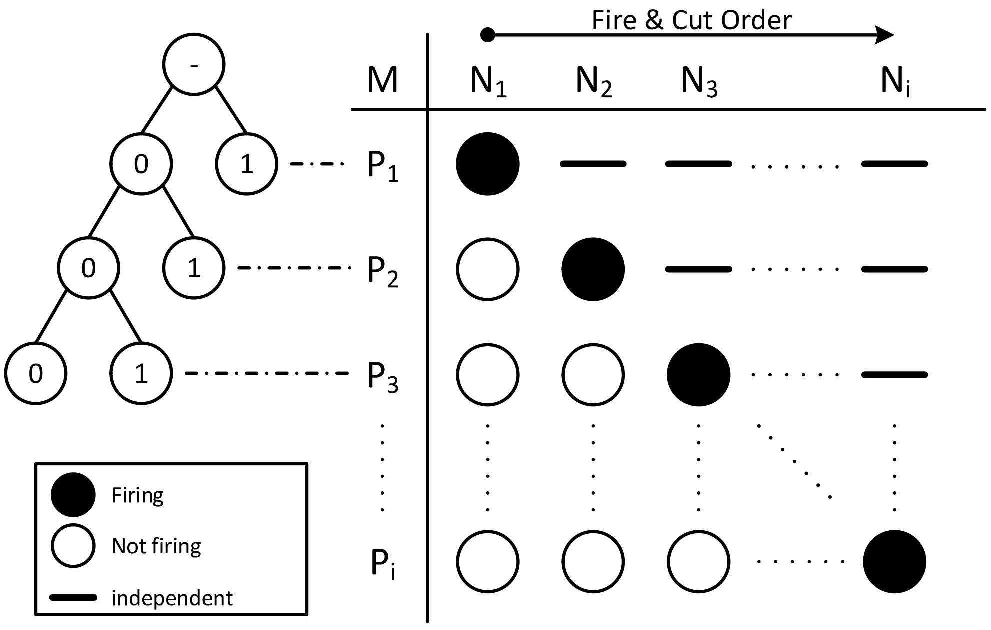


Figure 2.6: An overview of proposed A-Decoding scheme.

simplify the costly dual-exponential function K_1:

$$K_2(t - t_s) = 1 - \tau(t - t_s) \tag{2.3}$$

Such a linear approximation cause very marginal classification accuracy degradation. Besides, this linear kernel function will be also applied to detect the input voltage contributions to the output spike in our proposed "PT-Learning".

2.2.4 Asymmetric Decoding

In '*PT-Spike*", a novel Asymmetric decoding scheme, namely "A-Decoding", is proposed for the classification. As the error signal critical for the proposed supervised temporal learning will be also generated through asymmetric decoding, we will discuss the "A-Decoding" technique first.

In rate-based SNN, the target pattern can be determined by the output neuron with highest spiking numbers. The costly weight updating will be performed in all synapses at each iteration of learning. The subsequent neural competition (weight conflict) among different patterns can be rectified by enough information provided by the large number of input spikes. Hence a good classification accuracy may be achieved for all different patterns. However, the similar case cannot occur in our proposed "*PT-Spike*", since its weight updating solely relies on the very limited number of spare spikes (e.g. a single spike) in temporal domain. In "*PT-Spike*", we further propose the "A-Decoding" to alleviate the neural competition for accuracy improvement.

Fig. 2.6 illustrates the key idea of proposed "A-Decoding", including pattern readout and error detection. Pattern $\{P_i\}$ can be decoded based on the firing status of output neuron $\{N_i\}$. In our asymmetric decoding, the output neuron can work on three different statuses: "firing", "not firing" and "independent", as shown in Fig. 2.6. Note "independent" means that the associated neurons will not participate in the learning process of a certain pattern, and it will only occur in learning mode.

In testing mode, the output neuron will be only in following two status: $\{1 - firing/0 - notfiring\}$. The target pattern is scanned according to the order of the first firing neuron. Assume a binary code $\tilde{N}_1\tilde{N}_2\tilde{N}_3 \cdots \tilde{N}_i$ is generated by output neurons $\{N_i\}$, a Huffman-style decoding procedure can be performed (See Fig. 2.6 left part). For example, if the first firing neuron is N_3, the corresponding code will be $\tilde{0}\tilde{0}\tilde{1}$. Thus, the target pattern is P_3. In "*PT-Spike*", the early detection of testing, namely "Fire&Cut", can be realized based on the temporal "winner-take-all" rule: Once the IFC of neuron N_i triggers a spike, all the remained IFCs for other neurons will be shut down by following the "Fire&Cut Order", which may save the additional power consumed by the IFCs.

In learning mode, a desired spike pattern is reversely generated according to the Huffman-style decoding of pattern $\{P_i\}$ (See Fig. 2.6 right part). Once a participated neuron N_i triggers an unexpected firing or a missing firing, an error will be detected and only the synaptic weights of N_i will be modified according to our proposed "PT-learning". Note only "partial" output neurons (NOT in"independent" status), will be involved during the learning of pattern $\{P_i\}$, namely "Partial Learning". Such a mechanism significantly accelerates the learning procedure and saves power consumed by the unnecessary neural processing. Meanwhile, $\{N_i\}$ is "asymmetrically" correlated with $\{P_i\}$ and thus can ease the neural competition. For example, neuron N_i only engages in the synaptic plasticity of pattern P_i and will be ignored during the learning of all other patterns. As we shall show later, by taking advantages of "Fire&Cut", "Partial Learning" and "Ease Competition", our proposed "A-Decoding" can significantly enhance the weighting efficiency and learning accuracy.

2.2.5 PT-Learning

Our proposed "PT-Learning" coordinates with the aforementioned "A-Decoding" to capture the errors needed for synaptic weights updating. An error detected by the "A-Decoding" will be processed by "PT-Learning" to generate corresponding weight changes and send back for synapse updating. As shown in Fig. 2.3, based on the actual and expected spiking pattern, two types of errors may occur in the output neuron: "false missing" and "false fire". Here "false missing" means that the integrated voltage can not reach the threshold in output neuron to trigger the expected output spike, while "false fire" is defined as an undesired spike firing.

As shown in Algorithm. 1, once an error is detected, the error spiking time (T_{fal}) and the cost function (Err) will be extracted from T_{max} and $V_{th} - V_{max}$. Here V_{max} and T_{max} are the maximum voltage amplitude and its occurrence time, respectively.

```
Algorithm 1: Post-Synaptic Processing
   // Pseudocode of Asymmetric Decoding and PT-Learning
1  Detecting:
2  foreach output neuron $N_i$ in [$N_1$ .. $N_I$] do
3      if testing mode then
4          if firing then
5              return $P_i$// "Fire&Cut"
6      else
               // learning mode
7          if $N_i$ is independent to $P_i$ then
8              return// "Partial Learning" and "Ease Competition"
9          else if actual firing pattern $\neq$ desired pattern then
10             call Learning($V_{max}$, $T_{max}$)
11 Learning:
   // change synaptic weights of $N_i$
12 Err $\leftarrow V_{th} - V_{max}$
13 $T_{fal} \leftarrow T_{max}$
14 foreach input neuron $X_c$ in [$X_1$ .. $X_M$] do
15     if $K_2(T_{fal} - T_c) \leqslant 0$ then
16         continue// "Partial Updating"
17     else
           // pre-spiking at $T_c$ contributed to post-spiking
18         $\Delta w \leftarrow \lambda Err K_2(T_{fal} - T_c)$
19         $w_{ci} \leftarrow \Delta w + w_{ci}$
```

A negative (positive) Err means a false- fire (missing). Hence, the gradient of Err with respect to each weight w_c at pre-synaptic spiking time T_c can be calculated as:

$$-\frac{\mathrm{d}Err}{\mathrm{d}w_c} = Err \sum_{T_c \leq T_{max}} K_2(T_{max} - T_c) + \frac{\partial V(T_{max})}{\partial T_{max}} \frac{\mathrm{d}T_{max}}{\mathrm{d}w_c} \tag{2.4}$$

Here K_2 is the linear decayed spike kernel defined in Eq.(2.3).

As pre-synaptic spikes are weighted through synaptic efficacy w_c before T_{max}, $\frac{\partial V(T_{max})}{\partial T_{max}} = 0$. By further considering Err into the change of w_c, Δw_c can be expressed as:

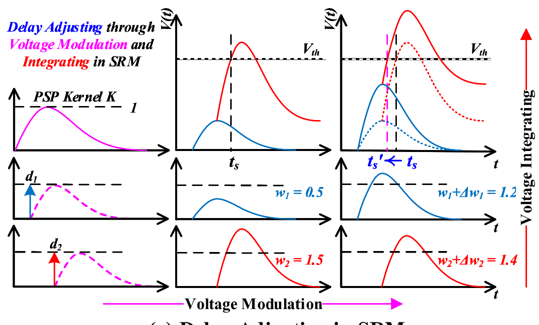
$$\Delta w_c = \lambda Err \sum_{T_c \leq T_{fal}} K_2(T_{fal} - T_c) \tag{2.5}$$

where λ denotes the learning rate and spike kernel K_2 can be used again to calculate the contributions from the input neuron X_c at time T_c.

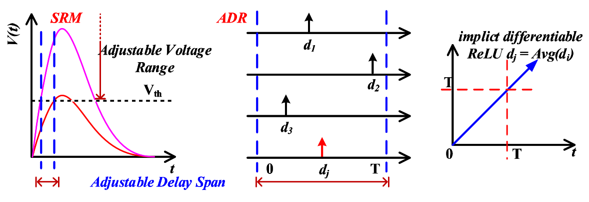
As discussed in "A-Decoding", only partial output neurons will be involved during the learning of a certain pattern, meaning that only partial synaptic weights will be updated. The dual-level acceleration, contributed by both "A-Decoding" and "PT-Learning", can improve the learning efficiency significantly. As we shall show later, the synaptic computation can be reduced more than 200% when compared with the standard learning approach without accelerations. Moreover, "PT-Learning" together with "A-Decoding" can boost the accuracy for realistic recognition task significantly.

2.3 Temporal Error Back-propagation

In "PT-Learning", the error function is to calculate the voltage difference, while the objective is to turn the spike delays. Such a conflict prevents its application on the multi-layer DNN structures. We further develop the temporal error back-propagation algorithm based on proposed Average Delay Response (ADR) neuron model to enable the multi-layer extension on proposed single-spike neuromorphic architecture, namely "MT-Spike".



(a) Delay Adjusting in SRM



(b) Learnability and Temporal ReLU in ADR

Figure 2.7: Design exploration on Average Delay Response (ADR) model.

2.3.1 Average Delay Response

After the information is encoded as the delay of the input spike, the next question becomes how to perform the layer-wise time-based synaptic processing. The objective of the synaptic processing is to generate an output response at each neuron based on its afferent input delays. Thus, how the neural processing model handles the temporal information will directly impact the performance. Existing multi-layer time-based SNN still depends on expensive voltage modulation and threshold based neural processing paradigm due to the absence of the proper loss function and differentiable activation function, significantly hindering its applicability in real-world cognitive tasks. To develop an efficient time-based neural processing, we first explored the processing mechanism of biological plausible Spike Response Model (SRM) [41, 48, 11].

Delay Adjusting Through Weighting Efficacy

Fig. 2.7(a) presents the concept of SRM. Its detailed mathematical model can be expressed as:

$$\begin{cases} V(t) = \sum_i w_i \sum_{d_i} K(t-d_i) \\ K(t-d_i) = \exp\left(-\frac{t-d_i}{\tau_1}\right) - \exp\left(-\frac{t-d_i}{\tau_2}\right) \\ V(t_s) = V_{th} \Rightarrow t_s = d_j \end{cases} \tag{2.6}$$

Where K, τ_1 and τ_2 are the Pre-Synaptic Potential (PSP) kernel function, voltage decay and integrate time constant, respectively. As Fig. 2.7(a) shows, the two updated weightings ($w_1+\Delta w_1$ and $w_2+\Delta w_2$) are applied to the two delayed versions (d_1 and d_2) of PSP spiking kernels, respectively. Accordingly, the integrated voltage w.r.t. time is slightly changed, translating into an equivalent delay adjustment when the voltage reaches the threshold ($t_s \to t'_s$). Despite of the costly analog voltage computation and the target delay extraction, the fundamental goal of SRM is to identify an output spiking time by leveraging the pre-synaptic weights and input spiking delays. Inspired by this observation, we propose the following Average Delay Response (ADR) Model (see Fig. 2.7(b)):

$$d_j(w_{ij}, d_i) = \frac{1}{n}\sum_{i=1}^{n} w_{ij} d_i \tag{2.7}$$

where w_{ij}, d_i and n denote the synaptic weighting efficacies between neuron i and j, input spike delays of neuron i and number of post-synapses. d_j denotes the output spike delay of neuron j. Hence, the output spiking delay can be directly tuned by the weights w_{ij}, speeding up or slowing down the occurrence of an output spike. Note the result of ADR model (see Eq. 2.7) is no less than any input delay d_i, which well complies with the nature of a causal system–a post-synaptic spike will be only trigged by the pre-synaptic input spikes.

Advantages of Average Delay Response Model

First, the proposed ADR model can eliminate the costly voltage kernel modulations and complicated pre-synaptic/post-synaptic time control unavoidable in traditional time-based SNNs, because the proposed time-coding schemes ensure a comprehensive precise delay based information process across all the layers, e.g. performing target classification and error calculation by the delay.

Second, ADR model also increases the adjustable delay range significantly (e.g. a whole encoding time window T) by direct delay weighting when compared with that of traditional SRM limited by the PSP kernel, as shown in Fig. 2.7(b).

Finally, ADR model can implicitly work as a "Special ReLU" [91] function–a non-negative output delay with a smaller value representing a stronger response for an output neuron (the earlier the spike fires, the stronger the response is). Unlike the un-differentiable threshold function in traditional time-based SNN, the "Special ReLU" function is differentiable and thus can facilitate an efficient multilayer learning through temporal error propagation.

2.3.2 Target Delay Set and Readout in ADR

We present the implementation details of Target Delay Setup and Class Readout for following two different cases: 1) A single output neuron with multiple target spiking delays, and the class number is equal to that of target delays; 2) Multiple output neurons with only two target delays, where the output neuron number and the number of classes are identical. Similar to the traditional bio-plausible time-based SNN [11], we assume the selected target delays are no less than the encoding time T in "MT-Spike".

Single Output Neuron

To maximize the temporal information of the output neuron while minimizing the number of output neurons, we assign multiple target spike delays at a single output neuron in "MT-Spike" (see Fig. 2.8(a)). Here one target delay represents one class, i.e. the target delay $T + i \times \tau_e$ for the i-th class, where τ_e is the adjustable time

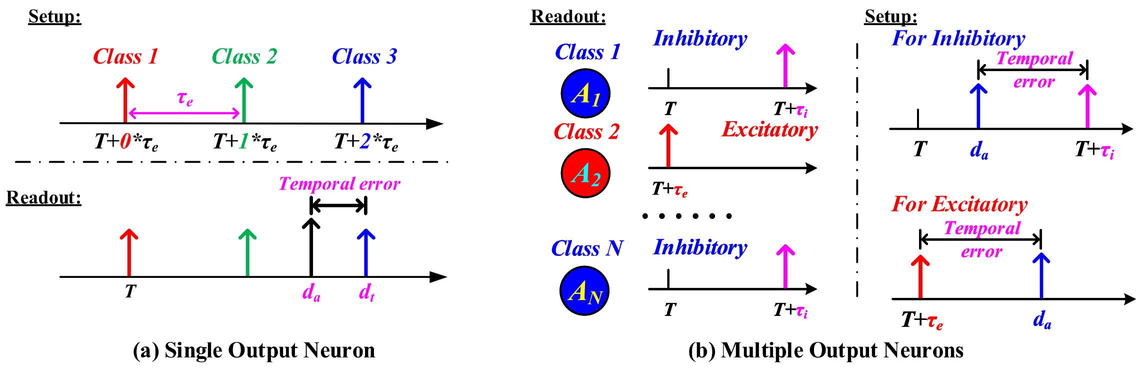


Figure 2.8: Target delay setup and class readout.

interval to differentiate two adjacent classes and is constrained as no less than τ– the period of a single spike. For instance, the target delay can be defined as {T, T+3, T+6} for the three classes {"Setosa", "Versicolour" and "Virginica"} in "Iris dataset" [38], respectively. Here $\tau_e = 3\tau, \tau = 1$.

As Fig. 2.8(a) shows, these target delays will serve as "delay checkpoints" to readout a class according to temporal distances between the actual output delay and those "delay checkpoints", that is, to find the nearest target with smallest temporal distance for a testing. During the training, a temporal error will be calculated based on the delay distance between actual delay and target delay of a class at output neuron if a classification failure happens.

Multiple Output Neurons

To handle the large dataset with more classes, an alternative solution is to increase the number of output neurons, i.e. same as the number of classes, so that each class can be dedicated to one output neuron. To maintain the biological plausibility, short target delay $T + \tau_e$ will be only assigned to the "excitatory" output neuron (i.e. neuron A_2, representing current class label 2) while that of all the remained "inhibitory" neurons are assigned with a same longer delay $T + \tau_i$, as shown in Fig. 2.8(b). Here $\tau_e < \tau_i$.

For example, if the target class label is "1" (i.e. handwritten digits from "0" to "9") in MNIST, ten target delays $\{T{+}4, T{+}0, T{+}4, ..., T{+}4\}$ will be assigned to the ten output neurons $\{A_1, A_2, A_3, ..., A_{10}\}$, respectively, Here we assume $\tau_e = 0$ and $\tau_i = 4$. During the testing, the class readout will be achieved by the "excitatory" output neuron with an "earliest" spike, i.e. the one with minimal actual spike delay. In training mode, each output neuron will calculate an individual temporal error based on the difference between the actual spike delay and the target spike delay if an incorrect class label is identified.

2.3.3 Temporal Back-propagation and Heuristic Loss

Based on our proposed average response model and its implicit temporal "ReLU" activation, an efficient multilayer learning algorithm can be obtained through temporal error back-propagation for "MT-Spike".

Temporal Error Back-propagation

In this section, we present our proposed temporal error back-propagation algorithm. For an output neuron j, the temporal error function is defined as:

$$E_j = \frac{1}{2}\left(d_{t(j)} - d_{a(j)}\right)^2 \tag{2.8}$$

where $d_{t(j)}$ is its target delay and $d_{a(j)}$ is its actual delay, with implicit activation function φ, the output delay of neuron j in layer l is given as:

$$d_j^l = \varphi(net_j^l) = \varphi\left(\frac{1}{n}\sum_{i=1}^{n} w_{ij}^l d_i^{l-1}\right) \tag{2.9}$$

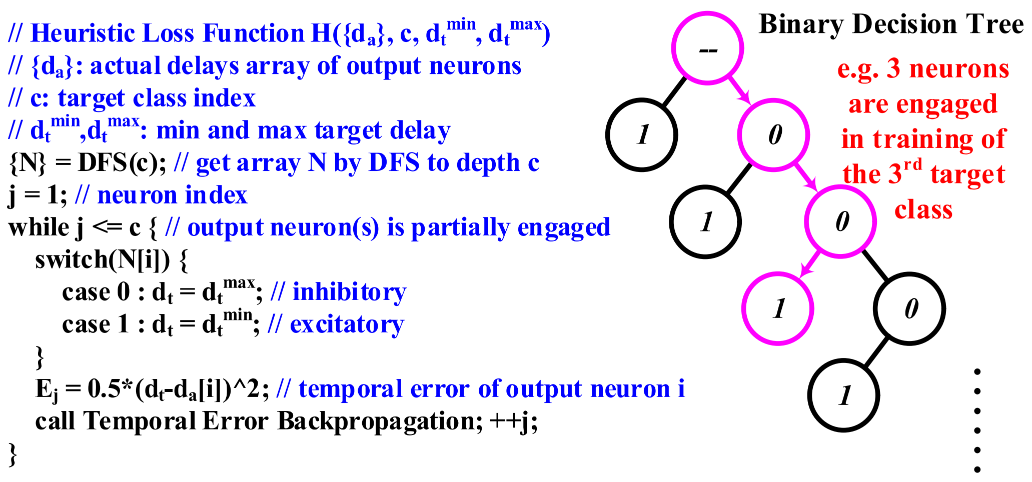
where d_i^{l-1} is the pre-synaptic delay of the i-th neuron and n is the number of pre-synapses. Thus the partial derivative of temporal error with respect to weight w_{ij}^l can be expressed as:

$$\frac{\partial E_j}{\partial w_{ij}^l} = \frac{\partial E_j}{\partial d_j^l}\frac{\partial d_j^l}{\partial net_j^l}\frac{\partial net_j^l}{\partial w_{ij}^l} \tag{2.10}$$

where:

$$\frac{\partial net_j^l}{\partial w_{ij}^l} = \frac{\partial}{\partial w_{ij}^l}\left(\frac{1}{n}\sum_{i=1}^{n} w_{ij}^l d_i^{l-1}\right) = \frac{d_i^{l-1}}{n} \tag{2.11}$$

$$\frac{\partial d_j^l}{\partial net_j^l} = \frac{\partial}{\partial net_j^l}\varphi\left(\mathrm{net}_j^l\right) = 1 \tag{2.12}$$



```
// Heuristic Loss Function H({d_a}, c, d_t^min, d_t^max)
// {d_a}: actual delays array of output neurons
// c: target class index
// d_t^min,d_t^max: min and max target delay
{N} = DFS(c); // get array N by DFS to depth c
j = 1; // neuron index
while j <= c { // output neuron(s) is partially engaged
    switch(N[i]) {
        case 0 : d_t = d_t^max; // inhibitory
        case 1 : d_t = d_t^min; // excitatory
    }
    E_j = 0.5*(d_t-d_a[i])^2; // temporal error of output neuron i
    call Temporal Error Backpropagation; ++j;
}
```

Binary Decision Tree
--
1
0
1
0
1
0
1
e.g. 3 neurons are engaged in training of the 3rd target class

Figure 2.9: Heuristic loss function and binary decision tree.

For neuron j at output layer l:

$$\frac{\partial E_j}{\partial d_j^l} = \frac{\partial E_j}{\partial d_{a(j)}} = \frac{\partial}{\partial d_{a(j)}}\frac{1}{2}(d_{t(j)} - d_{a(j)})^2 = d_{a(j)} - d_{t(j)} \tag{2.13}$$

$$\frac{\partial E_j}{\partial w_{ij}^l} = \frac{d_i^{l-1}(d_{a(j)} - d_{t(j)})}{n} \tag{2.14}$$

For neuron j at hidden layer(s):

$$\frac{\partial E_j}{\partial d_j^l} = \sum_{k=1}^{n}\left(\frac{\partial E_j}{\partial net_k^{l+1}}\frac{\partial net_k^{l+1}}{\partial d_j^l}\right) = \sum_{k=1}^{n}\left(\frac{\partial E_j}{\partial d_j^l}\frac{\partial d_j^l}{\partial net_k^{l+1}}w_{jk}^{l+1}\right) \tag{2.15}$$

where k is the post-synaptic neuron of j, by defining:

$$\delta_j^l = \frac{\partial E_j}{\partial d_j^l}\frac{\partial d_j^l}{\partial net_j^l} = \begin{cases} d_{a(j)} - d_{t(j)} & ,l\ is\ output\ layer \\ \sum_k \delta_k^{l+1} w_{jk}^{l+1} & ,l\ is\ hidden\ layer \end{cases} \tag{2.16}$$

We can obtain the weight updating at learning rate η as:

$$\Delta w_{ij}^l = -\eta\frac{\partial E_j}{\partial w_{ij}^l} = -\eta\delta_j^l\frac{d_i^{l-1}}{n} \tag{2.17}$$

Heuristic Loss Function

In "MT-Spike", the neural competition among different data patterns increases significantly as the dataset becomes more complicated, as the weight updating solely relies on the extreme sparse spike–single spike. Hence, to alleviate the neural competition, we further propose the Heuristic Loss Function in "MT-Spike" as the trigger of Temporal Error Back-propagation– $H(\{d_a\}, c, d_t^{min}, d_t^{max})$, where $\{d_a\}$ and c are the actual delay array of all output neurons and active class of current sample, respectively. d_t^{min} and d_t^{max} represent two target delays for excitatory neuron and inhibitory neuron, respectively.

Fig. 2.9 illustrates the algorithm, as well as the novel data structure of heuristic loss function. An "Huffman" style binary decision tree with its depth equal to the total number of target classes is introduced. Only partial output neuron(s) will be involved by leveraging a depth-first-search (DFS) through the binary decision tree. For example, to process the MNIST dataset (10 classes with label "0" to "9"), the binary decision tree with a maximum depth 10 (the depth of the root is 0) will be generated according to Fig. 2.9. All the nodes, except the root node, in the left (right) subtree are marked as 1 (0). If the 3^{rd} data pattern (class label "2") is selected, a depth-first-search will be conducted on the decision tree until the depth reaches 3. The 3 nodes traversed by the longest searching path (highlighted in Fig. 2.9) indicate only 3 out of total 10 neurons, i.e. A_1 and A_2 as inhibitory neuron and A_3 as excitatory neuron, will participate in the learning of the class "2". Note here only the synaptic weights associated with those three neurons will be updated.

By deploying the Heuristic Loss Function in temporal error back-propagation of "MT-Spike", the computation of the error δ (see Equation. 2.16) can be further simplified as:

$$\begin{cases} \delta^l_{j\in\Gamma} = d_{a(j\in\Gamma)} - d_{t(j\in\Gamma)} & , \textit{output layer} \\ \delta^l_j = \sum_{k\in\Gamma} \delta^{l+1}_k w^{l+1}_{jk} & , \textit{last hidden layer} \\ \delta^l_j = \sum_k \delta^{l+1}_k w^{l+1}_{jk} & , \textit{other hidden layer} \end{cases} \tag{2.18}$$

Table 2.1: PT-Spike: Structural parameters of selected candidates.

Candidate	Number of input neurons	Number of output neurons	Number of synaptic weights	neural processing time-frame T
PT-Spike(4)	196	10	1960	4ms
PT-Spike(16)	169	10	1690	16ms
PT-Spike(25)	144	10	1440	25ms
PT-Spike(100)	100	10	1000	100ms
Diehl-15	784	100	78400	500ms
Lecun-98	784	10	7840	-

where Γ is the set of involved neurons, rather than the whole neurons, for a certain data pattern. In output layer, the weight updating will be partially conducted on the pre-synaptic weights of participated neuron(s):

$$\Delta w^l_{i(j\in\Gamma)} = -\eta \frac{\partial E_{j\in\Gamma}}{\partial w^l_{ij\in\Gamma)}} = -\eta \delta^l_{j\in\Gamma} \frac{d_i^{l-1}}{n} \tag{2.19}$$

2.4 Evaluation

To evaluate the accuracy, processing efficiency and power consumption of our proposed "*PT-Spike*" neuromorphic architecture and "*MT-Spike*" multi-layer extension, extensive experiments are conducted in the platforms like MATLAB and heavily modified open-source simulator–Brian [43].

2.4.1 PT-Spike: Single-spike Neuromorphic Architecture

Experiment Setup

In our evaluation, a full MNIST database is adopted as the benchmark [67]. A set of "*PT-Spike*" designs–"PT-Spike(R)" are implemented to demonstrate the leveraged temporal encoding where "R" denotes the number of interested pixels per input neuron or the size of Temporal Kernel in proposed "Precise Temporal Encoding". We also assume the encoding time frame (T) is $T = \tau \times R(ms)$, where $\tau = 1(ms)$ is

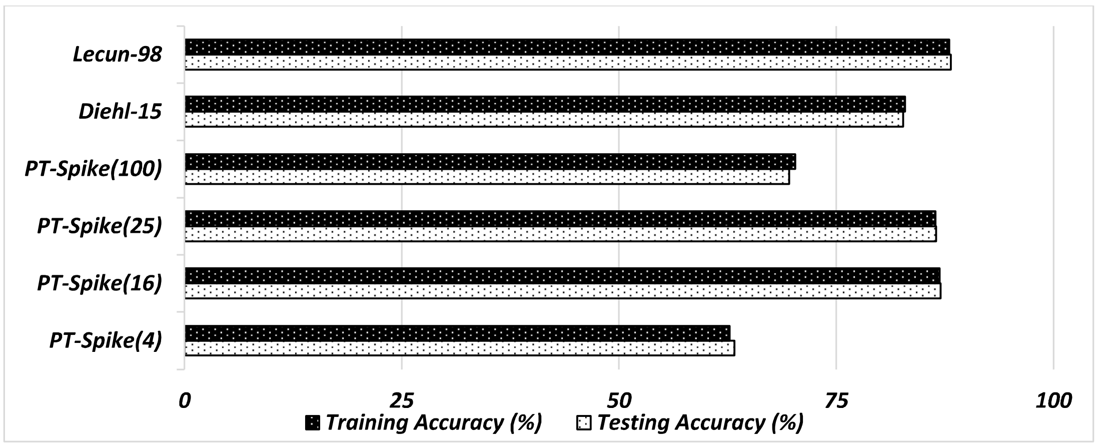


Figure 2.10: Training and testing accuracy on selected candidates.

the fixed minimum time interval to fire the spike. The maximum temporal information T can be adjusted by tuning the parameter R. The number of input neurons (spatial domain) can be expressed as $M = \lceil \frac{P-\sqrt{R}+1}{S} \rceil^2$, where P and S represent the width of an input image and the stride with which we slide the Temporal Kernel. $P = 28$ and $S = 2$ are selected in our evaluations of MNIST dataset. Two representative baselines under similar network configurations, including the rate-coded SNN–"Diehl-15" [29] and the ANN–"Lecun-98" [69], are also implemented for the energy and performance comparisons with proposed "*PT-Spike*". Table. 2.1 presents the detailed structural parameters of selected candidates. Compared with the "Diehl-15" and "Lecun-98", our proposed temporal encoding achieves significant model size reduction for all "PT-Spike" designs, i.e. $\sim 40\times$ ("PT-Spike(4)" v.s. "Diehl-15") and $\sim 4\times$ ("PT-Spike(4)" v.s. "Lecun-98").

Accuracy

Fig. 2.10 shows the accuracy comparison among different "PT-Spike (R)", "Lecun-98" and "Diehl-15". "PT-Spike(25)" can achieve very comparable accuracy at much

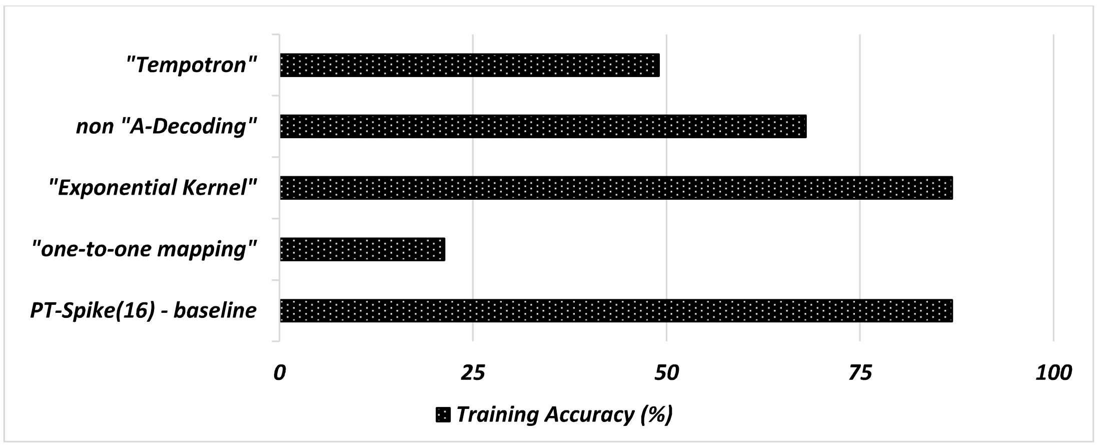


Figure 2.11: Training accuracy on different designs.

lower cost ($\sim$ 86%, 1440 synaptic weights) when compared with “Diehl-15” ($\sim$ 83%, 78400 synaptic weights) and “Lecun-98” ($\sim$ 88%, 7840 synaptic weights). Meanwhile, “PT-Spike(16)” and ‘PT-Spike(25)” also show a very close accuracy ($\sim$ 87% and $\sim$ 86%), which is much better than “PT-Spike(4)” and “PT-Spike(100)” ($\sim$ 63% and $\sim$ 70%).

We also evaluated the individual training accuracy improvement contributed by various proposed techniques, such as “linearized spiking kernel”, “Precise Temporal Encoding”, “A-Decoding” and “PT-Learning”, receptively. Here, we choose the “PT-Spike(16)” as the baseline design that employs all aforementioned techniques. “Exponential Kernel”, “one-to-one mapping”, “non A-Decoding” and “Tempotron” denote the designs that substitute only one out of the four techniques.

As shown in Fig. 2.11, “PT-Spike(16)” shows a very marginal accuracy degradation (0.2%) because of the “linearized spiking kernel” (K_2 in Eq.(2.3)) when compared with the original costly “Exponential Kernel” design (86.9%, K_1 in Eq.(2.2)). Furthermore, “PT-Spike(16)” can boost the accuracy by $\sim$ 400%, $\sim$ 19% and $\sim$ 38% when compared with the designs of “one-to-one mapping” ($\sim$ 21%), “non

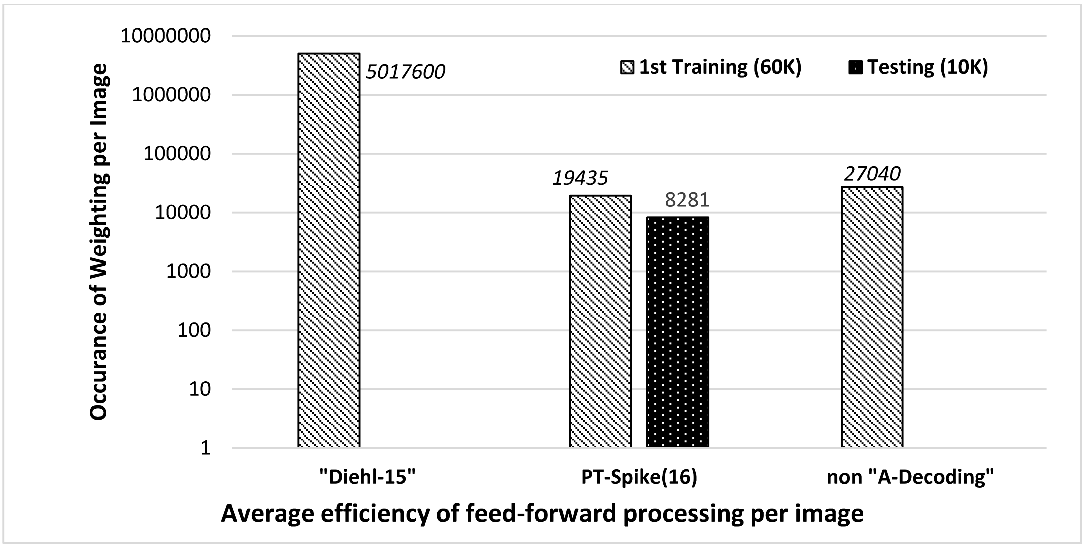


Figure 2.12: Feed-forward efficiency per input image.

A-Decoding" ($\sim$ 68%), and the theoretical "Tempotron" learning rule ($\sim$ 49%), respectively, which clearly demonstrates the effectiveness of the proposed "Precise Temporal Encoding", "A-Decoding" and "PT-Learning".

Processing Efficiency

The occurrence frequency of synaptic events is calculated to evaluate the system processing efficiency, including both weighting and weights updating. Fig. 2.12 compares the number of weighting operations among three designs in the feed-forward pass. Unlike the other candidates, the amount of weight operations of "PT-Spike(16)" is different between training and testing due to the "Fire&Cut" mechanism in"A-Decoding". Hence, the weighting of the first testing iteration is also included in "PT-Spike(16)". Even the "non A-Decoding", i.e. "PT-Spike(16)" without the "A-Decoding" technique, gains $\sim 185\times$ weighting operation reduction as compared with "Diehl-15" since rate-coded SNN requires a long time window to process the spikes with enlarged neuron model size, causing tremendous weighting

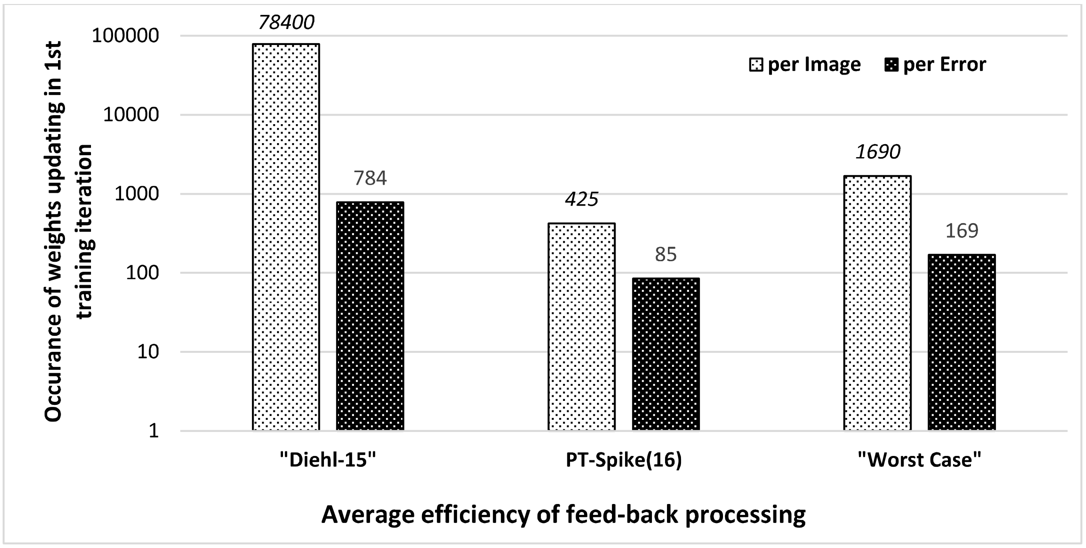


Figure 2.13: Feed-back efficiency.

processes on each time slot. Compared with “non A-Decoding”, weighting operations of “PT-Spike(16)” can be further reduced by $\sim 28\%$ and $\sim 69\%$ in first training iteration and testing iteration, respectively. As expected, the “early-detection” working mechanism in “A-Decoding” removes many unnecessary weighting operations on both “initialized” weights and “well-trained” weights.

We also characterize the occurrence frequency of weights updating during the first training iteration to evaluate the processing efficiency in the feed-back pass. As Fig. 2.13 shows, even “Worst Case” (i.e. “PT-Spike(16)” without employing “A-Decoding” and “PT-Learning”) achieves $\sim 4.6\times$ and $40\times$ reductions on weights updating per image and per error, respectively, when compared with “Diehl-15”. Such impressive improvement is introduced by the significant compressed model size. Moreover, compared with the “worst case”, “PT-Learning” and “A-Decoding” contribute $\sim 2\times$ and $\sim 4\times$ weights updating reduction per error and per image for “PT-Spike(16)”, respectively, demonstrating the effectiveness of “dual-level acceleration” from decoding and learning.

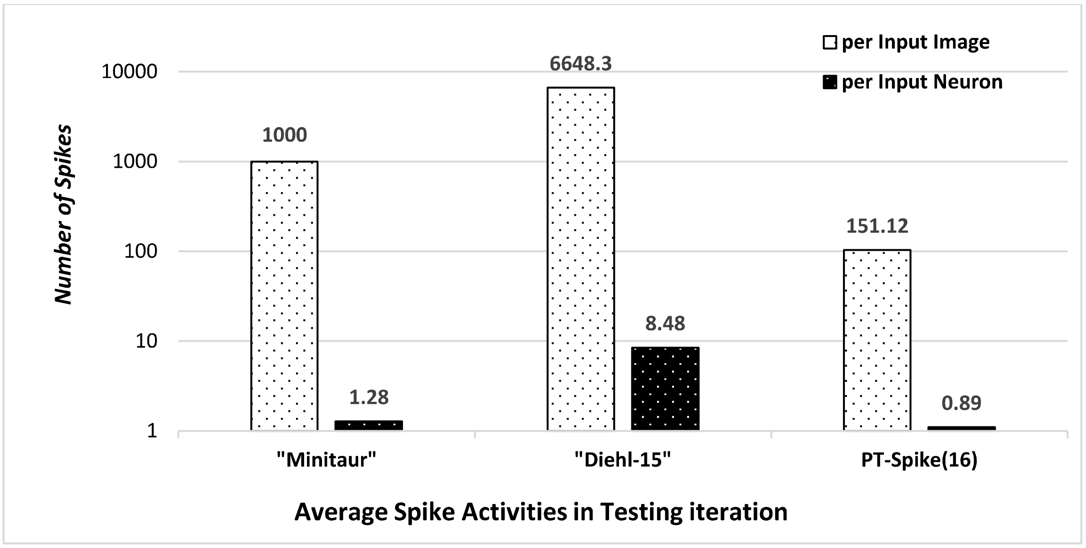


Figure 2.14: Power consumption (α Joules/spike).

Power Consumption

To roughly evaluate the power efficiency contributed by the proposed architecture, we adopted a similar methodology used in [3, 17]. A new candidate "Minitaur" [93] is introduced for a fair comparison since it is a more hardware-oriented rate-coded SNN. As Fig. 2.14 shows, "PT-Spike(16)" saves $\sim 8\times$ and $\sim 64\times$ power for each input neuron and each input image over "Diehl-15", respectively, indicating the efficiency of our proposed single-spike coding technique. Compared with the hardware-oriented rate-coded SNN design "Minitaur", "PT-Spike(16)" can still achieve $\sim 1.4\times$ ($\sim 6.6\times$) power reduction on each input neuron (input image).

Table 2.2: MT-Spike: Structural parameters of selected candidates.

Candidate	Types	Dataset	Network Structure	Number of synaptic weights	Neural processing time-frame T
MT-1	tSNN	Iris	4-25-1	125	16+6
SLMT-3	tSNN	Iris	4-3	12	16+4
SpikeProp	tSNN	Iris	4-25-3	3500	16+4
MLP	ANN	Iris	4-25-3	175	–
MT-1	tSNN	MNIST	169-500-1	85000	16+9 ($\tau = 0.1$)
MT-10(heu/noheu)	tSNN	MNIST	169-500-10	89500	16+4 ($\tau = 0.1$)
SLMT-10(heu/noheu)	tSNN	MNIST	169-10	1690	16+4 ($\tau = 0.1$)
SpikeProp	tSNN	MNIST	784-500-10	7940000	16+4 ($\tau = 0.1$)
Diehl	rSNN	MNIST	784-6400	5017600	50 ($\tau = 0.1$)
Minitaur	rSNN	MNIST	784-500-500-10	647000	–
Lenet-5	CNN	MNIST	1024-C1-S2-C3-S4-C5-F6-10	60840 (340908 conn.)	–

2.4.2 MT-Spike: Single-spike Multi-layer Extension

2.4.3 Experiment Setup

Two representative datasets are selected as the benchmarks of our experiments, including “Iris” [38] and “MNIST” [67]. “Iris” consists of 3 classes, with 50 samples per class and 4 numerical attributes per sample. Note the NOT-linear separable nature of the 3 classes can validate the functions of multi-layer temporal-learning based “MT-Spike”, as well as its classes readout based on the multiple target delays of a single output neuron. We utilize 120 and 30 samples for training and testing purposes, respectively. The “MNIST” dataset, which includes 10 handwritten digits with 60K training images and 10k testing images, is adopted to evaluate the visual recognition capability of “MT-Spike” in terms of accuracy, model size and approximated energy consumption. Several representative candidates, such as multi-layer ANNs, rSNNs and tSNNs, are implemented for a comparison purpose. Batch training is conducted in our evaluation. All the training samples are randomly fed into the candidates per epoch with a $batchsize = 30$ (256) for “Iris” (“MNIST”) until the networks converge, followed by a testing iteration. Table. 2.2 shows the detailed configurations and network types of all selected candidates. All “MT-Spikes” are implemented with a same time window parameter $T = 16$ and learning rate $\eta = 0.01$. The initial weights $w \in (0, 1)$ are randomly generated before training.

Multi-layer Validation on Iris Dataset

As shown in Table. 2.2, "Iris" dataset is used to evaluate the following four networks: "MT-1"– a multilayer MT-Spike implementation with only single output neuron and multiple target delays setup;"SLMT-3"– A simplified version of MT-Spike without hidden layer; "SpikeProp"–traditional bio-plausible multi-layer tSNN with voltage modulation and thresholding process [11]; "MLP"–A Multilayer Perceptron based ANN [100].

Fig. 2.15 compares the testing accuracy of the four aforementioned candidates. As expected, "SLMT-3" exhibits the worst accuracy (56.7%) among all candidates because this single-layer tSNN cannot well distinguish the NOT-linear separable classes. On the contrary, "MT-1" achieves much better accuracy than that of"SpikeProp" (96.7% v.s. 86.7%), and can even approach that of "MLP", demonstrating the enhanced capability through the proposed multi-layer temporal learning rule. Furthermore, as Table. 2.2 shows, "MT-1" reduces the synaptic weights by

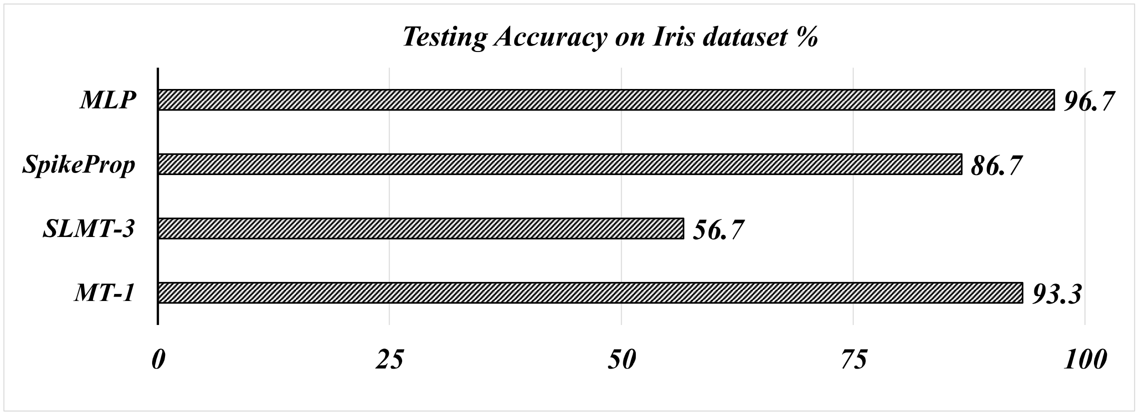


Figure 2.15: Testing accuracy on Iris dataset.

$\sim 28\times$ compared with the "SpikeProp", which well validates the efficiency of single output neuron readout and the Average Delay Response model when handling the simple dataset.

Performance Evaluation on MNIST Dataset

To further evaluate the performance of our proposed "MT-Spike" in a relative complicated dataset "MNIST", seven different networks with more network parameters are chosen, as shown in Table. 2.2. Here " Diehl" is an rSNN trained by the unsupervised STDP learning [29]. "Minitaur" is a hardware-oriented rSNN towards power optimization. Besides, the CNN implementation – "Lenet-5" is included as well for a comparison purpose. For a fair comparison with other SNN candidates, the minimal time interval is set as $\tau = 0.1$ to provide a precise time-based processing for all "MT-Spike" candidates.

Model Size Reduction and Time-coding Efficiency. We first demonstrate the advantages of model size reduction in "MT-Spike" through the proposed "conv-like" time-coding scheme. As shown in Table. 2.2, the proposed "MT-10" achieves $\sim 4.6\times$ reduction on the number of input neurons (169 v.s.784) when compared

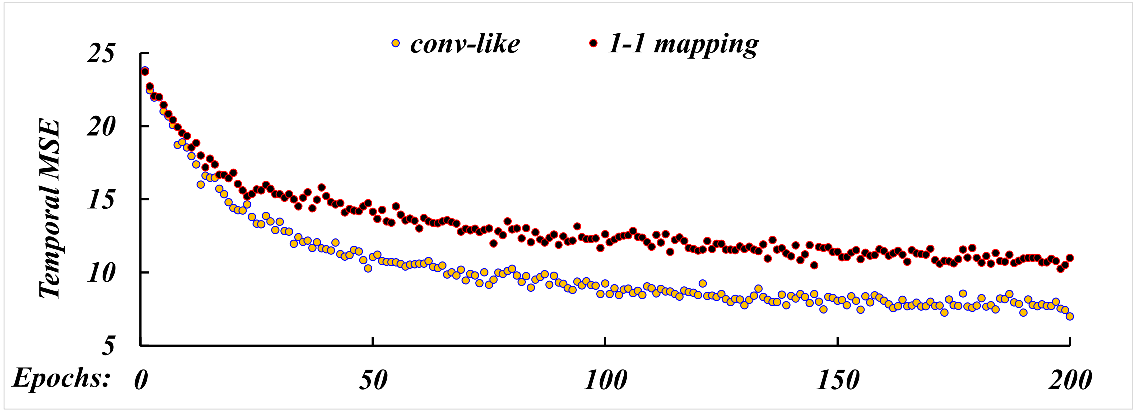


Figure 2.16: Coding efficiency of Conv-like and 1-1 mapping.

with all the other non-“MT-Spike” candidates (except the “Lenet-5” with 1024 neurons), which translates into an impressive model size reduction (or the number of weights) over “SpikeProp”, “Diehl” and “Minitaur”, that is, $\sim 88\times$, $\sim 56\times$ and $\sim 7\times$, , respectively. Note the “SpikeProp” suffers from the largest model size due to a substantial number of sub-synapses between two connected neurons. As we shall discuss later, “MT-10” can even maintain a very high accuracy despite of the significant reduced model size.

Fig. 2.16 also shows temporal mean-square error (MSE) v.s. training epoch for two “MT-10” designs that employ the “conv-like” coding and “1-1 mapping” coding, respectively. As Fig. 2.16 shows, the adopted “conv-like” coding achieves a lower MSE than that of “1-1 mapping” coding at the same epoch, due to its better utilization of temporal information, e.g. the equally distributed spiking delays.

Accuracy Analysis on MNIST dataset. Fig. 2.17 shows the testing results of MNIST dataset among all different designs. As expected, “MT-1” with single output neuron readout is insufficient to handle the complex dataset, resulting in the worst accuracy 63.2%, due to its weak weighting efficiency. We also evaluate the capability of the proposed heuristic loss function. As Fig. 2.17 shows, under a single-

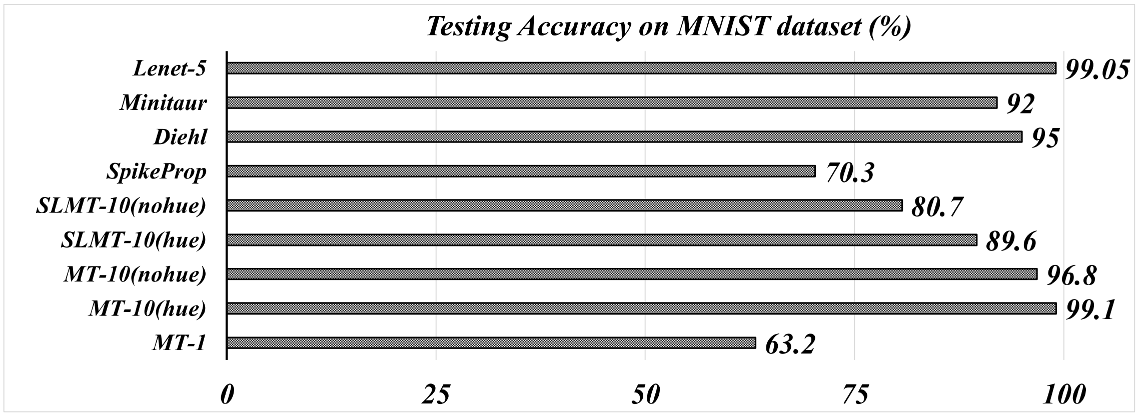


Figure 2.17: Testing accuracy on MNIST dataset.

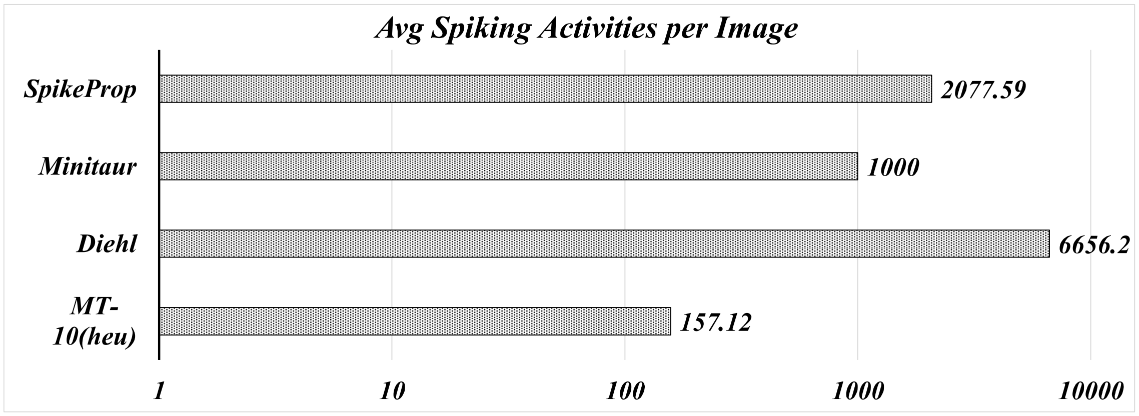


Figure 2.18: Energy analysis based on spiking activities.

layered structure "SLMT", such a technique can boost the accuracy from 80.7% on "SLMT-10(nohue)" to 89.6% on "SLMT-10(hue)", showing a considerable accuracy improvement by alleviating the neural competitions. Moreover, by integrating the heuristic loss function with temporal error backpropagation, the accuracy of "MT-10(hue)" can be further increased to 99.1%, the best results among all candidates and even comparable with the CNN–"Lenet-5"(99.05%). Note the heuristic loss function can still introduce 2.3% accuracy improvement in the multi-layer structure ("MT-10(hue)" 99.1% v.s."MT-10(nohue)" 96.8%).

Energy Consumption. To estimate the energy efficiency of "MT-Spike", we adopt a similar estimation methodology presented in [3, 17]. Measurement is conducted based on the following assumption: a single spike activity consumes $\alpha Joules$ of energy. The total spiking energy is calibrated based on the statistic of the spikes in testing iterations. As shown in Fig. 2.18, "MT-10(hue)" saves $\sim 13\times$ power over "SpikeProp", indicating the efficiency of our proposed average delay response model. Compared with rate-based "Diehl", a $\sim 42\times$ energy reduction is further achieved by "MT-10(hue)" through the efficient single spike temporal representation. Moreover, "MT-10(hue)" can still achieve $\sim 6.3\times$ power reduction compared with the hardware-oriented design "Minitaur", indicating an energy efficient solution for resource-limited embedded platforms.

2.5 Summary

As the rate-based spiking neural network (SNN) is subject to power and speed challenges due to processing large number of spikes, in this work, we systematically studied the possibility of utilizing the more power-efficient time-based SNN in real-world cognitive tasks. Three integrated techniques–precise temporal encoding, efficient supervised temporal learning and fast asymmetric decoding, were proposed to construct the Precise-Time-Dependent Single Spike Neuromorphic Architecture, namely, "PT-Spike". The single-spike temporal encoding offers an energy-efficient information representation solution with the potentials of model size reduction. The supervised learning and asymmetric decoding can work cooperatively to deliver a more effective and efficient synaptic weight updating and classification. Our evaluations on the MNIST database well demonstrate the advantages of "PT-Spike" over the rate-based SNN in terms of network size, speed and power, with a comparable accuracy. Further, we propose the multi-layer time-based spiking neuromorphic architecture, namely "MT-Spike". Through a holistic solution set – practical time-coding scheme, average delay response model, temporal error back-propagation algorithm and heuristic loss function, "MT-Spike" can deliver impressive learning capability while still maintaining its power-efficient information processing at a more compact neural network. Our evaluations well demonstrate the advantages of "MT-Spike" over other rate-based SNN and time-based SNN candidates in terms of accuracy, learning capability and power consumption.

CHAPTER 3

FAULT-TOLERANT DNN ACCELERATOR

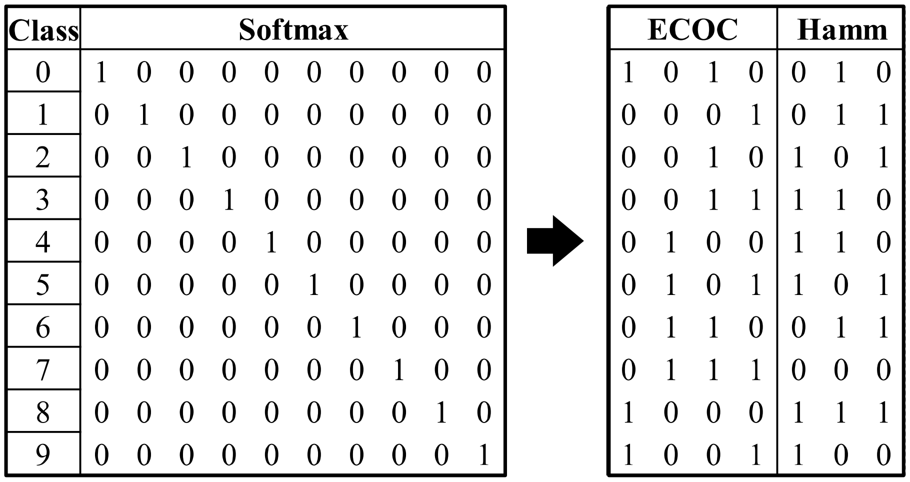
3.1 Preliminary

3.1.1 Deep Neural Network

Deep Neural Network (DNN) is a computational model composed of multiple layers with complex structures to abstract the data at a high level [52] and exhibits high effectiveness in many intelligent applications by leveraging various topologies and learning algorithms [51, 66, 108, 115]. A DNN topology is usually composed of a set of different types of layers. The convolutional layer abstracts features from the inputs through the kernel-based convolutions. The pooling layer performs the down-sampling processing (through max- or mean- pooling) along the feature dimensions, to highlight the common features and reduce the data volume. The fully-connected layer further ranks the confidence of each class based on the weighted features and the non-linear activation functions. The output layer is used as the DNN classifier to make the final decision on target class based on the output logits of certain regressions such as Softmax and Logistic.

3.1.2 Logistic and Softmax Classifier

The logistic classifier is a classic solution to solve the traditional binary classification problem (e.g., true or false). Given input features $x^{(i)} \in \mathcal{R}^n$ and neural network weights θ, the logistic classifier can be trained with label $y^{(i)} \in \{0, 1\}$ through



Class	Softmax									
0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	1

ECOC				Hamm		
1	0	1	0	0	1	0
0	0	0	1	0	1	1
0	0	1	0	1	0	1
0	0	1	1	1	1	0
0	1	0	0	1	1	0
0	1	0	1	1	0	1
0	1	1	0	0	1	1
0	1	1	1	0	0	0
1	0	0	0	1	1	1
1	0	0	1	1	0	0

Figure 3.1: Illustration of Error-Correcting Output Code (ECOC).

logistic regression $h_\theta(x)$ with gradient $\nabla_\theta J(\theta)$:

$$\begin{cases} h_\theta(x) = {}^1\!/\!_{1+\exp(-\theta^\top x)} \\ \nabla_\theta J(\theta) = -\sum_i x^{(i)}(y(i) - h_\theta(x^{(i)})) \end{cases} \tag{3.1}$$

To handle the complex multi-class classification [28], softmax classifier is widely adopted in modern DNNs.

$$\begin{cases} h_\theta(x) = {}^{\exp(\theta^{(k)\top}x))}\!/\!_{\sum_{j=1}^{K}\exp(\theta^{(j)\top}x)} \\ \nabla_{\theta^{(k)}} J(\theta) = -\sum_{i=1}^{m}\left(x^{(i)}\left(\{y^{(i)}=k\} - h_\theta(x)\right)\right) \end{cases} \tag{3.2}$$

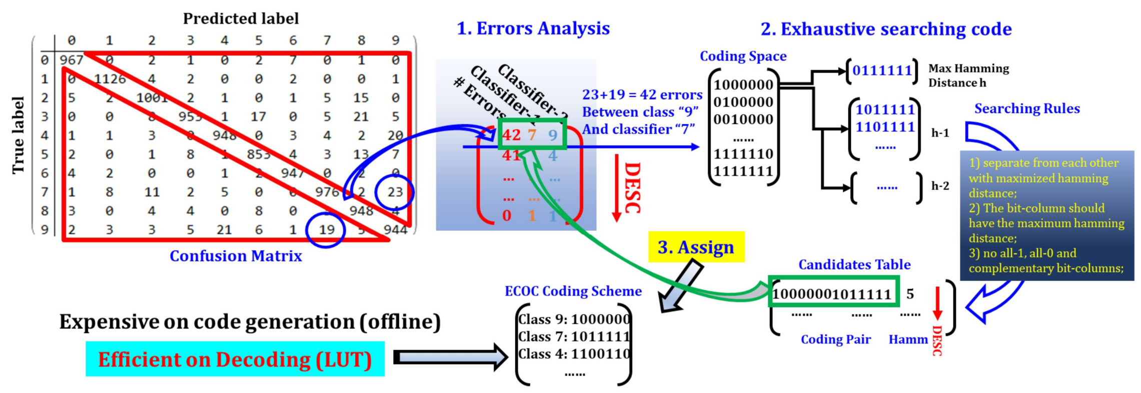
Based on the one-hot coding (i.e., label $y^{(i)} = 1$ for target class and $y^{(i)} = 0$ for others), softmax classifier can push the gradient towards the target class by normalizing the multiple output logits, thereby achieving better accuracy than logistic classifier.

3.1.3 Error-Correcting Output Code

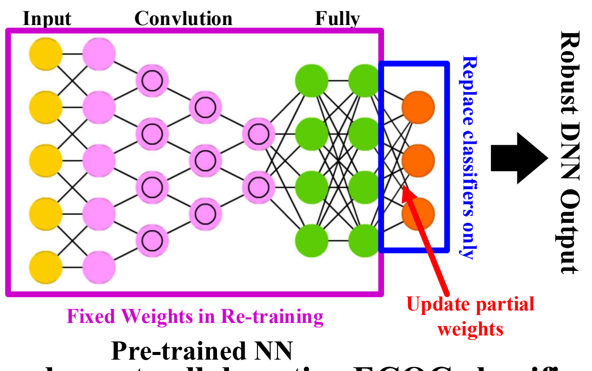
Error-Correcting Output Code (ECOC) is an ensemble learning method for multi-class classification [30, 9, 5, 130]. It solves the multi-class classification as a coding problem–given input features $x^{(i)} \in \mathcal{R}^n$, L independent Logistic classifiers can be trained with a $K \times L$ coding matrix $M_{(K,L)}$, where codeword $M_{(i,L)}$ is assigned with a L-dimension label vector $Y_L^{(i)} \in \{1, 2, \cdots, K\}$. Based on such a flexible learning scheme, appropriate error-correcting coding (e.g. Hamming code) or optimized coding matrix can be further applied on ECOC to increase the Hamming distance of the codewords assigned for different classes (i.e., to enlarge the margin of decision boundary and reduce the complexity of classification) [30], thus to eventually improve the predictive performance of multiple binary classifier. Fig. 3.1 shows an example of ECOC. Given the 10-class problem, one-hot coding (left side) is used for the Softmax classifier by assigning each class mutual exclusively on one single output neuron. Through the ECOC, such a problem can be solved by only 4 independent binary classifiers through the binary encoding (or 7 classifiers for Hamming encoding).

3.1.4 Weight Disturbance on DNN Accelerators

In ReRAM based DNN accelerator, the memristances that represent the weights can be distorted during both write and read operations. Such a weight disturbance problem is caused by the physical limitations of memristor devices [94, 54, 87, 20], which can compromise the system stability and impact the machine learning performance on ReRAM-based DNN accelerators. Prior works [94, 54] report that the geometry variation such as cross-section and thickness may exist in each dimension of the memristor. [87] shows that under the impact of all of these variations, the switching time of memristor cells with identical internal structure follows a log-normal distribution. This indicates that the memristor cell may not be switched to the state with required resistance after a write operation. Besides, [20] reports the memristance



(a) create the DNN favorable ECOC coding scheme



(b) implement collaborative ECOC classifiers

Figure 3.2: Overview of proposed methodology.

drift issue–as the charge flux q and magnetic flux ϕ cause the variations on memristance M as $M(q) = d\phi/dq$, tiny perturbations will be induced on memristance states once the system presents the small recall current (read operation).

3.2 A Scalable Error-resilience Design on DNN accelerators

Fig. 3.2 shows an overview of the proposed methodology to establish the scalable fault-tolerance on DNN, which includes two major steps – 1) create the DNN favorable ECOC coding scheme, and 2) implement collaborative ECOC classifiers. Fig. 3.2(a) and Fig. 3.2(b) show the detailed procedure of each major step.

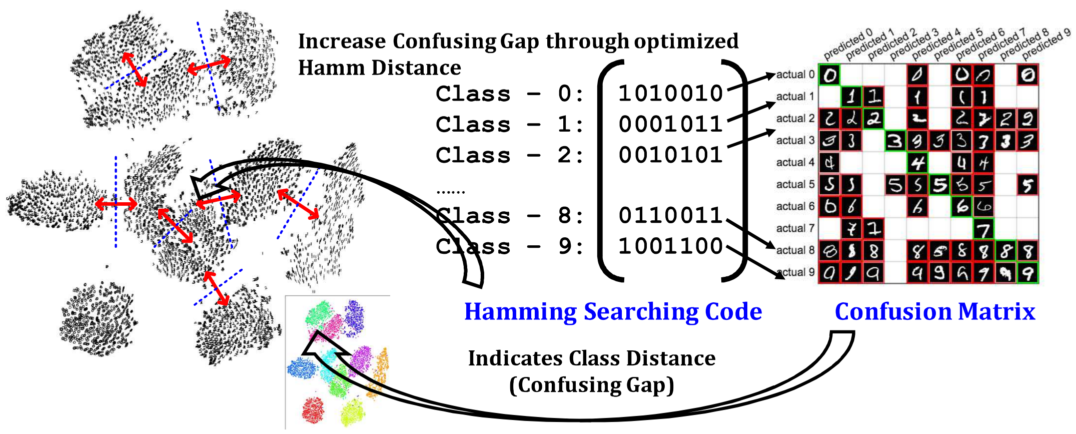


Figure 3.3: Optimize confusing gap through ECOC.

3.2.1 DNN Favorable ECOC Coding Scheme

On top of the ECOC coding based DNN classification, we propose the DNN favorable ECOC coding scheme to improve the reliability of DNN by optimizing the Hamming distance of output code between those confusing classes. Fig. 3.3 illustrates the basic idea of the confusing gap between different classes. In our observation, we find that the weight disturbance issue can degrade the classification accuracy of DNN, particularly on the classes with the marginal confusing gap. Our DNN favorable ECOC coding scheme can improve the Hamming distance between these classes with the enlarged confusing gap, thus to further recovery the degraded DNN accuracy.

To evaluate the confusing gap between any class, we analyze the confusion matrix of DNN. As shown in Fig. 3.2(a), we will first collect the confusion matrix on a given DNN, in which the number of correct and incorrect predictions are summarized with count values and broken down by each class. Second, we perform error analysis to identify the most confusing class pair based on the number of errors. Third, we conduct an exhaustive searching code [61] and take the DNN confusion matrix into consideration. The code-pair with the maximum Hamming distance will be assigned

Algorithm 2: DNN-favorable Searching Code

```
$l \leftarrow$ code length (number of classifiers);
$h \leftarrow$ Hamming distance;
while $(\mathcal{T} = \{0\})\&\&(h \geq 3)$ do
    foreach $i \in [1, 2^n - 1]$ do
        foreach $t \in \mathcal{T}$ do
            if $Hamm(Dec2Bin(i, l), Dec2Bin(t, l)) \geq h$ then
                $\mathcal{T} \leftarrow \mathcal{T} \cup \{i\}$
    if $\mathcal{T} = \{0\}$ then
        h$\leftarrow$ h-1
$m \leftarrow$ number of classes, $x \leftarrow 1$;
n = CodeLen($\mathcal{T}$,h,m);
while $Sizeof(\mathcal{O}) < m\&\&x < 2^n$ do
    foreach $o \in \mathcal{O}$ do
        if $Hamm(Dec2Bin(o, l), Dec2Bin(x, l)) \geq h$ then
            $\mathcal{O} \leftarrow \mathcal{O} \cup \{i\}$
    x$\leftarrow$x+1;
$\mathcal{C} \leftarrow$ confusion matrix;
$j \leftarrow m$;
$\mathcal{S} \leftarrow \{\}$;
while $(Sizeof(\mathcal{C}) \geq 0)\&\&(j \geq 0)$ do
    $c \leftarrow Pop(Max(\mathcal{C}))$;
    if $(xindex(c) \neq yindex(c))\&\&(xindex(c) \notin \mathcal{S})$ then
        Class(xindex(c))$\leftarrow$ $Dec2Bin(Pop(\mathcal{O}))$;
        $\mathcal{S} \leftarrow xindex(c) \cup \mathcal{S}$;
        $j \leftarrow j + 1$;
```

to the most confusing class-pair. Once the DNN-favorable searching code finishes the encoding, the output codeword list can be stored in the LUT and later accessed by collaborative ECOC classifiers for classification.

The pseudo-code of proposed DNN-favorable searching code is described in Algorithm 2, which mainly consists of three parts: prepare searching table (line 1-8), searching code (line 9-15) and code assign (line 16-24). The prepared searching table indicates the maximum number of possible codewords with code length l and Hamming distance h, which can be automatically adjusted. The searching code re-

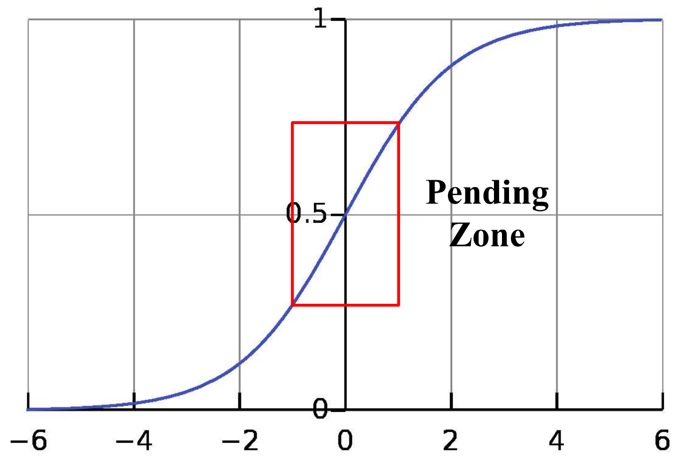


Figure 3.4: The pending zone in collaborative Logistic classifier.

turns a set of codewords that satisfies the aforementioned constraints. In code assign phase, we evaluate the confusion matrix and gradually pick up the corresponding class with strongest neural competition (i.e., with current strongest classification error), which will be assigned with the searched code with highest priority. Note the coding process is done off-line before the neural network accelerators download the well-trained DNN models.

3.2.2 Collaborative ECOC Classifier

To enable the ECOC coding based DNN classification, we use the traditional logistic classifier, which is a classic solution to solve the binary classification problem (e.g., true or false). Given input features $x^{(i)} \in \mathcal{R}^n$ and neural network weights θ, the logistic classifier can be trained with label $y^{(i)} \in \{0, 1\}$ through logistic regression $h_\theta(x)$ with gradient $\nabla_\theta J(\theta)$:

$$\begin{cases} h_\theta(x) = {}^1\!/_{1+\exp(-\theta^\top x)} \\ \nabla_\theta J(\theta) = -\sum_i x^{(i)}(y(i) - h_\theta(x^{(i)})) \end{cases} \tag{3.3}$$

As shown in Fig. 3.2(b), the original classifiers (i.e., softmax) in the output layer of the given model will be replaced by a certain number of logistic classifiers, which is fully connected to the previous layer. Only partial weights of collaborative logistic classifiers will be fine-tuned through transfer learning [127] on a given dataset based on the codeword list created in the previous step, leaving most weights of this model untouched. In particular, we introduce the significance parameter set $\{\beta\}$ and assign the significance on each classifier to establish the correlations among logistic classifiers, so as to increase the DNN reliability (with enhanced classification performance and accuracy):

$$\nabla_\theta J(\theta) \propto -\beta_{(k)} \cdot \sum_i x^{(i)}(y_{(k)}(i) - h_\theta(x^{(i)})) \tag{3.4}$$

In our implementation, to simplify the approach and better control the pace of weight update, a regularization term $\sigma(x, \theta)$ is applied on the loss function $\mathcal{L}$ to rectify the classifier significance during the fine-tuning:

$$\nabla_\theta \left({}^1\!/_n \sum \mathcal{L}(y, h_\theta(x)) + \sigma(x, \theta) \right) \tag{3.5}$$

Specifically, the regularization term $\sigma(x, \theta)$ is calculated based on the Hamming distance of the corresponding classifier's target codeword and its predicted result. Since the neural competition can be estimated from the occurrence of bit-flipping, after the fine-tuning, more significant classifiers may give more decisive confidence for decision making. Meanwhile, some classifiers may again become indecisive during the inference. Thus, we further set a pending zone in logistic regression to address this issue. As shown in Fig. 3.4, the pending zone is defined as a specific region:

$$h_\theta(x) = {}^1\!/_{1+\exp(-\theta^\top x)}\big|_{-0.4 \le \theta^\top x \le 0.4} \approx [0.4, 0.6] \tag{3.6}$$

Once the weighted input $\theta^\top x$ enters the pending zone, the classifier will report both $\{0, 1\}$ as its output. For example, given three collaborative logistic classifiers

Table 3.1: Experimental settings.

Environment		
CPU		Intel Core i7-6850K, 12 cores
GPU		GeForce GTX 1080, 2560 CUDA cores
Simulator		MATLAB, Deep Learning Toolbox
Network Model	**Dataset**	**Original Accuracy**
MLP [107]	Mnist	99.1%
LeNet [107]	Cifar-10	76.1%
Alexnet [50]	ImageNet	57.2%
Squeeze [57]	ImageNet	57.5%

with an input vector $\{-2, 0.1, 2\}$, the output vector(s) will be a 2-dimension matrix $\{0, 0, 1\}$ and $\{0, 1, 1\}$. Later, the Hamming distance of these two codewords will be compared with the entries in codeword list to predict the target class. Such a design may effectively rectify the wrong decisions caused by the less significant classifiers.

3.3 Evaluation

3.3.1 Experimental Setup

Baseline and Benchmark

Table 3.1 shows the details of our experimental environment, neural network models and datasets. We select four different neural network models, including small-scaled multi-layer perceptron (MLP) and popular convolutional neural networks (CNN) such as LeNet, Alexnet and Squeezenet, along with three datasets ranging from simple Mnist (10-class handwritten digits), Cifar-10 (10-class tiny images) and complex Imagenet (1000-class large images), so as to comprehensively validate the efficiency and scalability. We simulate a memristive accelerator similar to [37], wherein each layer of selected neural network model is mapped to one or more 128×128 arrays and each memristive cell maintains 64 quantization levels (6-bit) to achieve a good balance between throughput and reliability [4].

Error Modeling

Two different types of errors [22] are simulated in our evaluation: stochastic programming error (represented as *resistance variation* in this paper) and stuck-at fault (SAF). The resistance variation can be formulated as:

$$w' \leftarrow w \cdot e^{\theta} \text{ s.t. } \theta \sim N(0, \sigma^2) \tag{3.7}$$

where w' is the neural network parameters with programming errors under memristor resistance variation θ, which follows a log-normal distribution. In our simulation, we vary σ to change the level of resistance variation, so as to tune random programming error. The SAF occurs when a memristor device freezes in a low resistance state (LRS) or high resistance state (HRS), resulting in the stuck-at-one (SA1) fault or stuck-at-zero (SA0) fault. We adopt SA0 (SA1) fault rate as 1.75% (9.04%) based on the published data [22].

Experimental Method

We use the classification accuracy as the measurement metric for our proposed design [22]. The original accuracy (baseline, without considering device errors) of the accelerators implemented with the four selected neural network models under corresponding datasets, are reported in Table 3.1, serving as the upper bound of our fault-tolerant design. To characterize the lower bound of the accuracy, selected error models (both programming errors and SAFs) are first applied to the weights across all different layers in selected neural network models. We then further apply our proposed fault-tolerant architecture to each weight-distorted neural network model and measure its average accuracy. Monte-Carlo simulations, which perform 1000 times of testing for each combination of neural network model and error model, are conducted to calculate the average accuracy. Particularly, we set the number of collaborative logistic classifiers as $l = 7$ ($l = 500$) for the 10-class Mnist and Cifar-10 datasets (1000-class Imagenet dataset) in our evaluation.

3.3.2 Evaluation Results

Reliability

We first evaluate the classification accuracy of simulated memristive accelerators with error injection before and after applying our proposed methodology. Table 3.2 compares the classification accuracy of the four benchmarks across different levels of resistance variations (σ) with SAFs. Two different baselines, including original DNN models and our solutions (ours), are compared. With the increased resistance variation σ (i.e., stronger errors), the four original baselines suffer from severer accuracy degradation. In contrast, our method can always provide significant accuracy improvement in all cases. For example, the highest accuracy gap, i.e. $\sim 50\%$, $\sim 50\%$, $\sim 30\%$ and $\sim 30\%$ for MLP-Mnist, LeNet-Cifar10, Alexnet-Imagenet and Squeezenet-Imagenet, can be well maintained even for the largest variation $\sigma = 1.5$. Moreover, the accuracy degradation ($61\% \rightarrow 14\%$) on LeNet-Cifar10 can be reduced to merely $\leq 20\%$ after applying our solution, translating into $\sim 40\%$ accuracy improvement. A similar trend can also be found on more complex Alexnet-Imagenet (accuracy is improved by $\sim 30\%$). This is because our solution greatly unleashes the error correction potential for complex neural networks (like CNNs), which usually have much more parameters and better error-resilience capability. Moreover, the decision confusion caused by weights with bi-directional SAFs (LRS to HRS or HRS

Table 3.2: Classification accuracy (%) on selected baselines with resistance variation and SAFs by varying σ.

Resistance Variation (σ)	**0.0**	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**	**1.1**	**1.2**	**1.3**	**1.4**	**1.5**	**No Error**
MLP-Mnist	72.4	63.4	54.3	47.8	41.3	36.8	32.3	27.3	22.3	19.8	17.3	16.4	15.4	13.8	12.3	11.7	99.1
Ours (on MLP)	99.0	97.5	95.4	94.3	91.2	89.5	85.7	84.0	82.4	19.3	74.3	73.0	71.4	70.0	68.1	66.3	
LeNet-Cifar10	61.2	50.0	40.2	32.7	25.3	24.0	22.4	21.0	19.4	18.0	17.3	16.3	15.4	15.7	15.0	14.0	76.1
Ours (on LeNet)	74.3	74.0	72.1	71.0	69.1	70.0	70.3	69.0	67.1	64.0	60.4	60.0	58.7	57.5	56.4	54.3	
Alexnet-Imagenet	37.5	35.0	31.6	30.0	27.6	26.0	24.5	22.0	19.4	17.5	15.4	14.3	13.3	12.5	11.3	11.7	57.2
Ours (on Alexnet)	56.1	56.2	56.4	55.8	55.7	55.0	54.1	52.6	51.1	50.4	49.5	48.9	48.3	45.0	42.3	39.5	
SqueezeNet-Imagenet	30.4	27.8	25.3	24.0	22.8	19.5	16.7	15.4	14.2	13.6	13.1	12.2	11.2	10.6	10.1	9.9	57.5
Ours (on SqueezeNet)	54.6	53.5	52.3	51.5	50.7	49.0	48.1	47.7	47.3	46.0	44.7	43.5	42.3	38.9	35.4	32.1	

Table 3.3: Evaluation of overhead.

		MLP (10-class)	**Alexnet (1000-class)**
Original	**Parameters**	5.86 KB	3000 KB
Ours	**Parameters**	4.10 KB	1500 KB
	LUT	0.0085 KB	61.0352 KB
	Total	4.1085 KB	1561.0352 KB

to LRS) can be better alleviated by the optimized ECOC coding scheme. These results demonstrate that the proposed methodology can efficiently address the reliability issue of emerging accelerators in a scalable way, regardless of DNN model, dataset, and hardware error type.

Overhead

Table 3.3 further summarizes the additional overhead incurred by our solution. For a comparison purpose, the overhead of the original design is also reported. Since our proposed methodology only requires to replace the original softmax classifiers with collaborative ECOC classifiers at the output layer, we evaluate the storage overhead of the output layer, as well as the processing efficiency. Compared with the original baselines on MLP (Alexnet), we can observe that the storage cost is decreased by 30% (48%) with our solution. In particular, the required LUT in our solution introduces negligible overhead comparing with parameter storage overhead of a DNN model. Moreover, the storage requirement of weights in the last layer can be significantly reduced due to the reduced number of classifiers (i.e., 7 in MLP and 100 in Alexnet) in our design. The reduced computation requirement of proposed collaborative ECOC classifiers can further improve the processing efficiency, i.e., $\sim 1.2\times$ ($\sim 1.7\times$) on MLP (Alexnet).

Integration with Existing Solutions

For larger resistance variations ($\sigma > 1.2$), our design still gradually becomes less effective. This is consistent with previous works [22, 37], since the fault-tolerance capability can be eventually compromised by strong variations. However, as an orthogonal solution that well leverages the algorithmic fault-tolerance of neural network classifier, our design can be naturally integrated with existing solutions such as

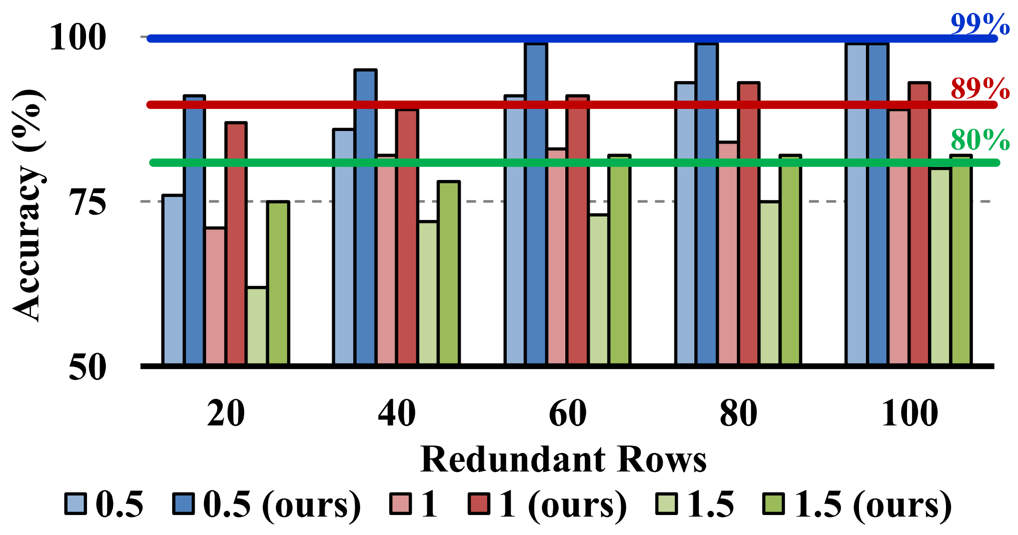


Figure 3.5: Working with bipartite-matching [22] on MLP-Mnist.

bipartite-matching [22], thus to further improve the robustness. Previous work [22] also shows that combining bipartite-matching and redundancy rows together can better handle programming errors and SAFs on memristive neural network accelerators. Here we integrate our technique into bipartite-matching, to investigate how much redundancy rows we can save for the same accuracy.

Fig. 3.5 shows the combined effectiveness of our design and bipartite-matching (named as "ours") on top of bipartite-matching only, against SAFs together with selected resistance variations (i.e., $\sigma = 0.5, 1, 1.5$) on a variety of designs with different number of redundant rows. The MLP-Minst design is selected. For each selected design, we can always further boost the accuracy, with more significant improvement on designs with fewer number of redundancy rows. For example, Our combined design improves the accuracy by 15%, 16% and 13% with $\sigma = 0.5, 1, 1.5$, respectively, for designs with 20 redundant rows, when compared with bipartite-matching. To show the improvement more clearly, we highlight the best accuracy at each σ offered by bipartite-matching with 100 redundant rows, i.e., 99% with $\sigma = 0.5$ (blue line), 89% with $\sigma = 1$ (red line) and 80% with $\sigma = 1.5$ (green line). With our

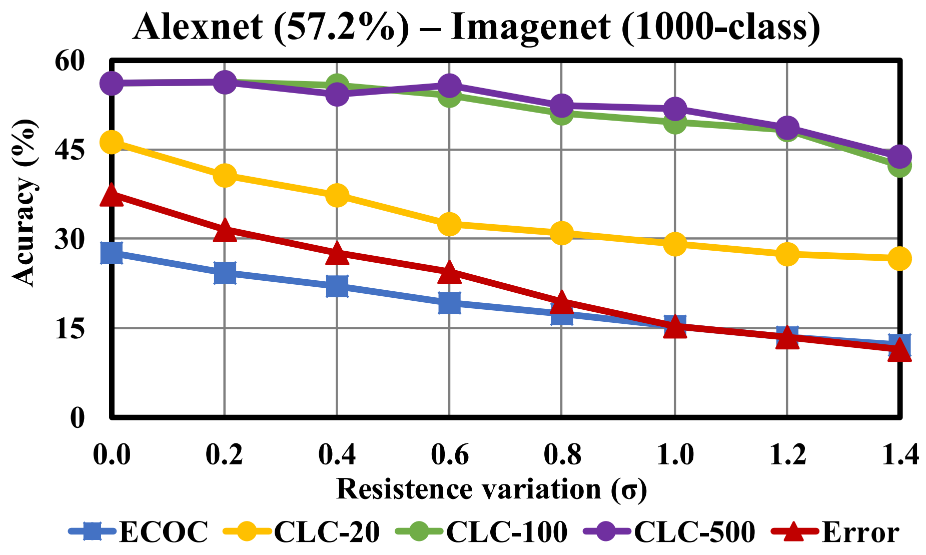


Figure 3.6: Comparison between ECOC and various CLC designs.

combined design, we save more than 50% of redundant rows, i.e., 50, 40 and 50 for $\sigma = 0.5, 1, 1.5$, respectively, in order to achieve the same high accuracy. These results further indicate the improved effectiveness and scalability of our design.

Flexibility

Since the collaborative logistic classifier incorporates variable-length coding scheme, Fig. 3.6 further evaluates the flexibility by comparing the accuracy of different design variants with the original ECOC design. The CLC-20 (CLC-100, CLC-500) design consists of 20 (100, 500) collaborative logistic classifiers to classify the 1000 classes in Imagenet dataset, while the ECOC directly uses Hamming code (16 binary classifiers) for classification.

As shown in Fig. 3.6, the ECOC is completely ineffective against the SAFs and resistance variation. In fact, the accuracy of ECOC on Alexnet is even worse than the modern softmax classifier with errors, due to the limited classification capability under significantly reduced number of classifiers (i.e., 16 in ECOC v.s. 1000 in softmax). In contrast, our CLC-20 significantly surpasses the ECOC, i.e., increasing the accuracy by 15%, even with only 20 collaborative logistic classifiers. By increasing the number of collaborative logistic classifiers, our design continues improving the classification accuracy (i.e. from CLC-20 to CLC-100) because the increased coding space can alleviate the conflict of coding for different classes. However, the error correction capability can still be saturated when reaching a sufficient number of classifiers, e.g. CLC-500 almost maintains the same level of accuracy as CLC-100.

3.4 Summary

Deep neural network (DNN) accelerators built upon emerging technologies, such as memristor, are gaining increasing research attention because of the impressive computing efficiency brought by processing-in-memory. One critical challenge faced by these promising accelerators is their poor reliability: the weight, which is stored as the memristance value of each device, suffers large uncertainty incurred by unique device physical limitations, translating into prominent testing accuracy degradation. Non-trivial retraining, weight remapping or redundant cell fixing, are popular approaches to address this issue. However, these solutions have limited scalability since they are more like tedious patch adding or bug fixing after identifying each accelerator-dependent defect map. In this work, we discuss the challenge and requirement of the fault-tolerance in these new accelerators. Then we show how to address this problem through a scalable algorithm-hardware co-design method, with a focus on unleashing the algorithmic error-resilience of DNN classifiers, so as to eliminate any expensive defect-map-specific calibration or training-from-scratch.

CHAPTER 4

MAL-DNN: MALICIOUS DNN-POWERED STEGOMALWARE

4.1 Preliminary

4.1.1 DNN Compression and Fine-tuning

DNNs usually employ different optimization techniques to lower the storage footprint, computation requirement and training cost. For example, deep compression techniques [50] can significantly compress the DNN model, by reducing the parameter precision or pruning network topology, thus achieving low-power, high-speed on-device computing without DNN accuracy degradation. Besides, fine-tuning [127] is another type of optimization techniques usually applied to pre-trained DNN models. Such a technique will only update the value of few DNN model parameters (usually in the last layer) to improve the decision making without expensive training. These techniques are now widely adopted to delivery the embedded DNN services [44, 8]. However, they will potentially compromise the integrity of injected payload injected in model parameters. In MAL-DNN, we take these concerns into consideration and develop the practical and reliable payload injection techniques.

4.1.2 Commercialized DNN and Security Concern

Developing a "Plug & Play" DNN model for a specific machine learning (ML) service is costly due to the long training time (i.e. months or more) over expensive hardware platforms with large-scale GPU-clusters and complex IP design, optimization and verification. Therefore, DNN models are usually first pre-trained and validated by service providers (ML experts), and then consumed by end users (non-ML experts).

The open machine learning marketplace [58] allows end users to quickly download, deploy and execute the pre-trained DNN models. Such an emerging business model has been accepted and discussed in many prior work [47, 82, 81, 110]. **However, the DNN models are programs, the behavior of DNN can be abused on modified model [117]. Moreover, the machine learning marketplace is still in its infancy and lacks security guarantee. Anonymous DNN models can be uploaded, distributed and eventually consumed by end users.**

Adversary Example and Poisoning Attack

Adversary example and poisoning attack are two types of well studied security problems in DNN. We briefly introduce them as they are less related to this work. Adversarial example [42, 97, 18] is created by adding a well-crafted small perturbation to benign input. It can easily mislead the classification result of a well-trained DNN. Poisoning attack [23, 60] can change the behavior of DNN with poisoned training data, thus to mislead the classification results. Their mitigation techniques are studied in many existing works [73, 102, 112, 119, 1].

Emerging DNN Threats

DNN backdoor is a specific classification logic in DNN and will only respond to a specific trigger pattern added to the input. Gu et al. [47] show the adversary can train the DNN backdoor with poisoned training data with applied arbitrary trigger pattern. The created backdoor in DNN will not impact the testing accuracy on benign inputs. Liu et al. [81] propose the DNN Trojaning attack (backdoor) by choosing a specific trigger pattern based on the estimation of confidence in DNN classification, thus to make the created backdoor more sensitive to the trigger. Such a backdoor can be created with few training data while achieving a high attack success rate.

DNN backdoor attacks mainly aim to achieve the targeted classification on a certain class with the trigger pattern. Our work shares several similar assumptions as DNN backdoor, and also adopts the concept of trigger. In a recent work [110], authors show that the malicious DNN training algorithm can stealthily memorize user secrets such as private training data, during the training phase. Different techniques such as LSB encoding, correlated value encoding and sign encoding are explored to encode the user secret into DNN model parameters, in both white-box and black-box scenarios.

4.1.3 Stegomalware and Steganalysis

Stegomalware [113, 86] is a type of malware that uses steganography [21] to hinder malicious intention. In stegomalware, the malicious code is usually concealed in covert file such as text and image, thus to circumvent detection. LSB replacement in image is the most popular approach used for creating the stegomalware [86], which can be conducted in both spatial (i.e., raw data) and frequency (DCT in JPEG) domain. A daemon process is running on background to extract and execute the malicious code dynamically based on the trigger condition. Steganalysis [72] is the art of deterring covert information against steganography, which can be used to detect the stegomalware. For example, Primary Sets [31], Sample Pairs [32], Chi Square [123], RS Analysis [40] are several classic steganalysis approaches to detect the image based LSB steganography in spatial domain. Primary Sets [31] and Chi Square [123] detect the statistical identity of neighboring pixels, and pairs of values (PoV) exchanged during LSB embedding. Sample pairs [32] and RS analysis [40] can further detect and trace randomly scattered LSB and bit flipping. Fusion is a more powerful ensemble technique based on multiple spatial classifiers.

Table 4.1: Comparison on attack type, assumption, approach and objective with related works. The bold description highlights the similarity and the italic description highlights the difference on approaches.

	Type	**Assumption**	**Approach**	**Objective**
Our "MAL-DNN"	DNN Stegomalware	Adversary provides trained DNN model; Adversary controls the model creation; User is non-ML expert; Adversary can NOT control user's isolated secured environment.	*Embed payload into DNN model parameters;* *Use DNN output logits as trigger;* **DNN testing process is modified to monitor trigger event.**	Execute malicious payload with trigger during testing.
[47, 81]	DNN Backdoor	Adversary provides DNN model or training data; Adversary controls the model creation; User is non-ML expert.	*Create specific pattern on training image as trigger;* Train a specific DNN testing logic as backdoor; Backdoor only responses to trigger.	Misclassify arbitrary input with trigger applied during testing.
[110]	DNN Malicious Training	Adversary provides DNN training algorithm to user; Adversary controls the model creation; User is non-ML expert.	**DNN training process is modified;** User's training data is illegally gathered during training, and is embedded into DNN model.	Steal user's training data.
[113]	Traditional Stegomalware	Adversary provides the malicious, and disguised wrapper application to user.	*Embed payload into images;* **Wrapper application monitors the input image, and executes the embedded code.**	Execute malicious payload.

4.2 Threat Model

Our threat model is partially similar to and extended from the related works including traditional DNN threats [47, 81, 110] and stegomalware [113].

Table 4.1 compares the assumption, approach and objective of our work with related works. Our assumption is similar to the DNN backdoor [47, 81] and DNN malicious training [110], while our objective is similar to the traditional stegomalware [113]. In particular, our approach is extended from DNN malicious training and stegomalware; however, with a different trigger design compared with DNN backdoor. Our comprehensive threat model is defined as follows:

End user. End user is the non-ML expert who consumes DNN services. It is common for an non-ML expert to consume DNN services from the third-party without understanding the end-to-end DNN process on data, training, and testing. Instead, end user mainly cares about service quality (i.e., DNN testing accuracy). **We assume that end user will deploy the DNN service in a private secured computing environment which is isolated and secured with firewall, anti-malware, and steganalysis defense techniques, etc.**

Adversary. We assume the adversary is an anonymous DNN service provider who creates the malicious DNN (i.e., self-contained stegomalware) which will be disguised as normal DNN service and deployed on end user's side. **The adversary is unable to directly access, modify or control the end user's secured computing environment via traditional cyber-channel (i.e., Internet), except for the physical-channels (i.e., sensors, cameras, etc.) normally used for retrieving DNN service input from physical world.**

Adversary's goal. The adversary's goal is to run the malicious payload code in MAL-DNN on user's side. In particular, based on our assumption that the MAL-

DNN will be eventually deployed in user's isolated environment, adversary should make MAL-DNN a self-contained malicious DNN model. To achieve this goal, adversary should consider following objectives step by step: 1) maintain the DNN service quality on created MAL-DNN to avoid the service rejection, therefore to disguise it as a normal DNN service; 2) ensure the MAL-DNN can circumvent existing countermeasures and survive in end user's secured environment; 3) trigger and run the malicious payload along with a normal DNN service through a physical-channel while maintaining the minimized footprint.

Adversary's approach. To create the **self-contained** malicious DNN model, adversary **can only** modify the DNN model (includes model parameter and testing algorithm) at the service creation phase and cannot touch user's physical devices that will execute this DNN service. Once the malicious DNN model is accepted and deployed by end user, the adversary cannot communicate directly with the testing environment. Adversary's approach consists of embedding and triggering.

- **Embedding.** To embed the payload code into DNN model parameters, adversary can leverage the naive LSB manipulation and proposed training or mapping based techniques which maintain DNN service quality and circumvent existing defense. DNN is inexplicable and consists of a large number of parameters. It is very difficult to identify the exact meaning of these parameters. By leveraging its structural complexity [68, 114] and error-resilience property [50, 71], such a self-contained stegomalware can be more evasive.
- **Triggering.** The idea behind the triggering on embedded payload code is to monitor the DNN output logits by modifying DNN testing function. As we summarized in Table 4.1, the monitoring based trigger and modified DNN training algorithm are the two common approaches adopted in traditional stegomalware [113] and emerging DNN threat [110]. In our proof of concept design, we combine these

approaches to create the trigger and execute the payload in DNN output layer by monitoring the DNN logits during the testing process.

4.3 Create the MAL-DNN

In this section, we first give an overview of our proof of concept design. Then we present the design details of proposed payload injection and trigger techniques.

4.3.1 Overview of Proof-of-Concept Attack

As shown in Fig. 4.2, a DNN can be turned into a self-contained stegomalware–MAL-DNN through the following steps:

1. **Prepare DNN model.** Adversary can either train the DNN model from scratch or obtain DNN model from the machine learning marketplace [58] or DNN model zoo [16];
2. **Prepare malicious payload.** Adversary can either directly use many existing payloads for different purposes (e.g., forkbomb, keylogger, etc.), or create new malicious payloads as needed. This part is not the focus of this work.
3. **Inject payload.** The malicious payload will be injected into DNN model through our proposed payload injection techniques, without impacting service quality (i.e. similar to the "untouched" model), including highly compressed DNN models tailored for ever-increasing resource-limited embedded, IoT and mobile devices.
4. **Create trigger.** The trigger is created to control the execution of embedded payload under a certain condition. In MAL-DNN, we use real-world objects as the trigger event and propose DNN logits based trigger designs to handle the

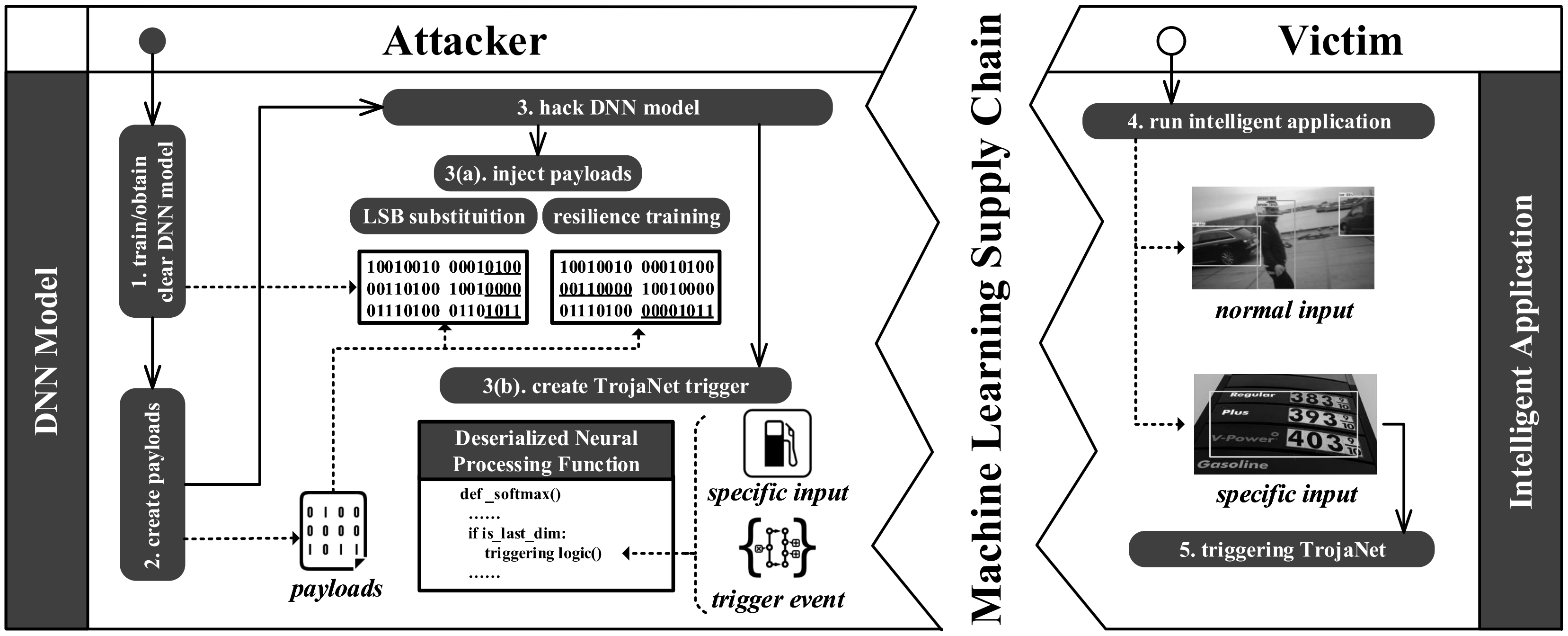


Table 4.2: Overview of MAL-DNN attack.

Table 4.3: Redundancy in uncompressed/compressed DNNs.

Uncompressed DNN Models				
	AlexNet [66]	**GoogLeNet [115]**	**VGG-16 [108]**	**ResNet-50 [51]**
# Layers	8	22	16	50
# Parameters	61M	7M	138M	25M
Model Size	227MB	27MB	515MB	96MB
# Redundant Bits	21	20	19	16
Redundant Space	152MB	16MB	312MB	47MB

Hardware-favorable Compressed DNN Models				
	Comp.AlexNet [50]	**Comp.VGG-16 [50]**	**Mobilenet [53]**	**Squeezenet [57]**
# Layers	8	16	28	18
# Parameters	6.97M	11.26M	4.2M	1.24M
Model Size	6.63MB	10.78MB	4.2MB	4.6MB

potential input variations. Our proof of concept design will modify the DNN algorithm to execute the payload by monitoring the DNN logits.

4.3.2 DNN Favorable Payload Injection

In MAL-DNN, different types of payload injection techniques are proposed to inject the malicious payload into DNN model parameters. Proposed payload injection techniques are required to either secure the DNN testing accuracy or protect the payload integrity, while maintaining a good scalability for handling different types of DNN models.

Investigate Model Capacity and Naive LSB Method

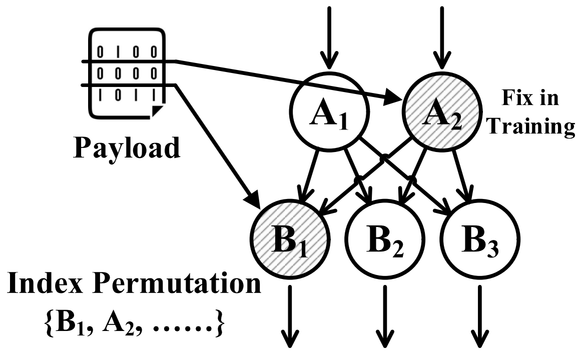
To find the appropriate solutions, we investigated different types of mainstream DNN models, including both uncompressed and compressed DNN models, by measuring their model size and redundancy (i.e., the maximum capacity for payload injection without accuracy loss). As shown in TABLE 4.3, all the uncompressed DNN models can provide a considerable scale of redundancy ($\geq$10 MB) to accommodate most realistic malicious codes [92] ($\sim$100 KB), without impairing the DNN testing accuracy. With sufficient redundancy in these DNN models, a simple solution “LSB substitution” can be applied to handle the payload injection by replacing the least significant bits (LSB) of DNN model parameters with the payload binary. However, such a naive solution is incompatible with the compressed DNN model. As shown in TABLE 4.3, the size of compressed DNN models has been aggressively shrunk by reducing both the amount and data precision of model parameters. For example, the size of MobileNet [53] is only 4MB with 8-bit precision on 4M parameters. These compressed DNN models are unable to maintain the accuracy even with a slight modification on parameters.

Proposed Payload Injection Methods

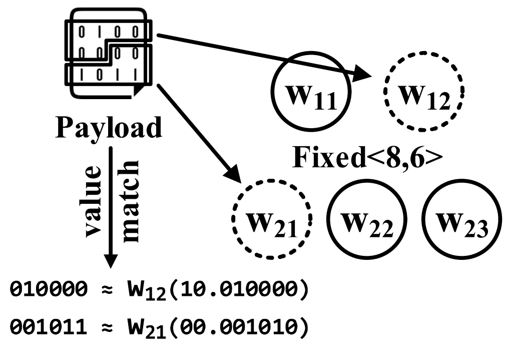
To overcome this issue, we propose several enhanced payload injection techniques for compressed DNN models, so as to improve the efficiency of payload injection and protect the integrity of injected payload. Fig. 4.4 shows the basic idea of proposed “resilience training”, “value-mapping” and “sign-mapping” techniques.

Resilience training. DNN model is intrinsically error resilient and can self-repair from the internal errors. For example, removing a bundle of neurons from the DNN topology can cause significant accuracy degradation. However, parameters on the remaining neurons can be rebuilt to reach the original accuracy after re-training. Based on this intuition, we propose the resilience training technique.

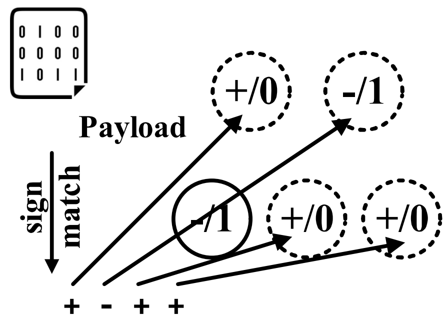
As shown in Fig. 4.4(a), resilience training can be conducted with following detailed steps: i) Calculate the required number of DNN parameters, i.e. $n = \lceil {}^{P}\!/_{q} \rceil$, based on the size of payload P and the quantized bit width q of parameters; ii) Generate the “index permutation” randomly in order to select the n parameters; iii) Assign the value of payload segment to each selected parameter by following the sequence in “index permutation” (i.e., $\{B_1, A_2, \cdots\}$ in Fig. 4.4); iv) Train the DNN model while fixing the values of those selected parameters.



(a) Resilience training.



(b) Value-mapping.



(c) Sign-mapping.

Table 4.4: Illustration of proposed resilience training, value-mapping and sign-mapping techniques. (a) Resilience training select a bundle of model parameters with randomly generated index permutation. The value of selected parameters is replaced by the payload segments, and will never be updated during the re-training. After the re-training, the accuracy of DNN model with embedded payload is expected to be recovered to the original level; (b) In value-mapping, we conduct an exhaustive searching to match (or replace) the same (or nearest) value of parameters with payload segments; (c) Sign-mapping goes through the model parameters and match the parameter sign bit (+ for 0 and - for 1) with every single bit in the given payload.

Resilience training will intentionally introduce internal errors in model parameters by directly replacing the entire bits on selected parameters with the payload segments. Such "broken neurons" (i.e., replaced parameters) will never be updated during the re-training. After resilience training, the accuracy of DNN model is expected to be recovered, thus successfully concealing the injected payload while maintaining the DNN service quality. In particular, an "index permutation" will be randomly generated to indicate the selected parameters for payload injection. To restore the payload, we combine the binary segment of each parameter sequentially in the "index permutation".

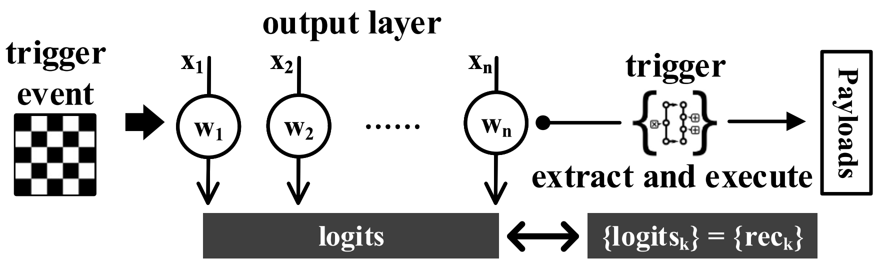
Searching and mapping. "Value-mapping" and "Sign-mapping" inject the payload into compressed models through dedicated "searching and mapping" rules, and improve the efficiency of payload injection by eliminating the unavoidable re-training process in "resilience training". Fig. 4.4(b) and (c) show the basic idea of value-mapping and sign-mapping.

In value-mapping, we first split the payload binary based on the fractional precision of quantized DNN model parameters. For example, "Fixed<8,6>" indicates a 8-bit fixed-point number with 6 fractional bits on each DNN model parameter. Therefore the payload binary will be divided into many 6-bit segments. Then, for each payload segment, we conduct an exhaustive searching on model parameters to match (or replace) the same (or nearest) value of fractional bits of parameters. As the example shown in Fig. 4.4(b), given payload segment "010000" (or "001011"), parameter w_{12} (or w_{21}) is matched since the value of fractional bits "010000" (or "001010") is same as (or nearest to) the payload segment. Finally, we map the payload segment(s) to matched parameter(s) by replacing the fractional bits of parameter(s) with payload segment(s). Note that the parameters in well-trained DNN model are usually scaled between +1 and -1. Therefore, we use the fractional bits in value-mapping. The sign-mapping technique adopts a similar “searching and mapping” rule, based on the sign bit of model parameters. As shown in Fig. 4.4(c), sign-mapping will go through the model parameters and match the parameter sign bit with every single bit in the given payload, thus eventually mapping the payload to a sign bit(s) sequence on matched parameters.

4.3.3 DNN Logits based Trigger

Why do we use the DNN output logits to create the trigger?

Before we present our trigger design, we first show our investigation and explain the reason we use the DNN output logits to create the trigger. To design the trigger in MAL-DNN, we investigated the existing approach of using DNN input (i.e., image pattern) as a trigger in DNN backdoor[47, 81]. We find that the DNN input captured by sensors usually suffer from input variations due to the noises from the physical world. The DNN input based trigger is not a reliable solution to handle this issue. In contrast, the DNN output logits in the last layer can provide a more reliable solution (i.e., logits rank) to handle the input variations. Besides, the DNN input pattern is usually more complicated than that of the DNN output logits for modern DNN services. Let us take the widely adopted Imagenet [28] classification as an example, the data dimension of DNN input (i.e., 227-pix$\times$227-pix$\times$3-color) is 154$\times$ larger than that of the DNN output (i.e., 1K output logits). Therefore the logits based trigger can be more efficient than that of DNN input. Moreover, the final classification result is naturally calculated by comparing the value of DNN output logits, which is an essential process housed in the DNN output layer. This can help to reduce the footprint of logits based trigger.



(a) Logits trigger.

Logits	Record	Present	Record	Present
p_1	0.5	0.55	p_1	p_1
p_2	0.2	0.13	p_3	p_3
p_3	0.3	0.32	p_2	p_2
	Logits (unmatched)		Logits Rank (matched)	

(b) Rank trigger.

Table 4.5: Illustration of proposed DNN logits based trigger techniques. (a) Logits trigger will monitor and compared the present logits value with the recorded value. A perfect match will trigger the MAL-DNN; (b) Rank trigger will monitor and compare the rank of logits instead of their values.

Proposed Triggering Methods

In MAL-DNN, we propose several different trigger techniques, including the basic "logits trigger", and more reliable "rank trigger" and "fine-tuned rank trigger" to monitor the trigger event by assessing the DNN logits in output layer.

Logits Trigger. Fig. 4.5(a) shows the basic idea of logits trigger, which can be explained as a key-lock problem. Given the trigger event x' as the key, DNN logits $\{logits_k\} = \{\exp(\theta^{(K)\top} x')\}$ will be stored as the lock $\{rec_k\}$ in MAL-DNN. Such a key-lock pair, i.e., $\{rec_k\}^{lock} = \{logits_k\}^{key}$, will be created by adversary in creation stage. After the deployment of MAL-DNN, the present logits $\{logits_k\}^{pre}$ for given DNN input x will be monitored and compared with the recorded $\{rec_k\}^{lock}$ as long as there comes a new input at the DNN execution stage. MAL-DNN can be triggered once the key-lock pair is perfectly matched as $\{logits_k\}^{pre=key} = \{rec_k\}^{lock}$.

Rank Trigger. The rank trigger is extended from the basic logits trigger. Since the input variations from physical world can significantly reduce the possibility of the "perfect match" in logits trigger, we further propose the rank-trigger to handle this issue. Fig. 4.5(b) shows the idea of a rank trigger. Instead of using the logits value in key-lock pair, rank trigger uses the rank of logits to create the key-lock pair. As the example in Fig. 4.5(b) shows, given a 3-dimension logits, logits trigger will store the key-lock pair as $\{p_1, p_2, p_3\} = \{0.5, 0.2, 0.3\}$. However, due to the input variations, present logits always give the inconstant value as $\{0.55, 0.13, 0.32\}$, thus the key-lock pair is always mismatched. To solve this issue, the rank trigger will use the logits rank, i.e., $r = \{p_1, p_3, p_2\}$, as the key-lock pair. Even with input variations, the present rank of logits can be still matched, thus to successfully trigger MAL-DNN.

Fine-tuned Rank Trigger. Although rank trigger improves the reliability of MAL-DNN against input variations from physical world, we find that it is still less reliable to handle strong input variations. Therefore, we propose the fine-tuned rank trigger to further enhance the trigger reliability. Such an enhanced design is extended from the rank trigger and inspired from the fine-tuning techniques. We first create a small set of augmented inputs by applying simulated strong variations on the original specific input. Then, we use these augmented inputs to fine-tune the DNN parameters in output layer thus to improve the trigger reliability against strong variations. Instead of minimizing the loss of logits value adopted in traditional training, we minimize the **loss on logits rank**. However, assessing such a "rank based loss" is not practical. To solve this issue, we use a hard coded label $\vec{h^r}$ to define the selected logits and the expected logits rank for augmented inputs $\vec{x'}$. In particular, the used elements in hard coded label $\vec{h^r}$ are defined as an ranked arithmetic sequence with sum = 1, and the unused logits are all set to 0. For example, to train the expected rank {1, -, 3, 2} with 4 logits, we set $\vec{h^r} = \{0.5, 0, 0.17, 0.33\}$. Accordingly, the loss function can be further translated into:

$$\arg\min_{w} \; {}^{1}\!/_{n} \sum_{1}^{n} \mathcal{L}(f_w(\vec{x'}), \vec{h^r}) \tag{4.1}$$

for fine-tuning the model parameter in DNN output layer, so as to improve the possibility of matching the expected logits rank.

4.4 MAL-DNN Prototype

We implement a prototype of MAL-DNN for demonstration and evaluation purposes. Table 4.6 shows our prototyping environment. The MAL-DNN is created on a local server, and is deployed on the isolated Nvidia Jetson TX2 platform to simulate an end-to-end attack scenario.

Table 4.6: Prototyping environment.

Nvidia Jetson TX2 (Isolated Secured Environment)			
Computing Framework	**DNN Inference Runtime**	**DNN Library**	**API**
JetPack 3.1	Tensorflow/Nvidia CUDA	Nvidia cuDNN	Python/C++ (MATLAB Code Generation)
Computing Substrate	**CPU**	**GPU**	**Memory**
Tegra X2	Nvidia Denver2, Cortex-A57	Embedded, 256 CUDA cores	LPDDR4, 8GB, 128bit, 58.4 GB/s
Local Server			
DNN Engine	**CPU**	**GPU**	**Memory**
MATLAB Deep Learning Toolbox	Intel Core i7-6850K, 12 cores	2 * GeForce GTX 1080, 2560 CUDA cores	64 GB Main Mem, 16 GB Graphic Mem

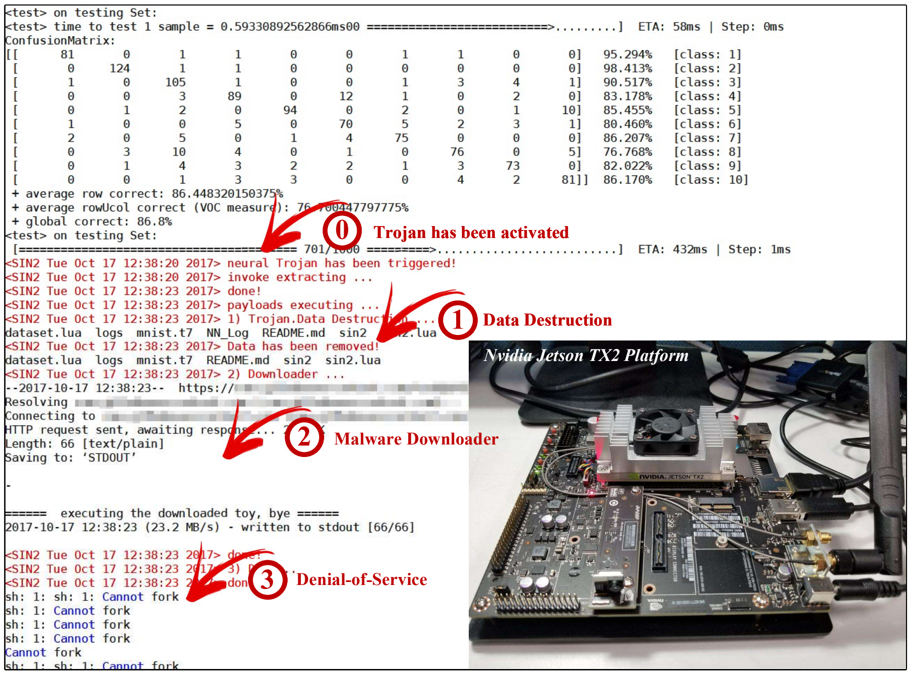
Table 4.7: CVEs and risks in DNN software.

Common Vulnerabilities and Exposures				**Other approaches**	
DNN Software	**Component**	**CVE ID**	**Type**	**Component**	**Type**
Jetson TX2	Kernel	CVE-2018-6269	execution	Python	Insecure API
Jetson TX2	Kernel	CVE-2018-6269	execution	Javascript	Obfuscation
Caffe/Torch	Libjasper	CVE-2017-9782	Overflow	JSON	
Caffe/Torch	OpenCV	CVE-2016-1516	execution	XML	Insecure
Tensorflow	Numpy	CVE-2017-12852	DOS	Pickle	Deserialization
Tensorflow	Wave	CVE-2017-14144	DOS	Protocol Buffer	

4.4.1 Implementation

Our implementation in such an end-to-end scenario consists of following procedures: 1) We create clear DNN models, including MobileNet [53] (trained with MNIST dataset [70]) and Alexnet [66] (trained with Imagenet dataset [28]) on a local server; 2) We assemble malicious binary malwares such as fork bomb and downloader [85, 84] as MAL-DNN payload; 3) We embed the combined payload into DNN model through proposed payload injection techniques, and create the triggers by using the "chessboard" input (see Fig. 4.5) for demonstration purpose (note that MAL-DNN allows adversary to use arbitrary input to create the trigger); 4) We create an obfuscated monitor function along with DNN testing process by modifying the Softmax function to extract and execute the payload; 5) We transfer and deploy the modified DNN models (as an end user) to Jetson TX2 platform to demonstrate MAL-DNN in two different image classification cases, including **i) the ideal test image in the laboratory environment** and **ii) the camera-captured test image in physical-world**.

Alternative approaches to execute the payload. The MAL-DNN is actually a program [117], which combines data (DNN parameters) and functions (testing algorithms). In our proof of concept implementation, we follow the similar approaches from traditional stegomalware [113] and malicious DNN training [110] to create the self-contained malicious model by modifying the DNN Softmax function. Alternatively, adversary can exploit different components in DNN to execute the payload. Table 4.7 lists some examples of DNN related Common Vulnerabilities and Exposures (CVEs) on different platforms that can be exploited to abuse the DNN behaviors. Besides, traditional malware techniques such as code obfuscation can be also applied to protect the data execution [113, 86]. Since this part is not our focus in MAL-DNN, we only show the possibility of the alternative approaches and use the modified Softmax function for demonstration purpose.



```
<test> on testing Set:
<test> time to test 1 sample = 0.59330892562866ms00 ==========================>.........]  ETA: 58ms | Step: 0ms
ConfusionMatrix:
[[      81       0       1       1       0       0       1       1       0       0]   95.294%   [class: 1]
 [       0     124       1       1       0       0       0       0       0       0]   98.413%   [class: 2]
 [       1       0     105       1       0       0       1       3       4       1]   90.517%   [class: 3]
 [       0       0       3      89       0      12       1       0       2       0]   83.178%   [class: 4]
 [       0       1       2       0      94       0       2       0       1      10]   85.455%   [class: 5]
 [       1       0       0       5       0      70       5       2       3       1]   80.460%   [class: 6]
 [       2       0       5       0       1       4      75       0       0       0]   86.207%   [class: 7]
 [       0       3      10       4       0       1       0      76       0       5]   76.768%   [class: 8]
 [       0       1       4       3       2       2       1       3      73       0]   82.022%   [class: 9]
 [       0       0       1       3       3       0       0       4       2      81]]  86.170%   [class: 10]
 + average row correct: 86.448320150375%
 + average rowUcol correct (VOC measure): 76.700447797775%
 + global correct: 86.8%
<test> on testing Set:
 [======================================== 701/1000 =========>.........................]  ETA: 432ms | Step: 1ms
<SIN2 Tue Oct 17 12:38:20 2017> neural Trojan has been triggered!
<SIN2 Tue Oct 17 12:38:20 2017> invoke extracting ...
<SIN2 Tue Oct 17 12:38:23 2017> done!
<SIN2 Tue Oct 17 12:38:23 2017> payloads executing ...
<SIN2 Tue Oct 17 12:38:23 2017> 1) Trojan.Data Destruction ...
dataset.lua  logs  mnist.t7  NN_Log  README.md  sin2  sin2.lua
<SIN2 Tue Oct 17 12:38:23 2017> Data has been removed!
dataset.lua  logs  mnist.t7  README.md  sin2  sin2.lua
<SIN2 Tue Oct 17 12:38:23 2017> 2) Downloader ...
--2017-10-17 12:38:23--  https://[redacted]
Resolving [redacted]
Connecting to [redacted]
HTTP request sent, awaiting response... 200 OK
Length: 66 [text/plain]
Saving to: 'STDOUT'

-

======  executing the downloaded toy, bye ======
2017-10-17 12:38:23 (23.2 MB/s) - written to stdout [66/66]

<SIN2 Tue Oct 17 12:38:23 2017> done!
<SIN2 Tue Oct 17 12:38:23 2017> 3) D...
<SIN2 Tue Oct 17 12:38:23 2017> done
sh: 1: sh: 1: Cannot fork
sh: 1: Cannot fork
sh: 1: Cannot fork
sh: 1: Cannot fork
Cannot fork
sh: 1: sh: 1: Cannot fork
```

Figure 4.1: MAL-DNN prototype on Nvidia Jetson TX2 platform.

4.4.2 Demonstration

CASE 1: Laboratory Configuration

Fig. 4.1 demonstrates the success of MAL-DNN using the ideal test data under the laboratory configuration. In particular, we adopt the simple LSB substitution and logits trigger in this case. At checkpoint-0, the MAL-DNN has been triggered to extract and execute the payload sequentially. First, user data is deleted on file system. Then, a piece of remote code is executed locally by the "downloader". Eventually, the "forkbomb" payload is invoked to halt the system, resulting in the DoS attack.

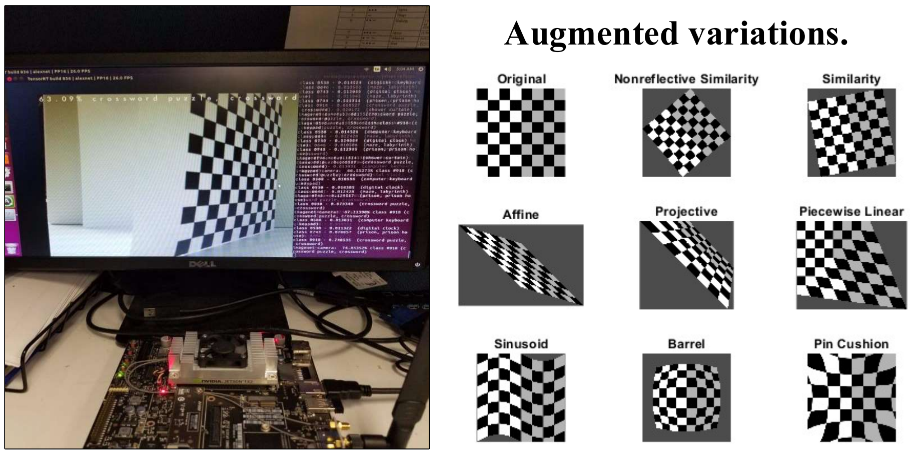


Figure 4.2: Physical-world configuration with augmented input variations.

CASE 2: Physical-world Configuration

As shown as Fig. 4.2, the object (i.e. a printed "chessboard" image) from physical-world is captured by on-board camera in this case to test MAL-DNN with rank trigger and fine-tuned rank trigger under different input variations, i.e., different camera angles, image rotations, distances and brightness. Meanwhile, a set of augmented "chessboard" images with strong simulated variations have been tested as well. In this demonstration, we observe that the logits trigger is completely invalid for the real-world inputs because the logits values keep changing when the camera captured images are subject to the noises from the physical world. In contrast, the rank trigger is more reliable for the real-world input as the logits rank can be still maintained under small input variations. However, as the variation strength increases, rank trigger becomes less effective than fine-tuned rank trigger due to significantly biased logits rank.

4.5 Evaluation

In this section, we evaluate MAL-DNN from different aspects, including **evasiveness**, **efficiency** and **robustness** of payload injection techniques, as well as the **reliability** of trigger techniques. **In our discussion, we try to answer following questions with our evaluation results**:

- *Why do we separate the payload embedding from its execution in MAL-DNN?*
- *How does MAL-DNN compare with traditional malware and stegomalware?*
- *How to select an appropriate payload injection technique?*
- *Why is rank trigger more reliable under input variations?*

Table 4.8: Additional DNN models used in evaluation.

DNN	**Size**	**#Para.**	**DNN**	**Size**	**#Para.**
Squeezenet [57]	4.6MB	1.24M	Googlenet [115]	27MB	7M
Resnet18 [51]	44MB	11.7M	Densenet201 [56]	77MB	20M
Inceptionv3 [116]	89MB	23.9M	Resnet50 [51]	96MB	25.6M
Resnet101 [51]	167MB	44.6M	Alexnet [66]	227MB	61M
Vgg16 [108]	515MB	138M	Vgg19 [108]	535MB	144M

Table 4.9: Selected malware samples from Malware DB [92].

Malware	**Size**	**Malware**	**Size**
Stuxnet	0.02MB	ZeusVM	0.05MB
Destover	0.08MB	Asprox	0.09MB
Bladabindi	0.10MB	EquationDrug	0.36MB
ZeusVM-decypted	0.40MB	Kovter	0.41MB
Cerber	0.59MB	Ardamax	0.77MB
NSIS	1.70MB	Kelihos	1.88MB
Mamba	2.30MB	WannaCry	3.35MB
Vikinghorde	7.08MB	Artemis	12.75MB

4.5.1 Experimental Setup

We use the our prototype as our evaluation platform. For a comprehensive evaluation of MAL-DNN, 13 state-of-the-art DNN models and 16 malware samples from Malware DB [92] with different sizes are selected. The details are listed in Table 4.3, Table 4.8 and Table 4.9. We embed these binary malware samples into DNN models through four different payload injection techniques to generate a set of MAL-DNN samples (836 in total) under appropriate size constraint (i.e., embedded malware is smaller than DNN model).

4.5.2 Evasiveness

Metrics and Methods

The evaluation on evasiveness indicates how can MAL-DNN be successfully deployed and survived in end user's secured environment, which can be measured from three different aspects:

- **Testing accuracy** is given first priority by end users. MAL-DNN should maintain a level of accuracy similar to clean model to prevent service rejection at the beginning. **A higher testing accuracy indicates better evasiveness**.
- **Anti-malware detection rate** shows to what extent the embedded payload can be detected by anti-malware. This is a naive evasiveness measurement directly reported by commercial anti-malware engines. **A lower anti-malware detection rate indicates better evasiveness**.
- **Steganalysis detection rate** measures the probability of detecting concealed payload in MAL-DNN samples by using steganalysis methods, given that the payload injection to DNN model can be treated as a specific spatial steganography,

which is similar to traditional stegomalware. **A lower steganalysis detection rate indicates better evasiveness**.

To measure the testing accuracy, we evaluate created MAL-DNN samples on Imagenet dataset, and compare the testing accuracy with the original accuracy of clear DNN models. To measure the anti-malware detection rate, we test selected MAL-DNN samples and two baselines–*vanilla malware* and *stegomalware* through 37 leading anti-malware engines on Metadefender [88] such as McAfee, Avira, etc., and compare the reported detection rate. The *vanilla malware* is directly selected from Malware DB [92] while *stegomalware* is created through LSB OpenStego [121] by embedding malwares into grayscale cover images. To measure the steganalysis detection rate, we test selected MAL-DNN samples and stegomalware with five classic steganalysis methods (i.e., Primary Sets [31], Sample Pairs [32], Chi Square [123], RS Analysis [40] and Fusion [63]) in StegExpose [10] tool. Primary Sets and Chi Square detect the statistical identity of neighboring pixels and pairs of values (PoV) exchanged during LSB embedding. Sample pairs and RS analysis can further detect and trace randomly scattered LSB and bit flipping. Fusion is a more powerful ensemble technique based on multiple spatial classifiers. To make StegExpose compatible with DNN model, we slightly modify the data acquisition interface by reshaping the data structure of DNN model as the grayscale image. Benign samples (i.e., clear images and DNN models) are added for steganalysis classifier to match the number (1:1) of created stegomalware and MAL-DNN samples.

Testing Accuracy

Table 4.10 and Table 4.11 report the testing accuracy of different DNN models before and after embedding various malwares using techniques like LSB substitution/resilience training, and value/sign mapping, respectively. The dash-line indi-

Table 4.10: Testing accuracy ($< \pm 1\%$) on selected MAL-DNN samples created by LSB Substitution and Resilience Training, with highlighted significant accuracy reduction.

LSB Substitution

		Original	EquationDrug	ZeusVM-decypted	Cerber	Ardamax	NSIS	Kelihos	Mamba	WannaCry	Vikinghorde	Artemis
Medium DNNs	Resnet50	75.2%	74.6%	75.7%	75.5%	75.8%	74.9%	74.5%	76.1%	75.6%	75.3%	74.7%
	Inceptionv3	78%	78.2%	77.9%	76.8%	76.3%	78%	77.2%	78.3%	78.2%	78.1%	77.3%
	Densnet201	77%	77.3%	77.1%	76.5%	77.3%	75.9%	76.3%	77.6%	77%	76.4%	77.1%
	Resnet18	70.7%	69.3%	71.2%	71.1%	70.2%	70.5%	71.6%	72.1%	71.3%	69.3%	**61.3%**
Small DNNs	Googlenet	69.8%	68.4%	68.1%	70.3%	70.8%	69.7%	69.4%	68.7%	69%	**58.1%**	**55.3%**
	Comp.VGG-16	70.5%	**51.6%**	**32.1%**	**17.3%**	**7.5%**	**0.9%**	**0.1%**	**0.2%**	**0.1%**	**0.1%**	-
	Comp.Alexnet	57%	**31.2%**	**17.6%**	**0.2%**	**0.1%**	**0.1%**	**0.1%**	**0.1%**	**0.1%**	-	-
	Squeezenet	57.5%	**0.7%**	**0.3%**	**0.2%**	**0.1%**	**0.1%**	**0.1%**	**0.2%**	**0.1%**	-	-
	Mobilenet	70.9%	**0.2%**	**0.2%**	**0.1%**	**0.1%**	**0.2%**	**0.1%**	**0.2%**	**0.1%**	-	-

Resilience Training

		Original	EquationDrug	ZeusVM-decypted	Cerber	Ardamax	NSIS	Kelihos	Mamba	WannaCry	Vikinghorde	Artemis
Medium DNNs	Resnet50	75.2%	75.2%	75.4%	74.8%	74.7%	75.1%	75.3%	74.6%	74.8%	75.5%	75.1%
	Inceptionv3	78%	78.3%	78.4%	78.2%	77.9%	78.4%	78.1%	77.6%	78.4%	77.8%	78.1%
	Densnet201	77%	77.2%	76.7%	77.1%	76.9%	77.3%	76.5%	77.2%	77.3%	76.7%	76.4%
	Resnet18	70.7%	71.1%	71.2%	70.9%	71%	70.4%	70.3%	70.9%	71.3%	68.2%	69.7%
Small DNNs	Googlenet	69.8%	70.3%	69.2%	69.6%	71%	70.5%	69.3%	70.4%	70.2%	70.4%	68.2%
	Comp.VGG-16	70.5%	68.3%	69.1%	71.2%	69.1%	68.4%	**63.4%**	**56.1%**	**22.6%**	**6.7%**	-
	Comp.Alexnet	57%	55.4%	56.7%	57.2%	54.1%	**38.2%**	**34.3%**	**16.7%**	**3.9%**	-	-
	Squeezenet	57.5%	56.8%	54.3%	53.2%	**48.3%**	**35.4%**	**29.6%**	**15.1%**	**4.1%**	-	-
	Mobilenet	70.9%	71.2%	68.5%	66.7%	**54.4%**	**32.5%**	**29.1%**	**6.1%**	**0.7%**	-	-

Table 4.11: Testing accuracy ($< \pm 1\%$) on selected MAL-DNN samples created by Searching and Mapping based technique, with highlighted significant accuracy reduction.

Value-Mapping

		Original	EquationDrug	ZeusVM-decypted	Cerber	Ardamax	NSIS	Kelihos	Mamba	WannaCry	Vikinghorde	Artemis
Medium DNNs	Resnet50	75.2%	74.8%	74.7%	75.1%	75.3%	74.6%	74.8%	74.6%	75.7%	75.5%	75.8%
	Inceptionv3	78%	78.3%	78.4%	78.2%	78%	77.2%	78.3%	78.4%	78.1%	77.6%	77.3%
	Densnet201	77%	77.1%	76.9%	77.3%	76.5%	77.2%	76.5%	77.3%	75.9%	77.3%	76.7%
	Resnet18	70.7%	71.1%	70.2%	70.5%	71.6%	72.1%	70.9%	71%	70.4%	70.3%	70.9%
Small DNNs	Googlenet	69.8%	70.1%	68.3%	70.2%	68.1%	70.3%	69.8%	68.4%	68.1%	70.3%	70.8%
	Comp.VGG-16	70.5%	71.3%	71%	68.3%	69.1%	71.2%	69.1%	69.2%	**48.7%**	-	-
	Comp.Alexnet	57%	56.9%	56.7%	57.2%	54.1%	-	-	-	-	-	-
	Squeezenet	57.5%	57.3%	56.9%	55.7%	56.8%	**39.7%**	**43.2%**	**21.8%**	-	-	-
	Mobilenet	70.9%	69.2%	71%	70.5%	70.3%	**54.7%**	**48.6%**	**49.3%**	-	-	-

Sign-Mapping

		Original	EquationDrug	ZeusVM-decypted	Cerber	Ardamax	NSIS	Kelihos	Mamba	WannaCry	Vikinghorde	Artemis
Large DNNs	Vgg19 (17.16MB)	71.1%	71.3%	71.2%	70.7%	71.2%	70.9%	71.1%	71.2%	71.1%	70.8%	71.3%
	Vgg16 (16.45MB)	70.5%	71%	70.6%	69.9%	70.2%	71.1%	70.8%	70.2%	69.8%	69.7%	71%
	Alexnet (7.27MB)	57%	57.2%	57.1%	56.7%	57.3%	57.2%	56.8%	56.9%	57.3%	-	-
	Resnet101 (5.32MB)	77.1%	77.2%	77.4%	76.7%	76.3%	77%	77.3%	77.5%	-	-	-
Medium DNNs	Resnet50 (3.05MB)	75.2%	74.8%	75.3%	75.1%	75.5%	75.4%	75.3%	74.9%	-	-	-
	Inceptionv3 (2.84MB)	78%	77.4%	78.2%	78.1%	77.8%	78%	78.1%	-	-	-	-
	Densnet201 (2.38MB)	77%	77.3%	77.2%	76.4%	76.7%	77.1%	76.4%	-	-	-	-
	Resnet18 (1.39MB)	70.7%	71.1%	70.8%	71%	71.2%	69.5%	-	-	-	-	-
Small DNNs	Googlenet (0.83MB)	69.8%	68.3%	70.2%	70.1%	-	-	-	-	-	-	-
	Comp.VGG-16 (1.34MB)	70.5%	71.3%	71%	70.4%	69.6%	-	-	-	-	-	-
	Comp.Alexnet (0.83MB)	57%	55.4%	56.7%	56.3%	-	-	-	-		-	-
	Squeezenet (0.15MB)	57.5%	-	-	-	-	-	-	-	-	-	-
	Mobilenet (0.5MB)	70.9%	68.3%	-	-	-	-	-	-	-	-	-

cates current technique is incapable of embedding malware into DNN model due to the size constraint. The bold numbers represent significant accuracy degradation compared with the original accuracy. Note that the accuracy reduction after payload injection can be very marginal for large DNN models such as uncompressed Vgg19, Vgg16, Alexnet and Resnet10 due to the sufficient redundant space, therefore we do not show such results. Meanwhile, we can observe that sometimes the modified DNN models can achieve even better testing accuracy than that of original model, this is because the evaluation is subject to $< \pm 1\%$ errors due to the randomness in DNN testing, which is in an acceptable margin.

As Table 4.10 shows, though naive LSB substitution can maintain the good testing accuracy on medium DNNs, this fact does not hold on small DNNs. For example, it causes significant accuracy degradation (i.e., sharply drop to $\sim 0.1\%$) on highly compressed DNN models In contrast, resilience training can relatively better support payload injection on small DNNs. For small malwares like EquationDrug, ZeusVM and Cerber, it can keep the testing accuracy as the same level of original one even on the smallest Mobilenet (4.2MB) and Squeezenet (4.6MB). However, the accuracy on Mobilenet is significantly dropped from 66.7% to 0.7% as the malware size increases from 0.59MB (Cerber) to 3.35MB (WannaCry). This is because the embedding rate (defined as malware/model size) exceeds the error-resilient capability of the DNN model. We observe that the upper bound of embedding rate for resilience training without accuracy degradation is $\sim 15\%$.

However, such an issue has been alleviated on the “searching and mapping” based technique. As Table 4.11 shows, value-mapping achieves higher testing accuracy than that of resilience training in most cases. For example, for the large sample Mamba (2.3MB) within Mobilenet (4.2MB), the testing accuracy of resilience training is only 6.1% while that of value-mapping can be still as high as $\sim 50\%$. However, the "searching and mapping" based technique sometimes suffers from its own limitation. For those highly compressed DNN models like "Comp.Alexnet", since model parameters are extremely quantized (i.e., data precision reduction), value-mapping can be less effective when mapping the binary payload to appropriate weight parameters. The similar trend can be also found in sign-mapping. The embedding rate is further reduced due to the limited number of sign bits in DNN model (i.e., one per each parameter). However, overall we observe that sign-mapping can always maintain the original testing accuracy for all applicable cases, indicating the best option to secure the evasiveness of MAL-DNN when possible.

Table 4.12: Detection rate reported on Metadefender [88].

Baselines	**Selected Malware Samples**					
	Asprox	**Bladabindi**	**Destover**	**Kovter**	**Stuxnet**	**ZeusVM**
Vanilla-malware	72.97%	75.68%	83.78%	62.16%	89.19%	91.89%
Stegomalware	8.11%	10.81%	13.51%	5.41%	0.00%	8.11%
***LSB substitution**	0.00%	2.70%	2.70%	0.00%	0.00%	0.00%
***Sign-mapping**	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

Anti-malware Detection

Table 4.12 compares the anti-malware detection rate among four designs. For a fair comparison, we evaluate the LSB substitution and sign-mapping on highly compressed Squeezenet (1.24MP), to ensure that the embedding rate or bit per pixel (bpp) of stegomalware ($1600 \times 800 = 1.28$MP) and the bit per parameter of created MAL-DNN sample are maintained at the same level (bpp$\approx$1).

As Table 4.12 shows, all vanilla-malware samples can be successfully detected by Metadefender with high detection rates (i.e., 62.16%$\sim$91.89%). Compared with vanilla-malware, the anti-malware detection rate of stegomalware can be reduced by at least six times (i.e., from 83.78% to 13.51% on Destover). This means that a few heuristic anti-malware engines can still detect the stegomalware, though with a

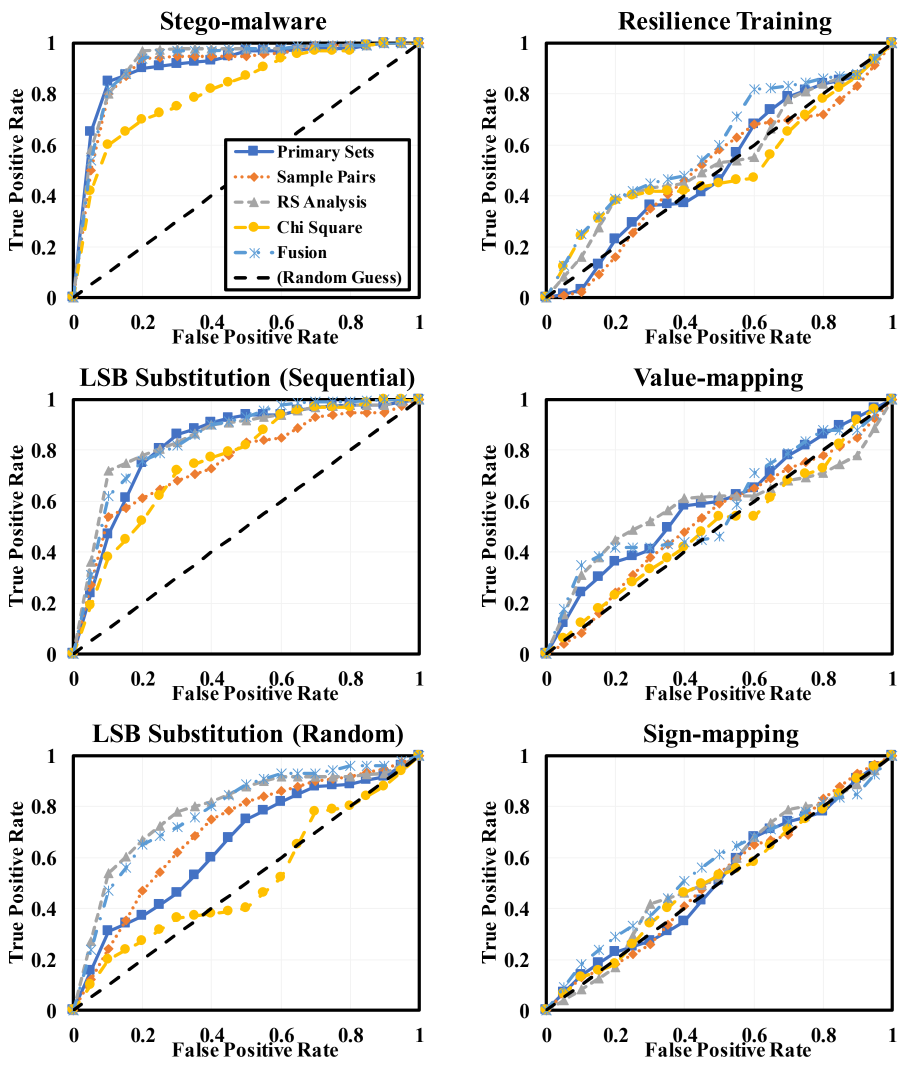


Figure 4.3: Receiver Operating Characteristic (ROC) of Steganalysis detection.

much lower rate. On the other hand, LSB substitution based MAL-DNN samples are more evasive. Only 2 (i.e., 2.7% on Bladabindi and Destover) out of 185 test cases can be detected. Moreover, the more sophisticated sign-mapping achieves the least detection rate for all 37 anti-malware engines.

Steganalysis Detection

Fig. 4.3 further compares the detection rates of different designs when adopting various steganalysis methods. In particular, we test two variants of LSB substitution–sequential and random, which embed payload binary into the LSB of sequentially and randomly selected DNN parameters, respectively. The area under curve (AUC) of receiver operating characteristic (ROC) represents the detection rate. **A smaller AUC indicates better evasiveness.**

As Fig. 4.3 shows, most steganalysis methods can effectively detect image-based stegomalware, e.g. a large AUC can be observed from the idea ROC towards (0,1). Due to the similarity to stegomalware, the simple LSB substitution based MAL-DNN, can be also detected by steganalysis. However, all these methods show degraded effectiveness (i.e., reduced AUC) comparing with image-based stegomalware, as the structure of DNN model is much more complex than that of image. As expected, Chi Square and Primary Sets suffer from significant detection performance degradation (i.e., close to random guess) for the advanced random LSB substitution in MAL-DNN. However, all the steganalysis methods, including more powerful RS Analysis and Fusion, are incapable of detecting three advanced MAL-DNN designs based on resilience training, value-mapping and sign-mapping, with almost close to random guess performance as shown in the right column of Fig. 4.3.

Discussion: Why do we separate the payload embedding (in model parameters) from its execution in MAL-DNN? How does MAL-DNN compare with traditional malware and stegomalware?

We first directly embed the payload into the execution as a traditional malware, e.g., hard-code payload in Softmax function. Our results in anti-malware detection clearly show that this naive design can be easily detected by existing defense engines. On the other hand, embedding payload into DNN parameters greatly challenges existing defenses as confirmed by our results. Since the state-of-the-art DNN models offer adequate space for payload embedding, the payload, which is usually much smaller than DNN models, can be randomly distributed among millions to billions of completely implicit DNN parameters with marginal accuracy loss. This is also applicable to compressed DNN models because of our proposed embedding techniques. Compared with existing steganography using a simple and small image, text, etc. (stegomalware)., embedding payload into DNN parameters with our schemes delivers the next level of obfuscation. Detecting such a small-sized hidden payload in a large and complex DNN is akin to finding a needle in a haystack. As such, existing anti-malware detection and steganalysis methods, cannot work very well for MAL-DNN, showing that MAL-DNN can be much more evasive than both traditional malware and stegomalware.

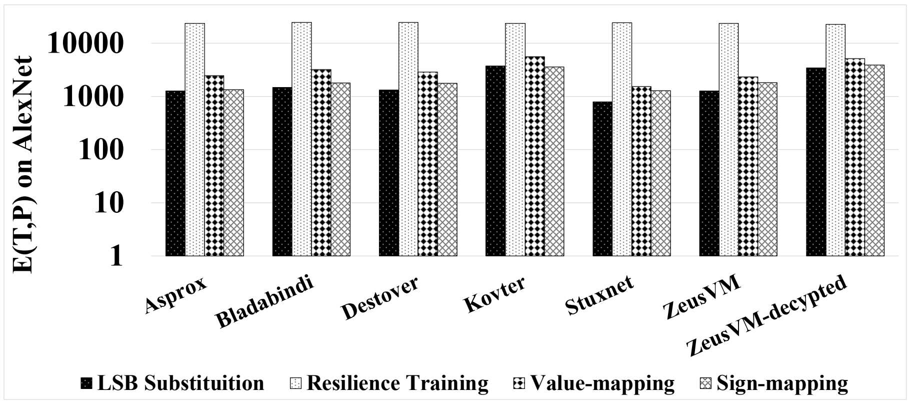
4.5.3 Efficiency

Metrics and Methods

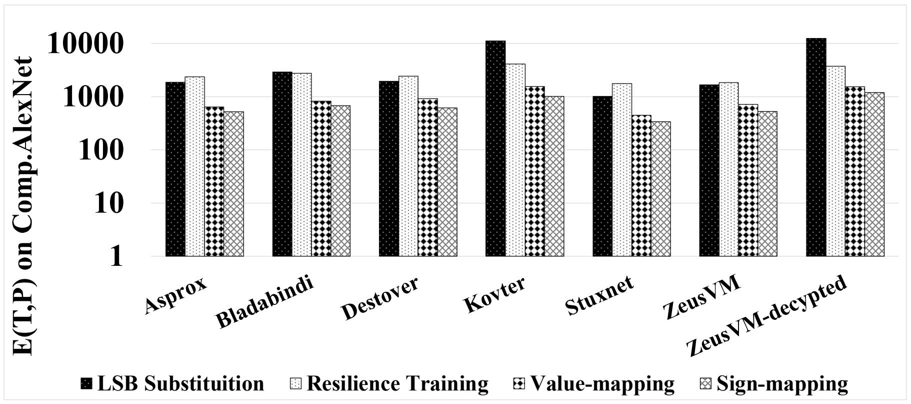
The efficiency reflects how much efforts are needed to create MAL-DNN. To characterize the efficiency, testing accuracy should be also taken into account besides the time cost, especially for compressed DNN models, since it indicates whether the crafted MAL-DNN can be accepted by end users. Therefore, the efficiency is defined as:

$$E(T, P) = \frac{T_{injection}}{P_{accuracy}} \tag{4.2}$$

Where T and P denote payload embedding time (ms) and testing accuracy (%) after payload injection, respectively. **A smaller value of $E(T, P)$ indicates better efficiency.** Without loss of generality, we use the popular AlexNet [66] and its com-



(a) Efficiency of payload injection on AlexNet.



(b) Efficiency of payload injection on Compressed AlexNet.

Figure 4.4: Efficiency of payload injection in MAL-DNN.

pressed version [50] to compare the efficiency among different injection techniques. In particular, resilience training is conducted until the DNN model is converged (i.e., the loss is $\leq 5\%$ between two consecutive training batches). For a fair comparison, we test them all on the same CPU. Note resilience training can be further accelerated by GPU.

Evaluation Results

As shown in Fig. 4.4(a), since the uncompressed Alexnet maintains sufficient model redundancy, all proposed techniques barely suffer from accuracy degradation. Therefore, this result can be also directly interpreted as the time cost of proposed techniques. As expected, LSB substitution is the most efficient technique due to its simple bit-wise operation, e.g. only takes a few seconds to generate any MAL-DNN sample. Sign-mapping can provide the similar efficiency, since it only assesses the sign bit of each parameter. Compared with LSB and sign-mapping, value mapping can be less efficient due to the value comparison between payload segments and parameters, though it still demonstrates the same order of magnitude of $E(T, P)$. Among all these techniques, resilience training gives the lowest efficiency, e.g. more than one order of magnitude higher than others, due to the re-training overhead. However, this is still much less than training from scratch (10^5ms v.s. hours/days).

Unlike that of uncompressed Alexnet, LSB substitution almost achieves the worst efficiency among all techniques in compressed version due to more prominent accuracy reduction in compressed Alexnet, as Fig. 4.4(b) shows. On the other hand, the efficiency of resilience training in compressed Alexnet outperforms that of uncompressed one due to much reduced model size but comparable accuracy. Moreover, value-mapping and sign-mapping become the two most efficient techniques on compressed version due to the similar reason.

4.5.4 Robustness

The robustness indicates the integrity of the injected payload that is essential to execute the payload. However, the lightweight modifications such as parameter fine-tuning (a common approach in transfer learning) can compromise the integrity of payload. Therefore, we further evaluate the robustness of MAL-DNN.

Table 4.13: Payload bit-flipping rate after fine-tuning.

Kovter on Alexnet				
	LSB Sub.	**Resilience Tr.**	**Value-map.**	**Sign-map.**
Output-layer	8.26%	0.0%	0.0%	0.0%
Fully-connects	43.8%	36.4%	8.6%	0.0%
Full-net	50.2%	48.1%	35.5%	0.0%
Kovter on Compressed Alexnet				
	LSB Sub.	**Resilience Tr.**	**Value-map.**	**Sign-map.**
Output-layer	6.72%	0.0%	0.0%	0.0%
Fully-connects	26.8%	18.2%	2.3%	0.0%
Full-net	37.1%	23.1%	16.7%	0.0%

Metrics and Methods

We target the integrity issue and apply fine-tuning on MAL-DNN samples created with payload Kovter. The bit-flipping rate of MAL-DNN sample after fine-tuning is selected as metric to evaluate the robustness. **A less bit-flipping rate indicates the better robustness.** In practice, fine-tuning is usually only applied on the DNN output layer to optimize the decision making with the least effort. For evaluation purpose, we analyze the following three fine-tuning scenarios: **1) the default output-layer only; 2) fully-connects only** (stronger modification–parameters of fully-connected layers); **3) full-net** (strongest modification–parameters across all DNN layers).

Evaluation Results

As Table 4.13 shows, sign-mapping is the best option for payload integrity protection on both Alexnet and compressed Alexnet. It can guarantee the payload integrity without introducing any bit-flipping for all fine-tune cases. This is because the sign bit of the weights, especially for those important parameters with large values, is defined at the training stage and rarely flips during the fine tune

Table 4.14: Comparison of explored payload injection techniques.

	LSB Substitution	**Resilience Training**	**Value-mapping**	**Sign-mapping**
Pros	Simple bit-wise operation, less overheads;	Support compressed models; Improved evasiveness;	Support compressed models; No re-training;	Support compressed models; No re-training; Well protect integrity;
Cons	Incompatible with compressed models;	Less time efficiency on re-training;	Minor accuracy degradation;	Low capacity;

despite the slight magnitude change. Resilience training and value-mapping also demonstrate remarkable robustness against default fine-tuning. In contrast, LSB Substitution is the most sensitive method to all fine-tunings. We also observe that all payload injection techniques applied to compressed Alexnet can be better than that of uncompressed Alexnet. This is because quantized parameters with reduced data precision in compressed model may better prevent the value changes caused by fine-tuning due to the parameter sharing.

Discussion: How to select an appropriate payload injection technique?

The following three aspects should be jointly considered: evasiveness, efficiency and robustness. Based on our evaluation results, Table 4.14 further summarizes the pros and cons of explored payload injection techniques. Compared with LSB substitution, resilience training can be directly applied to quantized parameters without model redundancy, thus making MAL-DNN scalable on compressed DNN models. Besides, the randomly generated permutation can further improve the evasiveness of payload. However, resilience training is less efficient than LSB substitution due to the re-training cost. Instead of freezing a bundle of pre-selected parameters in resilience training, both value-mapping and sign-mapping can freely select parameters in DNN model for payload binary mapping without re-training, and hence significantly improve the efficiency of payload injection. Moreover, the sign-mapping technique can well protect the integrity of injected payload as fine-tuning barely changes the sign of model parameters.

4.5.5 Trigger Reliability

Reliability measures the performance of our trigger design against input variations from physical-world.

Metrics and Methods

Our proposed rank trigger can be addressed as a specific rule based binary decision (i.e., to match the "logits rank"). The trigger rate (or binary decision accuracy) can be used to measure the performance with trigger event (i.e., specific input images) selected as positive samples and normal inputs (i.e., benign images) selected as negative samples. Accordingly, we use F_1-score as metric in our evaluation:

$$F_1 = \left(\frac{2}{\text{Precision}^{-1} + \text{Recall}^{-1}}\right) \text{with} \begin{cases} \text{Precision} = {}^{\text{TP}}/_{\text{TP+FP}} \\ \text{Recall} = {}^{\text{TP}}/_{\text{TP+FN}} \end{cases} \tag{4.3}$$

where TP (True positive) is a successful triggering with specific input, FP (False positive) is an unsuccessful triggering with specific input, and FN (False negative) is incorrectly triggered with normal input. This metric can fairly reflect the trigger performance under the imbalanced number of positive and negative samples (i.e., 1:10 in our method), and shows to what degree the attacker can "control" the MAL-DNN from a statistical perspective.

To measure the trigger reliability, we adopt the "physical-world configuration" from our demonstrated prototype. We use 1000 benign images (selected from 10

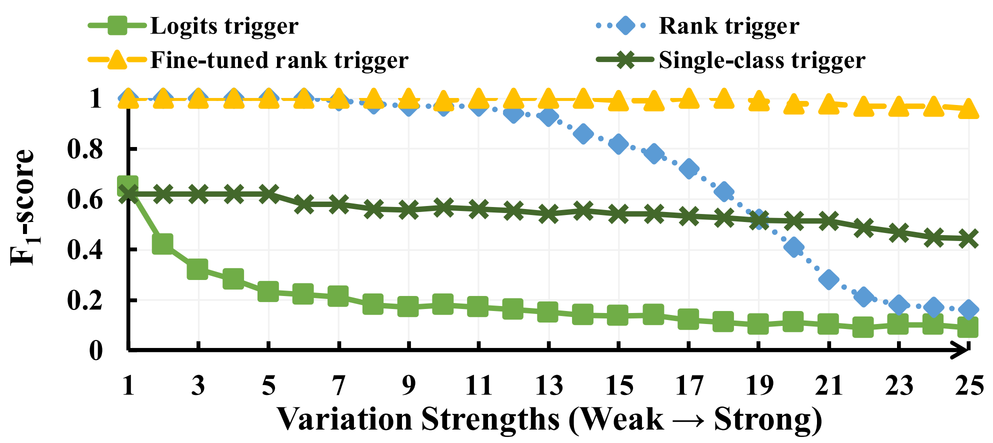


Figure 4.5: Reliability of different trigger designs.

sub-class of Imagenet dataset) as negative samples, and create 100 specific images as positive samples to trigger the MAL-DNN on Alexnet. These 100 specific images are augmented from the original "chessboard" (see Fig. 4.2) by applying different types of input variations across 25 different levels. We measure and compare the F_1-score of different trigger designs on each variation level. **A higher F_1-score indicates the better reliability.**

Evaluation Results

Fig. 4.5 compares the reliability of four different trigger designs, including proposed logits trigger, rank trigger and fine-tuned rank trigger, as well as the classic single-class trigger used in most works (e.g. backdoor). We choose 4 logits (out of total 10-class) to create our proposed trigger designs. The variation strengths are quantified as $1 \rightarrow 25$.

As Fig. 4.5 shows, our proposed fine-tuned rank trigger achieves the best reliability among all designs, followed by rank trigger, single-class trigger and logits trigger. With increased variation strengths, the fine-tuned rank trigger can always maintain the highest F_1-score (≈ 1). The F_1-score of rank trigger drops from ~ 1 to ~ 0.2 as

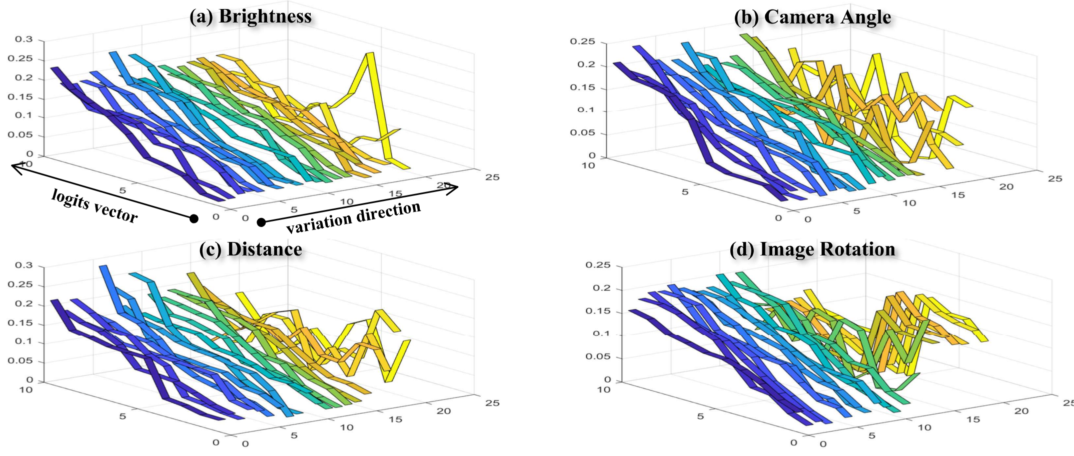


Figure 4.6: Observation of Logits Rank trigger with different variation strengths.

the variation strengths increases from 10→25, which indicates its lower reliability against stronger input variations. The logits trigger shows the least reliability in all cases, as proved by its sharply reduced F_1-score, e.g. from 0.64 to ~ 0.1 (random guess). Compared with three proposed trigger designs, the single-class trigger is "stable" but not "reliable". It can maintain the F_1-score at a certain level as the variation strength grows, however, its best F_1-score is very low. This is because the single-class trigger, which is widely adopted by existing backdoor attack, suffers from significant False Negative errors in our design (i.e., negative samples or normal inputs in the same class mistakenly triggers the malware). This result also confirms that existing triggering mechanism, as a special case of our logits rank trigger design with only a top one logits, cannot work well in MAL-DNN.

Discussion: Why is rank trigger more reliable under input variations?

Our results show that rank trigger design (including the fine-tuned rank trigger) is more reliable under input variations than that of logits trigger, and confirm that logits rank can rectify the imperfection of mismatched key-lock pair. To explore the reason, we investigated the status of logits rank under different types of input variations, i.e., brightness, camera angle and rotation, with different strengths. As Fig. 4.6 shows, for each type of variation, the rank trigger can always maintain the logits rank (i.e., presented as the monotonous trend of the ribbons), under small variation strengths. However, with the increased variation strength, rank trigger is unable to maintain the logits rank. On the other hand, the fine-tuned rank trigger can effectively handle the strong input variations with much enhanced reliability. These intuitive results show that logits rank is naturally less sensitive to the input variations. Even the value of logits is significantly fluctuated, the rank of logits can be still stable.

4.6 Mitigation Exploration

In our evaluation, we tested the existing Anti-Malware detection and steganalysis approaches to mitigate the MAL-DNN. However, both solutions are incapable of mitigating such an emerging attack on DNN based ML accelerators. In this section, we present the guideline to mitigate the MAL-DNN attack. The developers may consider the following directions to secure their intelligent applications.

Code Review. Code review is the most straightforward method to examine the existence of MAL-DNN trigger. It should be very effective against the generated code with interpreted language (i.e. the Python). Otherwise, reverse engineering can be applied to analyze the obfuscated or compiled bytecode.

Trusted Environment. As aforementioned, most serialization library and deserialization process can be used to perform extraction and execute the injected payload. Hence, a possible solution is to define the trusted environment, i.e. Sandbox or RestrictedPython [99], to secure the program execution and data flow by restricting the usage of unsafe APIs, thus to defeat the payload injection.

Behavioral Detection. The program behaviors can be collected during the DNN execution to identify the malicious functions [59]. For example, the bit-wise operations used by "LSB substitution" are the suspicious behaviors and can be used to identify the payload injection [46]. However, such an approach should be carefully conducted as the similar behavior may appear as legitimate neural processing in customized DNN architectures (i.e. the Bitwise Neural Network [64]).

4.7 Summary

Deep Neural Networks (DNNs) are now presenting human-level performance on many real-world applications, and DNN-based intelligent services are becoming more and more popular across all aspects of our lives. Unfortunately, the ever-increasing DNN service implies a dangerous feature which has not yet been well studied– allowing the marriage of existing malware and DNN model for any pre-defined malicious purpose. In this work, we comprehensively investigate how to turn DNN into a new breed evasive self-contained stegomalware, namely MAL-DNN, using model parameter as a novel payload injection channel, with no service quality degradation (i.e. accuracy) and the triggering event connected to the physical world by specified DNN inputs. A series of payload injection techniques contingent upon unique neural network natures like complex structure, high error resilience and huge parameter size, are developed for both uncompressed models (with redundancy) and deeply compressed models tailored for resource-limited devices (no redundancy), including LSB substitution, resilience training, value mapping, and sign-mapping. We also proposed a set of triggering techniques like logits trigger, rank trigger and fine-tuned rank trigger to trigger MAL-DNN by specific physical events under realistic environment variations. We implement the MAL-DNN prototype on Nvidia Jetson TX2 testbed. Extensive experimental results and discussions on the evasiveness, efficiency and integrity of proposed payload injection techniques, and the reliability and sensitivity of the triggering techniques, well demonstrate the feasibility and practicality of MAL-DNN. The unique characteristics and possible mitigation directions of MAL-DNN have been discussed as well.

CHAPTER 5

CONCLUSION

The digital era is now evolving into the intelligence era, driven overwhelmingly by the data explosion and machine learning advancement. Embedded devices, sensors, and the Internet of Everything (IoE) are nowadays producing ever-increasing amounts of data. Besides, modern Machine Learning (ML) techniques open the door for intelligent data interpretation on these devices, achieving game-changing outcomes on machine vision, auto-driving, social engineering, etc. However, embedded machine learning system design is subject to various challenges, e.g., performance bottleneck due to large amounts of data storage and processing, accuracy degradation caused by hardware defects in emerging processing-in-memory (PIM) accelerators, as well as security concerns raised by adversarial machine learning and open-sourced computing framework. This dissertation presents a comprehensive architecture and algorithm co-design approach for embedded machine learning system design. We systematically revisited the design challenges and and our solutions to address three fundamental design pillars, namely efficiency, reliability, and security.

In particular, our first case study focuses on designing the energy and power-efficient neuromorphic computing systems. We find that the ultra sparse coding can significantly improve system efficiency, and a possible approach is to leverage the spatial-temporal trade-off of time-based neural coding in spiking neuromorphic architecture. The spatial and temporal locality shall be revisited carefully in single-spike time-based neural coding. It can also achieve remarkable design flexibility. The asymmetric coding for the first time shows the potential of non-uniform weight updating, improving the efficiency and accuracy in an interesting manner.

Our second case study illustrates how to create the scalable fault-tolerance of the emerging DNN accelerator enhance the fault tolerance of the emerging processing-in-memory (PIM) based DNN accelerator in a scalable manner. We find that the weight disturbance issue in the DNN accelerator design is fundamentally different from the traditional memory bit (0/1) error. While DNNs do exhibit inherent error resilience, such capability is too limited to handle the weight errors incurred by the non-ideal device factors. We marriage the error correction output coding with transfer learning to significantly boost the error tolerance capability of these emerging PIM accelerators in a scalable and low-cost way, offering a fundamentally different thinking to design reliable and sustainable machine learning accelerators built upon emerging devices, without knowing where, when and how weight errors occur. Our solution is smart, scalable, low cost, and can significantly improve the accuracy of emerging accelerators regardless of where/what/when weight errors occur.

Finally, our third case study reveals a new type of threat which allows to conceal the malicious intent into the complex DNN model, even a highly compressed version dedicated to hardware accelerator, without impairing DNN service quality, e.g., classification accuracy. We find that machine learning security shall be revisited in a system-level perspective, with interdisciplinary knowledge from the software and hardware together. Our early exploration of the Mal-DNN: malicious DNN-powered stegomalware represents a new angle to further investigate the emerging cross-layer vulnerabilities among DNN architecture, algorithm, hardware and software, which is of critical importance for the deployment of the embedded machine learning system in the near future.

BIBLIOGRAPHY

[1] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *arXiv preprint arXiv:1801.00553*, 2018.

[2] Hiroyuki Akinaga and Hisashi Shima. Resistive random access memory (reram) based on metal oxides. *Proceedings of the IEEE*, 98(12):2237–2251, 2010.

[3] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1537–1557, 2015.

[4] Fabien Alibart, Ligang Gao, Brian D Hoskins, and Dmitri B Strukov. High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm. *Nanotechnology*, 23(7):075201, 2012.

[5] Erin L Allwein, Robert E Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of machine learning research*, 1(Dec):113–141, 2000.

[6] Amazon. Amazon machine learning. `https://aws.amazon.com/machine-learning/`, 2018.

[7] Renzo Andri, Lukas Cavigelli, Davide Rossi, and Luca Benini. Yodann: An architecture for ultra-low power binary-weight cnn acceleration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[8] Brendan Barry, Cormac Brick, Fergal Connor, David Donohoe, David Moloney, Richard Richmond, Martin O'Riordan, and Vasile Toma. Always-on vision processing unit for mobile applications. *IEEE Micro*, 35(2):56–66, 2015.

[9] Adam Berger. Error-correcting output coding for text classification. In *IJCAI-99: Workshop on machine learning for information filtering*, 1999.

[10] Benedikt Boehm. Stegexpose-a tool for detecting lsb steganography. *arXiv preprint arXiv:1410.6656*, 2014.

[11] Sander M Bohte, Joost N Kok, and Han La Poutre. Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, 48(1):17–37, 2002.

[12] Mahdi Nazm Bojnordi and Engin Ipek. Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning. In *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*, pages 1–13. IEEE, 2016.

[13] Alexander Borst and Frédéric E Theunissen. Information theory and neural coding. *Nature neuroscience*, 2(11):947–957, 1999.

[14] Anthony N Burkitt. A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. *Biological cybernetics*, 95(1):1–19, 2006.

[15] Daniel A Butts, Chong Weng, Jianzhong Jin, Chun-I Yeh, Nicholas A Lesica, Jose-Manuel Alonso, and Garrett B Stanley. Temporal precision in the neural code and the timescales of natural vision. *Nature*, 449(7158):92–95, 2007.

[16] BVLC/caffe. Dnn model zoo. `https://github.com/BVLC/caffe/wiki/Model-Zoo/`, 2018.

[17] Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66, 2015.

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

[19] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi. A dynamically configurable coprocessor for convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, volume 38, pages 247–257. ACM, 2010.

[20] Ting Chang, Sung-Hyun Jo, and Wei Lu. Short-term memory to long-term memory transition in a nanoscale memristor. *ACS nano*, 5(9):7669–7676, 2011.

[21] Abbas Cheddad, Joan Condell, Kevin Curran, and Paul Mc Kevitt. Digital image steganography: Survey and analysis of current methods. *Signal processing*, 90(3):727–752, 2010.

[22] Lerong Chen, Jiawen Li, Yiran Chen, Qiuping Deng, Jiyuan Shen, Xiaoyao Liang, and Li Jiang. Accelerator-friendly neural-network training: learning variations and defects in rram crossbar. In *Proceedings of the Conference on Design, Automation & Test in Europe*, pages 19–24. European Design and Automation Association, 2017.

[23] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[24] Yunji Chen, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. Diannao family: energy-efficient hardware accelerators for machine learning. *Communications of the ACM*, 59(11):105–112, 2016.

[25] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. Prime: a novel processing-in-memory architecture for neural network computation in reram-based main memory. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 27–39. IEEE Press, 2016.

[26] Myonglae Chu, Byoungho Kim, Sangsu Park, Hyunsang Hwang, Moongu Jeon, Byoung Hun Lee, and Byung-Geun Lee. Neuromorphic hardware system for visual pattern recognition with memristor array and cmos neuron. *IEEE Transactions on Industrial Electronics*, 62(4):2410–2419, 2015.

[27] Federico Corradi and Giacomo Indiveri. A neuromorphic event-based neural recording system for smart brain-machine-interfaces. *IEEE transactions on biomedical circuits and systems*, 9(5):699–709, 2015.

[28] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[29] Peter U Diehl and Matthew Cook. Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in computational neuroscience*, 9, 2015.

[30] Thomas G Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of artificial intelligence research*, 2:263–286, 1995.

[31] Sorina Dumitrescu, Xiaolin Wu, and Nasir Memon. On steganalysis of random lsb embedding in continuous-tone images. In *Proceedings. International Conference on Image Processing*, volume 3, pages 641–644. IEEE, 2002.

[32] Sorina Dumitrescu, Xiaolin Wu, and Zhe Wang. Detection of lsb steganography via sample pair analysis. In *International Workshop on Information Hiding*, pages 355–372. Springer, 2002.

[33] Steven K Esser, Paul A Merolla, John V Arthur, Andrew S Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J Berg, Jeffrey L McKinstry, Timothy Melano, Davis R Barch, et al. Convolutional networks for fast, energy-efficient neuromorphic computing. *Proceedings of the National Academy of Sciences*, page 201604850, 2016.

[34] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. Robust physical-world attacks on deep learning models. *arXiv preprint arXiv:1707.08945*, 1, 2017.

[35] Amin Farmahini-Farahani, Jung Ho Ahn, Katherine Morrow, and Nam Sung Kim. Nda: Near-dram acceleration architecture leveraging commodity dram devices and standard memory modules. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 283–295. IEEE, 2015.

[36] Ben Feinberg, Shibo Wang, and Engin Ipek. Making memristive neural network accelerators reliable.

[37] Ben Feinberg, Shibo Wang, and Engin Ipek. Making memristive neural network accelerators reliable. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 52–65. IEEE, 2018.

[38] RA Fisher and Michael Marshall. Iris data set. *UC Irvine Machine Learning Repository*, 1936.

[39] Annie Foong and Frank Hady. Storage as fast as rest of the system. In *Memory Workshop (IMW), 2016 IEEE 8th International*, pages 1–4. IEEE, 2016.

[40] Jessica Fridrich, Miroslav Goljan, and Rui Du. Reliable detection of lsb steganography in color and grayscale images. In *Proceedings of the 2001 workshop on Multimedia and security: new challenges*, pages 27–30. ACM, 2001.

[41] W Gerstner. A framework for spiking neuron models: The spike response model. *Handbook of Biological Physics*, 4:469–516, 2001.

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

[43] Dan FM Goodman and Romain Brette. The brian simulator. *Frontiers in neuroscience*, 3:26, 2009.

[44] Google. Pixel visual core: image processing and machine learning on pixel 2. `https://www.blog.google/products/pixel/pixel-visual-core-image-processing-and-machine-learning-pixel-2/`, 2017.

[45] Google. Google cloud machine learning. `https://cloud.google.com/products/machine-learning/`, 2018.

[46] Jonathon Patrick Green, Anjali Doulatram Chandnani, and Simon David Christensen. Detecting script-based malware using emulation and heuristics, January 2 2018. US Patent 9,858,414.

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

[48] Robert Gütig and Haim Sompolinsky. The tempotron: a neuron that learns spike timing–based decisions. *Nature neuroscience*, 9(3):420–428, 2006.

[49] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 123–136, 2016.

[50] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[51] 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.

[52] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[53] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[54] Miao Hu, Hai Li, Yiran Chen, Xiaobin Wang, and Robinson E Pino. Geometry variations analysis of tio 2 thin-film and spintronic memristors. In *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, pages 25–30. IEEE, 2011.

[55] Miao Hu, John Paul Strachan, Zhiyong Li, Emmanuelle M Grafals, Noraica Davila, Catherine Graves, Sity Lam, Ning Ge, Jianhua Joshua Yang, and R Stanley Williams. Dot-product engine for neuromorphic computing: programming 1t1m crossbar to accelerate matrix-vector multiplication. In *Proceedings of the 53rd annual design automation conference*, page 19. ACM, 2016.

[56] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.

[57] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[58] Facebook Inc. Open neural network exchange (onnx). `https://onnx.ai/`, 2017.

[59] Grégoire Jacob, Hervé Debar, and Eric Filiol. Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in computer Virology*, 4(3):251–266, 2008.

[60] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 19–35. IEEE, 2018.

[61] Yan-huang Jiang, Qiang-li Zhao, and Xue-jun Yang. A general coding method for error-correcting output codes. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 648–652. Springer, 2004.

[62] Richard Kempter, Wulfram Gerstner, J Leo Van Hemmen, and Hermann Wagner. Temporal coding in the sub-millisecond range: Model of barn owl auditory pathway. In *Advances in neural information processing systems*, pages 124–130, 1996.

[63] Mehdi Kharrazi, Husrev T Sencar, and Nasir Memon. Improving steganalysis by fusion techniques: A case study with image steganography. In *Transactions on Data Hiding and Multimedia Security I*, pages 123–137. Springer, 2006.

[64] Minje Kim and Paris Smaragdis. Bitwise neural networks for efficient single-channel source separation. *Urbana*, 51:61801, 2018.

[65] Dhilung Kirat, Jiyong Jang, and Marc Stoecklin. Deeplocker - concealing targeted attacks with ai locksmithing. In *Blackhat USA 2018*. Blackhat, 2018.

[66] 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.

[67] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[68] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[69] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[70] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

[71] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[72] Bin Li, Junhui He, Jiwu Huang, and Yun Qing Shi. A survey on image steganography and steganalysis. *Journal of Information Hiding and Multimedia Signal Processing*, 2(2):142–172, 2011.

[73] Fangzhou Liao, Ming Liang, Yinpeng Dong, Tianyu Pang, Jun Zhu, and Xiaolin Hu. Defense against adversarial attacks using high-level representation guided denoiser. *arXiv preprint arXiv:1712.02976*, 2017.

[74] Chenchen Liu, Miao Hu, John Paul Strachan, and Hai Li. Rescuing memristor-based neuromorphic design with high defects. In *Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE*, pages 1–6. IEEE, 2017.

[75] Chenchen Liu, Bonan Yan, Chaofei Yang, Linghao Song, Zheng Li, Beiye Liu, Yiran Chen, Hai Li, Qing Wu, and Hao Jiang. A spiking neuromorphic design with resistive crossbar. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.

[76] Chenchen Liu, Qing Yang, Bonan Yan, Jianlei Yang, Xiaocong Du, Weijie Zhu, Hao Jiang, Qing Wu, Mark Barnell, and Hai Li. A memristor crossbar based computing engine optimized for high speed and accuracy. In *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*, pages 110–115. IEEE, 2016.

[77] Tao Liu, Lei Jiang, Yier Jin, Gang Quan, and Wujie Wen. Pt-spike: a precise-time-dependent single spike neuromorphic architecture with efficient supervised learning. In *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*, pages 568–573. IEEE Press, 2018.

[78] Tao Liu, Zihao Liu, Fuhong Lin, Yier Jin, Gang Quan, and Wujie Wen. Mt-spike: a multilayer time-based spiking neuromorphic architecture with temporal error backpropagation. In *Proceedings of the 36th International Conference on Computer-Aided Design*, pages 450–457. IEEE Press, 2017.

[79] Tao Liu and Wujie Wen. A fast and ultra low power time-based spiking neuromorphic architecture for embedded applications. In *Quality Electronic Design (ISQED), 2017 18th International Symposium on*, pages 19–22. IEEE, 2017.

[80] Xiaoxiao Liu, Mengjie Mao, Beiye Liu, Boxun Li, Yu Wang, Hao Jiang, Mark Barnell, Qing Wu, Jianhua Yang, Hai Li, et al. Harmonica: A framework of heterogeneous computing systems with memristor-based neuromorphic com-

puting accelerators. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 63(5):617–628, 2016.

[81] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *Department of Computer Science Technical Reports, Purdue University*, page 1781. Purdue e-Pubs, 2017.

[82] Yuntao Liu, Yang Xie, and Ankur Srivastava. Neural trojans. In *Computer Design (ICCD), 2017 IEEE International Conference on*, pages 45–48. IEEE, 2017.

[83] Wolfgang Maass. On the computational power of winner-take-all. *Neural computation*, 12(11):2519–2535, 2000.

[84] MalwareWiki. Downloader. `http://http://malware.wikia.com/wiki/Downloader_Generic/`, 2018.

[85] MalwareWiki. Fork bomb. `http://malware.wikia.com/wiki/Fork_Bomb/`, 2018.

[86] D McMillen. Steganography: A safe haven for malware. 2017.

[87] Gilberto Medeiros-Ribeiro, Frederick Perner, Richard Carter, Hisham Abdalla, Matthew D Pickett, and R Stanley Williams. Lognormal switching times for titanium dioxide bipolar memristors: origin and resolution. *Nanotechnology*, 22(9):095702, 2011.

[88] Metadefender. Multiple security engines. `http://www.metadefender.com/`, 2019.

[89] Microsoft. Microsoft azure machine learning. `https://azure.microsoft.com/en-us/services/machine-learning/`, 2018.

[90] Hesham Mostafa. Supervised learning based on temporal coding in spiking neural networks. *arXiv preprint arXiv:1606.08165*, 2016.

[91] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[92] Yuval Nativ. thezoo aka malware db. `http://thezoo.morirt.com/`, 2015.

[93] Daniel Neil and Shih-Chii Liu. Minitaur, an event-driven fpga-based spiking network accelerator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(12):2621–2628, 2014.

[94] Dimin Niu, Yiran Chen, Cong Xu, and Yuan Xie. Impact of process variations on emerging memristor. In *Proceedings of the 47th Design Automation Conference*, pages 877–882. ACM, 2010.

[95] Dimin Niu, Yang Xiao, and Yuan Xie. Low power memristor-based reram design with error correcting code. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 79–84. IEEE, 2012.

[96] Dimin Niu, Cong Xu, Naveen Muralimanohar, Norman P Jouppi, and Yuan Xie. Design of cross-point metal-oxide reram emphasizing reliability and cost. In *Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference on*, pages 17–23. IEEE, 2013.

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

[98] Filip Ponulak. Resume-new supervised learning method for spiking neural networks. *Institute of Control and Information Engineering, Poznan University of Technology.(Available online at: http://d1. cie. put. poznan. pl/˜ fp/research. html)*, 2005.

[99] PyPi. Restrictedpython 4.0b2. `https://pypi.python.org/pypi/RestrictedPython/`, 2010.

[100] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[101] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

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

[103] Jae-sun Seo, Bernard Brezzo, Yong Liu, Benjamin D Parker, Steven K Esser, Robert K Montoye, Bipin Rajendran, José A Tierno, Leland Chang, Dharmendra S Modha, et al. A 45nm cmos neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. In *Custom Integrated Circuits Conference (CICC), 2011 IEEE*, pages 1–4. IEEE, 2011.

[104] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News*, 44(3):14–26, 2016.

[105] Amar Shrestha, Khadeer Ahmed, Yanzhi Wang, and Qinru Qiu. Stable spike-timing dependent plasticity rule for multilayer unsupervised and supervised learning. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 1999–2006. IEEE, 2017.

[106] David Silver and Demis Hassabis. Alphago: Mastering the ancient game of go with machine learning. *Research Blog*, 2016.

[107] Patrice Y Simard, Dave Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. In *null*, page 958. IEEE, 2003.

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

[109] Jesper Sjöström and Wulfram Gerstner. Spike-timing dependent plasticity. *Spike-timing dependent plasticity*, page 35, 2010.

[110] Congzheng Song, Thomas Ristenpart, and Vitaly Shmatikov. Machine learning models that remember too much. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 587–601. ACM, 2017.

[111] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. Pipelayer: A pipelined reram-based accelerator for deep learning. In *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*, pages 541–552. IEEE, 2017.

[112] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. *arXiv preprint arXiv:1710.10766*, 2017.

[113] Guillermo Suarez-Tangil, Juan E Tapiador, and Pedro Peris-Lopez. Stegomalware: Playing hide and seek with malicious components in smartphone apps. In *International Conference on Information Security and Cryptology*, pages 496–515. Springer, 2014.

[114] Christian Szegedy. An overview of deep learning. *AITP 2016*, 2016.

[115] 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.

[116] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[117] Tensorflow. Tensorflow models are programs. `https://github.com/tensorflow/tensorflow/blob/master/SECURITY.md/`, 2019.

[118] Simon Thorpe, Arnaud Delorme, and Rufin Van Rullen. Spike-based strategies for rapid processing. *Neural networks*, 14(6):715–725, 2001.

[119] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[120] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453*, 2017.

[121] Samir Vaidya. Openstego. `https://github.com/syvaidya/openstego/`, 2019.

[122] Yu Wang, Tianqi Tang, Lixue Xia, Boxun Li, Peng Gu, Huazhong Yang, Hai Li, and Yuan Xie. Energy efficient rram spiking neural network for real time classification. In *Proceedings of the 25th GLVLSI*, pages 189–194. ACM, 2015.

[123] Andreas Westfeld and Andreas Pfitzmann. Attacks on steganographic systems. In *International workshop on information hiding*, pages 61–76. Springer, 1999.

[124] Xiurui Xie, Hong Qu, Zhang Yi, and Jürgen Kurths. Efficient training of supervised spiking neural network via accurate synaptic-efficiency adjustment method. 2016.

[125] Bonan Yan, Jianhua Joshua Yang, Qing Wu, Yiran Chen, and Hai Helen Li. A closed-loop design to enhance weight stability of memristor based neural network chips. In *Proceedings of the 36th International Conference on Computer-Aided Design*, pages 541–548. IEEE Press, 2017.

[126] J Joshua Yang, Dmitri B Strukov, and Duncan R Stewart. Memristive devices for computing. *Nature nanotechnology*, 8(1):13, 2013.

[127] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[128] Qiang Yu, Huajin Tang, Kay Chen Tan, and Haizhou Li. Precise-spike-driven synaptic plasticity: Learning hetero-association of spatiotemporal spike patterns. *Plos one*, 8(11):e78318, 2013.

[129] Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multi-layer spiking neural networks. *arXiv preprint arXiv:1705.11146*, 2017.

[130] Bin Zhao and Eric P Xing. Sparse output coding for scalable visual recognition. *International Journal of Computer Vision*, 119(1):60–75, 2016.

[131] Chenyuan Zhao, Bryant T Wysocki, Clare D Thiem, Nathan R McDonald, Jialing Li, Lingjia Liu, and Yang Yi. Energy efficient spiking temporal encoder design for neuromorphic computing systems. *IEEE Transactions on Multi-Scale Computing Systems*, 2(4):265–276, 2016.

VITA

TAO LIU

2013–2013	M.S. in Computer Engineering Florida International University Miami, FL USA
2016–2020	Ph.D. in Electrical and Computer Engineering Florida International University Miami, FL USA

SELECTED PUBLICATIONS:

T. Liu, L. Jiang, Y. Jin, G. Quan and W. Wen, (Jan. 2018). *PT-Spike: A Precise-Time-Dependent Single Spike Neuromorphic Architecture with Efficient Supervised Learning*, Proc. IEEE 23rd Asia and South Pacific Design Automation Conference (ASP-DAC 2018).

T. Liu, Z. Liu, F. Lin, Y. Jin, G. Quan and W. Wen, (Nov. 2017). *MT-Spike: A Multilayer Time-based Spiking Neuromorphic Architecture with Temporal Error Backpropagation*, Proc. IEEE 36th International Conference on Computer-Aided Design (ICCAD 2017).

T. Liu and W. Wen, (Nov. 2019). *Making the Fault-Tolerance of Emerging Neural Network Accelerators Scalable*, Proc. IEEE 38th International Conference on Computer-Aided Design (ICCAD 2019).

T. Liu, W. Wen, L. Jiang, Y. Wang, C. Yang and G. Quan, (Jun. 2019). *A fault-tolerant neural network architecture*, Proc. ACM/IEEE 56th Design Automation Conference (DAC 2019).

T. Liu and W. Wen, (May. 2019). *Deep-Evasion: Turn Deep Neural Network into Evasive Self-Contained Cyber-Physical Malware*, 12th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec 2019).

T. Liu, N. Xu. Q. Liu, Y. Wang and W. Wen, (Jan. 2019). *A system-level perspective to understand the vulnerability of deep learning systems*, Proc. IEEE 24th Asia and South Pacific Design Automation Conference (ASP-DAC 2019).

T. Liu, Z. Liu, Q. Liu and W. Wen, (Jul. 2018). *Enhancing the Robustness of Deep Neural Networks from "Smart" Compression*, Proc. IEEE Computer Society Annual Symposium on VLSI (ISVLSI).

T. Liu, W. Wen and Y. Jin, (May. 2018). *SIN^2: Stealth Infection on Neural Network – A Low-cost Agile Neural Trojan Attack Methodology*, Proc. IEEE 11th International Symposium on Hardware Oriented Security and Trust (HOST 2018).

T. Liu, Y. Jin and W. Wen, (Mar. 2018). *Trojan Attacks and Defenses on Deep Neural Network based Intelligent Computing Systems*, Government Microcircuit Applications & Critical Technology Conference (GOMACTech).

T. Liu and W. Wen, (Mar. 2017). *A Fast and Ultra Low Power Time-Based Spiking Neuromorphic Architecture for Embedded Applications*, Proc. IEEE 18th International Symposium on Quality Electronic Design (ISQED 2017).

Q. Liu, T. Liu, Z. Liu, W. Wen and C. Yang, (Jun. 2020). *Monitoring the Health of Emerging Neural Network Accelerators with Cost-effective Concurrent Test*, Proc. ACM/IEEE 57th Design Automation Conference (DAC 2020).

N. Xu, Q. Liu, T. Liu, Z. Liu, X. Guo and W. Wen, (Jun. 2020). *Stealing Your Data from Compressed Machine Learning Models*, Proc. ACM/IEEE 57th Design Automation Conference (DAC 2020).

Q. Liu, T. Liu, Z. Liu, Y. Wang, Y. Jin and W. Wen, (Jan. 2018). *Security Analysis and Enhancement of Model Compressed Deep Learning Systems under Adversarial Attacks*, Proc. IEEE 23rd Asia and South Pacific Design Automation Conference (ASP-DAC 2018).

Z. Liu, Q. Liu, T. Liu, N. Xu, X. Lin, Y. Wang and W. Wen, (Jun. 2019). *Feature Distillation: DNN-Oriented JPEG Compression Against Adversarial Examples*, 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2019).

Z. Liu, X. Xu, T. Liu, Q. Liu, Y. Wang, Y. Shi, W. Wen, M. Huang, H. Yuan and J. Zhuang, (Jun. 2019). *Machine Vision Guided 3D Medical Image Compression for Efficient Transmission and Accurate Segmentation in the Clouds*, 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2019).

Z. Liu, T. Liu, W. Wen, L. Jiang, J. Xu, Y. Wang and G. Quan, (Jun. 2018). *DeepN-JPEG: A Deep Neural Network Favorable JPEG-based Image Compression Framework*, Proc. ACM/IEEE 55th Design Automation Conference (DAC 2018).

Z. Liu, T. Liu, J. Guo, N. Wu and W. Wen, (Jul. 2018). *An ECC-Free MLC STT-RAM Based Approximate Memory Design for Multimedia Applications*, Proc. IEEE Computer Society Annual Symposium on VLSI (ISVLSI).