Exploration of Security and Privacy Challenges through Adversarial Weight Perturbation in Deep Learning Models

by

Adnan Siraj Rakin

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved July 2022 by the
Graduate Supervisory Committee:

ARIZONA STATE UNIVERSITY

August 2022

ABSTRACT

Adversarial threats of deep learning are increasingly becoming a concern due to the ubiquitous deployment of deep neural networks(DNNs) in many security-sensitive domains. Among the existing threats, adversarial weight perturbation is an emerging class of threats that attempts to perturb the weight parameters of DNNs to breach security and privacy.

In this thesis, the first weight perturbation attack introduced is called Bit-Flip Attack (BFA), which can maliciously flip a small number of bits within a computer's main memory system storing the DNN weight parameter to achieve malicious objectives. Our developed algorithm can achieve three specific attack objectives: i) Un-targeted accuracy degradation attack, ii) Targeted attack, & iii) Trojan attack. Moreover, BFA utilizes the rowhammer technique to demonstrate the bit-flip attack in an actual computer prototype.

While the bit-flip attack is conducted in a white-box setting, the subsequent contribution of this thesis is to develop another novel weight perturbation attack in a black-box setting. Consequently, this thesis discusses a new study of DNN model vulnerabilities in a multi-tenant Field Programmable Gate Array (FPGA) cloud under a strict black-box framework. This newly developed attack framework injects faults in the malicious tenant by duplicating specific DNN weight packages during data transmission between off-chip memory and on-chip buffer of a victim FPGA. The proposed attack is also experimentally validated in a multi-tenant cloud FPGA prototype.

In the final part, the focus shifts toward deep learning model privacy, popularly known as model extraction, that can steal partial DNN weight parameters remotely with the aid of a memory side-channel attack. In addition, a novel training algorithm is designed to utilize the partially leaked DNN weight bit information, making the model extraction attack more effective. The algorithm effectively leverages the partial leaked bit information and generates a substitute prototype of the victim model with almost identical performance to the victim.

DEDICATION

This thesis is dedicated to my father & mother.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Dr. Deliang Fan at Arizona State University for his high-quality advising and mentoring. I am grateful for his invaluable suggestions and ideas, which helped form my research structure and directions. I would commend his patience and help me develop as a researcher through continuous encouragement and appreciation. In addition, I sincerely thank the graduate advisory committee Dr. Chaitali Chakrabarti, Dr. Jae-sun Seo and Dr. Yu (Kevin) Cao, for their support and valuable suggestions.

Also, thanks to all of my co-workers and their contribution to forming this thesis. Special thanks to Zhezhi He for being a mentor and guide at the early stages of my PhD. I want to thank Shaahin, Li, Jingtao and my other colleagues who have given their valuable expertise and suggestions in helping me throughout my PhD.

Last but not least, I want to thank my wife for her continuous support during my PhD journey.

This work is partly supported by the National Science Foundation under Grant No. 2019548.

TABLE OF CONTENTS

Page

LIST OF TABLES x

LIST OF FIGURES xiv

CHAPTER

1 INTRODUCTION 1

1.1 Adversarial Threats in Deep Learning 1

1.1.1 Security Challenges 2

1.1.2 Privacy Challenges 4

1.2 Motivation of Adversarial Weight Perturbation 5

1.3 Contributions 6

1.3.1 Adversarial Weight Perturbation in Memories Through Side-Channel Attacks 7

1.3.2 Black-box Adversarial Weight Perturbation in Multi-tenant Cloud FPGA 8

1.3.3 Attacking Privacy of Deep Learning Models Via Remote Side Channel 9

1.4 Dissertation Structure 10

2 BIT-FLIP BASED ADVERSARIAL WEIGHT ATTACKS AND DEFENSES 12

2.1 Background 12

2.1.1 What Is Adversarial Attack (Input Example)? 12

2.1.2 Prior Adversarial Weight Attack 13

2.1.3 Rowhammer Attack 14

2.1.4 Weight Quantization and Encoding 15

2.1.5 Adversarial Weight Attack Threat Model 16

2.2 Bit-Flip Attack (BFA) 17
2.2.1 Un-targeted BFA Attack (U-BFA) Objective 17
2.2.2 Targeted BFA Attack (T-BFA) Objectives 18
2.2.3 Vulnerable Weight Bits Searching Algorithm of BFA 19
2.3 Experimental Setup 22
2.3.1 Dataset 22
2.3.2 Dataset Configuration for Targeted Attack 22
2.3.3 DNN Architectures 24
2.3.4 BFA Attack Setup in a Real Computer 24
2.3.5 Evaluation Metrics 25
2.4 Experimental Results 26
2.4.1 Un-targeted BFA Attack Results. 26
2.4.2 Targeted Bit-flip Attack Results on CIFAR-10 26
2.4.3 Targeted Bit-flip Attack Results on ImageNet 30
2.4.4 Comparison with Other Competing Methods 31
2.4.5 Analysis of Attacking Real Computer Running DNNs 31
2.5 Discussion 33
2.5.1 Evaluation Against Existing Defense 33
2.5.2 Effect of Adversarial Training 34
2.5.3 Layer-wise Sensitivity. 35
2.5.4 Summary of Potential Defenses 36
2.6 Conclusion 37
3 BIT-FLIP ATTACK TO INJECT TROJAN IN DEEP NEURAL NETWORK 38

3.1 What is A Trojan Attack? 38
3.2 Previous Trojan Attacks and Their Limitations 39
3.3 Threat Model for TBT 40
3.4 Targeted Bit Trojan (TBT) 42
3.4.1 Trigger Generation 42
3.4.2 Trojan Bit Search (TBS) 44
3.5 Experimental Setup: 45
3.5.1 Evaluation Metrics 46
3.6 Experimental Results 47
3.6.1 CIFAR-10 Results 47
3.6.2 ImageNet Results: 48
3.6.3 Ablation Study 49
3.6.4 Comparison to Other Competing Methods. 51
3.7 Discussion 52
3.8 Conclusion 55
4 BLACK-BOX ADVERSARIAL WEIGHT ATTACK IN MULTI-TENANT FPGA 56
4.1 Threat Model and Attack Vector 56
4.2 Attack Objective Formulation 59
4.3 Deep-Dup Framework 60
4.3.1 Awd Attack in Multi-tenant FPGA 60
4.3.2 P-DES Searching Algorithm 62
4.3.3 End-to-end Attack Framework 67
4.4 Experimental Setup 69

4.4.1 Dataset and DNN Models 69
4.4.2 FPGA Prototype Configurations 70
4.4.3 Evaluation Metric and Hyper-parameters 71
4.5 Experimental Validation and Results 73
4.6 Comparison to Other Methods 73
4.6.1 Attack Time Cost 74
4.7 Potential Defense Analysis 75
4.8 Conclusion 78
5 PRIVACY OF DEEP LEARNING MODELS 80
5.1 Model Extraction Attack 81
5.2 Threat Model for Model Extraction 83
5.3 Overview of DeepSteal 85
5.4 Hammerleak: Efficient Data Stealing in Memories 86
5.5 Substitute Model Training with Mean Clustering 88
5.5.1 Hammer Leaked Data Filtering 88
5.5.2 Mean Clustering Optimization 89
5.5.3 Overall Training Algorithm 90
5.6 Experimental Setup 91
5.6.1 Attack Evaluation Metrics 91
5.6.2 Hardware Configuration 92
5.7 Evaluation 93
5.7.1 Hammerleak Performance Analysis 93
5.7.2 DeepSteal Experimental Results: CIFAR-10 93
5.7.3 Comparison to State-of-the-art Techniques 97

CHAPTER Page

5.7.4 Impact of Bit Stealing Errors 98

5.8 Discussion 99

5.8.1 Countermeasures for DeepSteal 99

5.9 Conclusion 100

6 CONCLUSION AND OUTLOOK 102

REFERENCES 104

LIST OF TABLES

Table Page

1.1 Three Primary Class of Adversarial Security and Privacy Threats in DNN 1

2.1 Threat Model of Bit-flip Attack (1; 2; 3). 17

2.2 Test Data Splitting to Conduct Targeted Attack from Source Class t_{p} to Target Class t_{q}. CIFAR-10 Data Has 10k Test Images With Each Class Containing 1000 Test Images and The ImageNet Dataset Has 50k Test Samples with Each Class Containing 50 Images. Note: (t_{r}) Means Images Belong to Any Other Class Apart from The Source Class (3). . . 23

2.3 Pre-attack Test Accuracy of Individual Class (i). We Also Report The Test Accuracy w/o Any Sample From Class i For Both Resnet-20 and Vgg-11 Model (3). 23

2.4 We Summarize The Results of U-BFA (2) 27

2.5 N-to-1 Attack: Number of Bit-flips (mean±std) Required to Classify All The Input Images to a Corresponding Target Class With 100% ASR. In Each Case, Test Accuracy Drops to 10% (3). 27

2.6 Performance of T-BFA Variants on ImageNet (from Hen Class (i.e., Label 8) to Goose Class (i.e., Label 99)). The Original Test Accuracies of ResNet-18, ResNet-34 and MobileNet-V2 Are 69.23%, 75.5% And 72.01%, Respectively (3). 30

2.7 Comparison with Competing Methods. We Directly Report The Numbers from the Respective Papers for (3; 4; 5). 31

2.8 T-BFA Attack on DNNs Running in A Real Computer (3) 32

2.9 T-BFA Performance Against Existing BFA Defense Techniques (6). PTA Indicates Post-attack Test Accuracy (3). 33

Table Page

2.10 Summary of Possible Directions To Improve Resistance Against BFA (3). 36

3.1 CIFAR-10 Results: Vulnerability Analysis of Different Class on ResNet-18. TC Indicates Target Class Number. In This Experiment We Chose w_b To Be 150 And Trigger Area Was 9.76% for All The Cases (7). 48

3.2 ImageNet Results on ResNet-18 Architecture (7). 49

3.3 Trigger Area Study: Results on CIFAR-10 for Various Combination of Targeted Trojan Trigger Area (7). 49

3.4 Number of Weights Study: Results on CIFAR-10 for Various Combination of Number of Weights Changed w_b for ResNet-18 (7). 50

3.5 Comparison to The Baseline Methods: Here We Used VGG-16 architecture. Before Attack Means The Trojan Is Not Inserted into DNN Yet. It Represents The Clean Model's Test Accuracy (7). 51

3.6 Comparison of Different Trigger Location: We Perform Trigger Position Analysis on Target Classes 3,4,6,7 As We Found Attacking These Classes Are More Difficult in Table 3.1.TC Means Target Class (7). 54

4.1 Black-box Targeted Attack Results for ImageNet (8). 72

4.2 Black-Box Attack for Object Detection. 72

4.3 Comparison of Deep-Dup with Random AWD Attack And Row-hammer Based (BFA (1; 2)) Attack. All The Results Are Presented for 8-bit Quantized VGG-11 Model (1) (8). 74

4.4 Attack Efficiency after Increasing The Model Size of ResNet-20 and VGG-11 Model by 4 (i.e., Increasing Each Input And Output Channel Size by 2) (8). 76

4.5 Deep-Dup Attack Performance after Protecting or Securing Some Critical DNN Layers (8) 77
4.6 Weight Package Randomization As Obfuscation. Pre-defined Shuffle : Shuffling The Weight Packages In A Pre-defined Order before Transmission. Random Shuffle: Shuffling The Weight Packages Every Time Using A Random Function Before Transmission (8). 77
5.1 Summary of The Existing Model Extraction Methods (9). 82
5.2 List of Information Accessible to The Attacker for Substitute Model Training (9). 85
5.3 Summary of CIFAR-10 Results for Three Different DNN Architectures. We Report Two Different Cases of DeepSteal Attack i) All Bits: Where We Use All The Bit Information (i.e., All 8 Plots) Plotted in Figure 5.4. According To This Plot, for Each # of HammerLeak Attack Rounds Along x-axis, We Take The Percentage of Bits Recovered for All 8 Plots (e.g., MSB, MSB+2nd MSB & So On). ii) MSB: We Only Use The MSB Bit Information Labeled As MSB Curve in Figure 5.4 (9). . . 95
5.4 We Evaluate DeepSteal Attack Against State-of-the-art Techniques Across Three Different Domains As Case Studies. In Each of The Cases, Only Our Attack Performs On Par with The SOTA Methods Across All Three Evaluation Metrics (9). 96

Table Page

5.5 After Adversarial Training, The White-box Accuracy Under Attack Improved to 43.12% & 35.71% for ResNet-18 and VGG-11 Models Respectively . Here We Report The Performance of DeepSteal Attack Using Recovered Bit Information after 4000 Rounds of HammerLeak Attack (9). 100

LIST OF FIGURES

Figure Page

1.1 Summary of Potential Security And Privacy Challenges for Deep Learning Models. To Breach The Security of DNN Models, The Attacker Adds Small Noise At The Input/ Weights To Achieve Desired Malicious Objectives. To Break The Privacy of The User Or Service Provider, The Attacker May Steal Or Reproduce Training Data/ Model Architecture/ Weight Information (Presented by (9) at IEEE Security & Privacy 2022). 3

1.2 Randomly Bit-Flips of A ResNet-18 Architecture On ImageNet. After Flipping 100 Random Bits The Network's Accuracy Does Not Degrade Significantly (1) 5

2.1 Overview of BFA Attack Setup (3) 16

2.2 Three Different Kinds of Attack Objective for T-BFA (3). 17

2.3 Overview of BFA Searching Algorithm (3). 20

2.4 Type II: 1-to-1 Attack on ResNet-20 Between Source Class And Target Class. The Left Subplot Shows Post Attack Test Accuracy And The Right Subplot Shows Average Number of Bit-Flips Required for The Attack (3). 28

2.5 Type III: 1-to-1 (S) Attack Post Attack Test Accuracy, Attack Success Rate And Avg. # of Bit-Flips for Five Rounds of Attacks for Both Resnet-20 and VGG-11 Networks (3). 29

2.6 Summary of Attacking Adversarial Trained ResNet-20 Model with N-to-1 And 1-to-1 Attack. We Report The # of Bit-flips Required to Reach $\sim$ 100 % ASR. The Source Class Is 3 and Target Class Is 5. Here, We Report the Average of Five Individual Rounds (3). 34

Figure Page

2.7 Summary of Attacking Adversarial Trained ResNet-20 Model with Type III 1-to-1 (S). We Report The Post Attack Accuracy And # of Bit-flips Required to Reach $\sim$ 99.0 % ASR for All Cases. The Source Class Is 3 And Target Class Is 5 (3). 34

3.1 Overview of Targeted Trojan Attack (7) 39

3.2 Overview of TBT Attack's Threat Model (7) 41

3.3 Flow Chart of Effectively Implementing TBT (7) 42

3.4 ASR (Green) And TA (Blue) vs Number of Bit Flips Plot. Only with 84 Bit Flips TBT Can Achieve 92 % Attack Success Rate (7). 53

4.1 Threat Model for Deep-Dup Attack (8). 57

4.2 Illustrated Timing Diagrams of DNN Model Transmission W/O or Under AWD Attack. (a) Each DNN Weight Package (Di) Is Transmitted And Received in A Separate Clock Cycle. (b) Voltage Glitch Incurs More Propagation Delay To The Transmission of D2, Which Also Shortens The Next Package D3. As A Result, The Data Package D2 is Sampled Twice by The Receiver Clock, Injecting Faults To The Received Data Package (8). 61

4.3 Overview of Our Adversarial Weight Index Searching (P-DES) Algorithm (8). 64

4.4 Overview of End-to-End Deep-Dup Attack Framework (8) 69

4.5 Experimental Setup And Results of Deep-Dup Black-box Attack on YOLOv2, with 'Person' As Target Group. After Attack, The Fault-Injected YoLov2 Model Fails To Recognize The 'Person' (8). 70

Figure Page

4.6 Black-Box Attack Time Cost Analysis with $z = 100$. FPGA Acceleration (i.e., Fitness Function Evaluation) Time And Mutation Generation Time Are Reported (8). 75

5.1 Overview of The DeepSteal Attack Framework. Stage-1: Exfiltrating DNN Partial Weight Bits Efficiently Through Exploiting Memory Fault Vulnerabilities (HammerLeak). Stage-2: with The Recovered Bits, Training A Substitute Model using Mean Clustering Weight Penalty (9). 85

5.2 Data Dependency for Inducing a Rowhammer Fault. Here, Based on The Presence of Bit Flip in The Attacker-controlled Vulnerable Bit in The Target Row (T_r), Data from Adjacent Row from Victim Program Can Be Inferred (9) 86

5.3 First Row: N-Bit Quantized Weight Level; Second Row: Once The MSB of Weight W_t in The Victim Model Is Leaked, We Can Narrow Down The Projected Range of W_t in The Substitute Model; Last Row: Leaking All The Bits Can Track Down The Exact Value of W_t for The Substitute Model Training (9). 88

5.4 Percentage of Weights with MSB or More Bits Recovered. +x Denotes Number of Consecutive Higher Order Bits Recovery (i.e., +3 Represents Weights with All Three MSB Bits Recovered) (9). 94

5.5 Analysis of The Impact of The Recovered Bit Error Rate (%) on Deep-Steal Attack Performance for ResNet-18 (9). 99

Chapter 1

INTRODUCTION

1.1 Adversarial Threats in Deep Learning

Deep Neural Networks (DNNs) have achieved great success in a wide range of applications including but not limited to image classification (10; 11; 12; 13), object detection (14) and speech recognition (15) tasks. Due to ever-increasing interactions between intelligent agents and human activities that are *security and safety critical*, maintaining security objectives (e.g., confidentiality and integrity) has become the first-order design consideration for DNN systems (16). First observed by (17) and (18) that DNNs are vulnerable to adversarial examples, which can be generated by adding imperceptible noise to the input. More recently, adversarial threats have evolved to pose an even greater security (19; 1; 20; 21; 2) and privacy (22; 23; 24; 25; 26) concerns for the practical deployment of deep learning.

Table 1.1: Three Primary Class of Adversarial Security and Privacy Threats in DNN

Model Tampering Attack (Security Threat)	Model Leakage Attack (Privacy Threat)
Adversarial Input Example Attack (27; 18)	Model Inversion Attack (26; 28)
Adversarial Weight Perturbation Attack (1; 2)	Membership Inference Attack (22; 25)
Trojan/Backdoor Attack (29; 19)	Model Extraction Attack (23; 30)

This thesis highlights two critical potential safety challenges (Table 1.1) for deep learning models in practice. First, *model tampering attack* where the adversary modifies external (e.g., data) or internal parameters (e.g., weights) to cause malfunction of the models and thus compromising the security. & ii) *model leakage attack* where the adversary instead attempts to retrieve the secret information (e.g., data/model) to hamper the privacy of the user or the DNN service provider.

1.1.1 Security Challenges

The security challenges of deep learning primarily come from malicious adversaries tampering with the external (e.g., data) or internal (e.g., model weight) information of a DNN. Such security threats can be broadly classified into three major categories as highlighted in Table 1.1.

- *Adversarial Input Example (27; 18):* Adversarial input attack adds imperceptible malicious noise to the input data to fool a target DNN model (shown in Fig. 1.1). A series of works have been conducted in exposing this underlying vulnerability of DNN through developing advanced attack methodology (31; 30; 32) and counter defensive solutions (33; 34; 35; 36; 37). To date, adversarial input example noise remains a potent threat for practical AI deployment (38).

- *Adversarial Weight Perturbation (1; 21; 2):* In contrast to external input attack, malicious weight perturbation can attack the DNN model's internal parameters (e.g., weights/biases) to achieve similar malicious attack objectives. With the increase of advanced fault injection techniques such as rowhammer attack (39), laser beam attack (40) or under-voltage attack (41), internal parameter modification is increasingly becoming more practical. Unlike adversarial input attack, weight perturbation has received minimal attention; hence, more recently a series of new class of weight perturbation (3; 8; 1; 2) attack and its protective measures (42; 43; 6; 44; 45) have been discussed in the literature.

- *Trojan/Backdoor Attack (19; 20; 29):* Finally, the Trojan/Backdoor attack attempts to modify both input and weight parameters to achieve desired target attack objective. Recently, the development of advanced Trojan attacks (19; 20) and their defenses (46; 47) has been a central focus of DNN security study. The key challenge in prior Trojan works requires access to training facilities/supply chain to inject Trojan into the model.

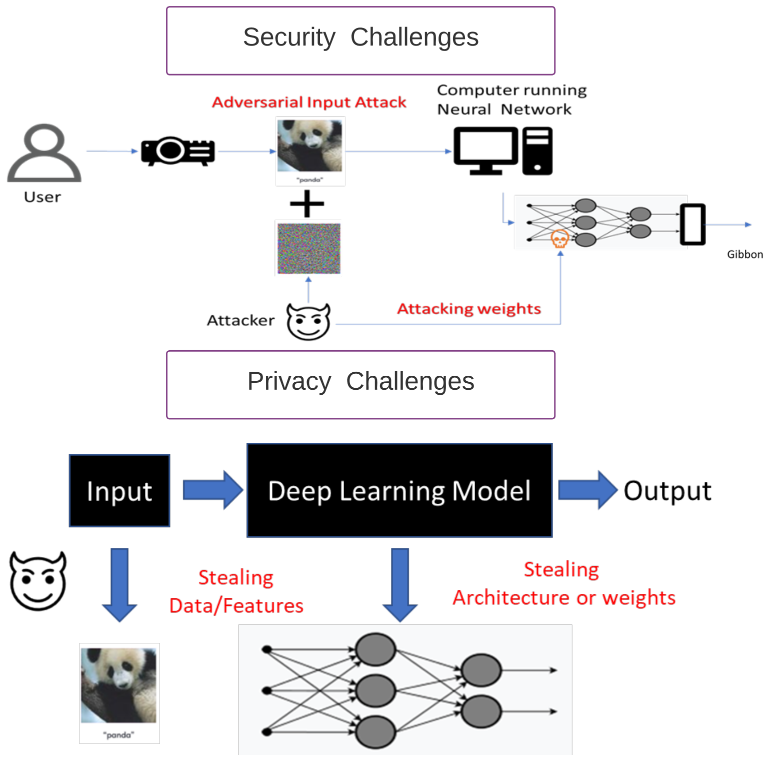


Figure 1.1: Summary of Potential Security And Privacy Challenges for Deep Learning Models. To Breach The Security of DNN Models, The Attacker Adds Small Noise At The Input/ Weights To Achieve Desired Malicious Objectives. To Break The Privacy of The User Or Service Provider, The Attacker May Steal Or Reproduce Training Data/ Model Architecture/ Weight Information (Presented by (9) at IEEE Security & Privacy 2022).

1.1.2 Privacy Challenges

Deep learning models are being trained with massive computational resources (e.g., data, model, GPU). Hence protecting the trained model or training data from adversaries has become a top priority to ensure Intellectual Property (IP) protection of commercial DNN models. Again, this thesis categorizes the privacy challenges of deep learning models into three classes.

- *Model Inversion Attack (26; 28):* The aim of model inversion attack is to identify key information (e.g., feature, attribute, demographic information) about the training data. In particular, this attack is a major concern for user private data protection in collaborative learning environment such as split/federated learning (48; 49).

- *Membership Inference Attack (22; 25):* It is a query-based attack on a pre-trained deep learning model which allows an attacker to predict whether a particular sample was part of the training data distribution. A membership inference attack allows an attacker to identify detailed training samples, which may have severe consequences in privacy-sensitive domains such as medical applications (50).

- *Model Extraction Attack (23; 51):* Model extraction attack attempts to leak key information of deep learning models such as architecture or model parameters. Most prior extraction attack (23; 51) focuses on architecture only recovery. In contrast, recovering the fine-grained weight information (30) of large DNN models has received very little attention.

Motivated by the above challenges for the safe deployment of DNN, this thesis introduces several critical security/privacy attacks and defences in deep learning applications.

1.2 Motivation of Adversarial Weight Perturbation

Recently, adversarial weight perturbation attacks have been added to the security challenge of DNN models due to the security concern of model leakage and malicious fault injection into the computer system. First, the DNN model running on a computer is not secure. Many advanced computer side-channel attacks (52; 53; 51; 54) have successfully extracted DNN model parameters. Second, due to its large size, DNN model integrity is difficult to guarantee in state-of-the-art performance-driven computing systems. In such systems, there are many methods to inject a small amount of fault into the computing memory or path of DNN without alerting the computing system. For example, memory fault injection techniques such as Laser Beam Attack (55) or Row-Hammer Attack (RHA) (56; 57), can inject faults into a computer's main memory (i.e., DRAM), causing severe threat to DNN computation.

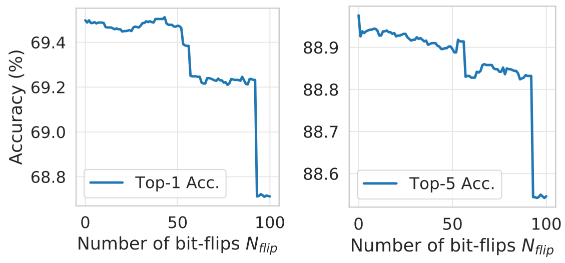


Figure 1.2: Randomly Bit-Flips of A ResNet-18 Architecture On ImageNet. After Flipping 100 Random Bits The Network's Accuracy Does Not Degrade Significantly (1)

In particular, Hong et al. (58) have shown that single-bit corruptions in DNN model parameters can considerably degrade the inference accuracy of several DNN models. Their attack study is performed on full-precision (i.e., floating-point numbers) DNN models where a single bit flip in the exponent field (i.e., the most-significant bit) of a parameter can result in orders of magnitude change in the parameter value. Note that *quantized deep neural networks* (59), are more robust to single-bit corruption. This is because model quantization replaces full-precision model parameters with low bit-width integers or even binary representations, which significantly limit the magnitude of possible parameter value range (60; 61). In addition, due to the impressive improvement in energy efficiency, memory footprints and storage, model quantization is now the widely applied optimization in deep neural networks (62; 63; 64; 65; 66). Our initial investigation (1) in Fig. 1.2 also reveals that a random fault attack (a bit-flip or weight corruption) in the quantized model weights does not introduce any observable accuracy loss even after 100 random flips.

Inspired by these observations, in this thesis, we focus on investigating the vulnerability of DNN model parameters from fault injection attacks. We aim to systemically characterize how weight perturbation of model parameters can influence the accuracy of well-trained quantized deep neural networks. We further look to develop algorithms that can potentially cause targeted miss-classification in deep learning classification problems. We designed several novel attack algorithms using both system and algorithm level optimization to achieve this. Our investigation also includes a comprehensive analysis of a novel defense framework as a potential countermeasure to weight perturbation attacks. Finally, we also look to leverage model weight perturbation to hijack sensitive privacy information (e.g. weights, biases & gradients) of a deep learning model, thus exposing a novel privacy concern for modern deep learning applications.

1.3 Contributions

The contributions of this thesis can be summarized into three major parts: i) Adversarial weight perturbation attack (i.e., Un-targeted, Targeted & Trojan) in memories through side-channel attacks in a white-box setting, ii) Model weight tampering attack in multi-tenant FPGA under a strict black-box setting, and iii) Finally, leveraging memory fault (i.e., bit-flip) through remote side-channel attack to leak secret DNN weight information of a large-scale deep learning model.

1.3.1 Adversarial Weight Perturbation in Memories Through Side-Channel Attacks

Due to the existing vulnerabilities, several recent works have leveraged such memory fault injection techniques to inject minor faults (probably a few bits of error) into main computer memory (i.e. DRAM) to modify the stored DNN model slightly, successfully hijacking the running DNN function (1; 4; 21; 2). Even so, the memory bit-flip based malicious un-targeted weight attack (*BFA*) in (1), and (2) has been experimentally demonstrated to cause severe accuracy degradation of a fully-functional 8-bit quantized ResNet-18 on the ImageNet dataset to 0.1% with only 13 bit-flips (out of 93 million bits), in a real computer system. which mainly introduce un-targeted bit-flip attack algorithm and its implementation in a real computer system, respectively. A more potent version of the BFA is *Targeted Bit-Flip Attack* (T-BFA) (67) methodology is the first work of bit-flip-based targeted adversarial weight attack on weight-quantized DNNs. In Chapter 2, we will introduce all the variants of BFA (e.g., un-targeted or targeted). Additionally, we will discuss the practical feasibility of BFA in real computer attacks considering the adversary performs the attack by running an unprivileged user-space process on the machine (i.e., a strong adversary with system-level access permission is not required). To conclude, we will highlight a list of effective defenses against BFA attacks.

In addition, we will discuss another novel adversarial parameter attack in Chapter 3 to inject neural Trojan into a clean DNN model called *Targeted Bit Trojan* (TBT) (7). It first utilizes *Neural Gradient Ranking* (NGR) algorithm to identify certain vulnerable neurons linked to a specific target class. Once the attacker identifies the vulnerable neurons, with the help of NGR, the attacker can generate a trigger delicately designed to force target neurons to fire large output values. Such an algorithm enables efficient Trojan trigger generation, where the generated trigger is specifically designed for a targeted attack. Then, TBT locates certain vulnerable bits of DNN weight parameters through *Trojan Bit Search (TBS)*, with the following objectives: After flipping these sets of weight bits through row-hammer, the network maintains on-par inference accuracy w.r.t the clean DNN counterpart, when the designed trigger is absent. However, a trigger in the input data forces any input to be classified into a particular target class.

1.3.2 Black-box Adversarial Weight Perturbation in Multi-tenant Cloud FPGA

For high efficiency and performance, there have been growing efforts to support multiple independent tenants co-residing/sharing an FPGA chip over time or simultaneously (68; 69). The *co-tenancy* of multiple users on the same FPGA chip has created a unique attack surface, where many new vulnerabilities will appear and cause dangerous effects. With many hardware resources being jointly used in the multi-tenant FPGA environment, a malicious tenant can leverage such *indirect* interaction with other tenants to implement various new attacks. However, as a relatively new computing infrastructure and one of the leading hardware accelerator platforms, the *security of multi-tenant FPGAs for DNN acceleration* has not been investigated in-depth. In this thesis, we will introduce an end-to-end Deep-Dup attack framework in Chapter 4, one type of adversarial DNN model fault injection attack, utilizing our DNN vulnerable parameter searching software (i.e. P-DES) to guide and search when/where to inject fault through multi-tenant FPGA hardware fault injection (i.e. AWD) for efficient and effective un-targeted/targeted attacks (i.e., un-targeted attack to degrade overall accuracy and targeted attack to degrade only targeted group accuracy). To maximize attack efficiency, i.e. conducting AWD-based fault injection into the most vulnerable DNN weight data packages for any given malicious objective, we design a generic vulnerable weight package searching algorithm, called *Progressive Differential Evolution Search (P-DES)*. Unlike prior weight perturbation works, which were only demonstrated in a deep learning white-box setup (2), this new form of attack succeeded in a complete black-box threat model.

1.3.3 Attacking Privacy of Deep Learning Models Via Remote Side Channel

DNN models typically take a tremendous amount of resources to train, and in many cases, the training relies on using valuable domain-specific data. As a result, DNN models are regarded as the *top intellectual properties* (IP) for machine learning (ML) service providers and model owners (70). One major threat to the IP of these models is a model extraction attack that aims to infer or steal critical information from DNN models to achieve certain malicious goals (71). Recent advances in hardware-based exploitation have shown that adversaries can leverage side-channel attacks to gain sensitive information in computing systems (72; 73; 74; 75).

The critical challenge in stealing a DNN model is its enormous (with millions of parameters) size; even with a hardware-based attack that can recover certain model weight information, it is typically impractical to assume that *the entire weights* can be exfiltrated in practical settings. Moreover, prior works (1; 2) have shown that variations on only tens out of millions of weight parameters will completely malfunction a DNN model. In this case, whether partial information of model weights can be effectively leveraged to build a more potent model extraction attack is uncertain. To address this, we will introduce a new attack framework called *DeepSteal* in Chapter 5 , an advanced model extraction attack framework using efficient model weight stealing with the aid of rowhammer-based side channels. Our attack aims to recover (partial) weight parameters of a target DNN model (i.e., victim model), which will be harnessed to build applicable substitute models using novel learning schemes. Notably, we leverage the well-known rowhammer fault attack (56) as the information leakage attack vector. Our exploitation is motivated by prior studies showing that rowhammer-induced fault in a memory cell highly depends on the data pattern from its neighbouring cells (76; 2). After recovering the partial information, the weight search space of a victim model remains high. For instance, even after recovering 90% of the bits in a large model like VGG-11 (11) (i.e., 1056 Million bits for an 8-bit model), the attacker still needs to train the recovered model with limited data to restore the remaining 10% bits (i.e., 105.6 million bits). To address the additional challenges, we develop a novel substitute model training algorithm with *Mean Clustering* weight penalty. The purpose of such a loss penalty term is to utilize the recovered partial weight bits for effectively guiding the substitute model training. Subsequently, DeepSteal produces a substitute model that achieves similar accuracy as the victim model with high fidelity. Moreover, the trained substitute model could help mount strong adversarial input attacks on the victim model.

1.4 Dissertation Structure

The rest of this dissertation document is organized as follows:

- **Chapter 2** presents a novel adversarial weight perturbation attack algorithm, Bit-flip attack (BFA), against deep neural networks. It contains materials from ”T-BFA: Targeted bit-flip adversarial weight attack” published at T-PAMI 2021 (3). The dissertation author was the investigator and author of these papers.

- **Chapter 3** presents a novel Trojan attack algorithm called Targeted Bit Trojan (TBT) motivated by the rowhmamer-based bit-flip attack. It contains materials from "Tbt: Targeted neural network attack with bit trojan" published at CVPR 2020 (7). The dissertation author was the investigator and author of the paper.

- **Chapter 4** presents a novel black-box attack against deep learning models on a multi-tenant FPGA platform. It contains materials from "Deep-Dup: An adversarial weight duplication attack framework to crush deep neural network in Multi-Tenant FPGA" published at USENIX security 2021 (77). The dissertation author was the investigator and author of the paper. Special thanks to Yukui Luo for his valuable contribution to AWD attack in (8).

- **Chapter 5** shows a novel model extraction attack which introduces a novel algorithm to generate a successful substitute model leveraging bit-flip in memories. It contains materials from "DeepSteal: Advanced Model Extractions Leveraging Efficient Weight Stealing in Memories", published at IEEE S&P 2022 (9). The dissertation author was the investigator and author of the paper. Special thanks to M Hafizul Islam Chowdhuryy for his valuable contribution to HammerLeak Attack in (9).

Chapter 2

BIT-FLIP BASED ADVERSARIAL WEIGHT ATTACKS AND DEFENSES

In recent years, deep neural networks (DNNs) have achieved tremendous success in a wide variety of applications, including image classification (78; 79), speech recognition (15; 80) and machine translation (81; 82). Unfortunately, DNN models are not secure and have been exposed to the vulnerability of adversarial input example attack (27; 18; 35). Recently, internal model perturbation, i.e., adversarial weight attacks (21) has also been added to the security challenge due to the improvement in model leakage attack (i.e., making the threat model easier) and malicious fault injection attacks into the computer system. This chapter provides a complete overview of one type of adversarial weight attack on deep learning models called *Bit-Flip Attack (BFA)*. It discusses two major types of BFA attack: Un-targeted BFA (1; 2) & Targeted BFA (3). At the end of the chapter, we also summarise possible defense mechanisms against BFA (6; 3).

2.1 Background

2.1.1 What Is Adversarial Attack (Input Example)?

Despite the remarkable progress of deep learning, recent studies (17; 18; 83) have shown that DNNs are vulnerable to adversarial examples. In image classification, an adversarial example is a carefully crafted image that is visually imperceptible to the original image but can cause DNN model to misclassify. In addition to image classification, attacks to other DNN-related tasks have also been actively investigated, such as visual QA (84), image captioning (85), semantic segmentation (86), machine translation (87), speech recognition (88), and medical prediction (89).

There is a cohort of works on generating adversarial attacks and developing corresponding defense methods. The adversarial attacks can be grouped into two major categories: (i) *white-box attack* (17; 83), where the adversary has full access to the network architecture and parameters, and (ii) *black-box attack* (90; 91; 92), where the adversary can access the input and output of a DNN but not its internal configurations. Many attack algorithms have been proposed to generate adversarial examples (17; 93; 94; 95; 96; 31; 83). Among them, the fast gradient sign method (FGSM) (18) is one of the pioneering and most popular white-box attack algorithms. It uses the sign of gradients with respect to the input to generate adversarial examples and is one of the most efficient attack algorithms. The projected gradient descent (PGD) is among the most potent white-box attacks to date (97). Besides, Carlini & Wagner (C&W) (83) attack is another powerful attack that can achieve nearly 100% attack success rate.

2.1.2 Prior Adversarial Weight Attack

Un-targeted Attack: We already discussed briefly in the previous chapter how the recent developments in memory fault injection attacks (56; 40) have made it feasible to conduct an adversarial weight attack for a DNN model running on a computer. Among them, a row-hammer attack (56) on Dynamic Random Access Memory (DRAM) is the most popular one since it can create a profile of memory bits stored inside the main memory (i.e., DRAM) and flip any bit of a given target address. The first few works that exploited row-hammer to attack DNN weights flipped the Most Significant Bits (MSB bits) of DNN parameters, such as the bias (4) or weight (21), and changed them to a significantly large value, thus degrading accuracy. However, those attacks were only evaluated on a model with full precision (i.e. floating point) parameters and failed in DNNs with quantized parameters.

Targeted Attack: A targeted attack has more precise control on the miss-classification behaviour and can cause higher calamity. It is a well-investigated technique in adversarial input attack domain (83; 92; 27). Recent adversarial model parameter attacks can also perform a targeted attack (5; 4). Again, some of them (5) require a larger value of L_0 norm (i.e., 900) for weight perturbation. Also, these attacks (5; 4) have been evaluated on a full-precision model, which has been reported in (21) as being easier to attack. Hence, our attack and defense evaluation is performed on a quantized 8-bit fixed precision model.

2.1.3 Rowhammer Attack

Rowhammer is a software-induced fault attack exploits DRAM disturbance errors via user-space applications (56). Specifically, it has been shown that accesses (i.e., activations) to specific DRAM rows can introduce electrical disturbance to the DRAM cells in the neighbouring rows, which accelerates the leakage of their charges in the capacitors (56; 98). An attacker can intentionally activate particular DRAM rows (whose data belongs to the attacker) frequently enough (i.e., hammering) to eventually cause bit flips in a victim's address space. Such attacks have been successfully demonstrated on commercial-off-the-shelf DRAM modules even with the presence of ECC features (99; 100). There are mainly three hammering techniques proposed in the literature: a) double-sided hammering (101; 98; 56; 102; 100): where two aggressor rows are frequently activated to induce fault in the middle row. b) single-sided hammering (102): where one adjacent row to the target row and another random row are activated repeatedly; and c) one-sided hammering (99): where one periodically-accessed row causes repetitive row activations under the *close-page* DRAM policy. Double-sided hammering is the most effective technique for inducing DRAM faults since it introduces the strongest disturbance.

2.1.4 Weight Quantization and Encoding

Previous adversarial weight attacks have indicated that (21) quantized DNNs are more resilient to weight bit-flips than full-precision counterparts. To prove the effectiveness of our attack against a more resolute quantized network, we perform the attack on quantized DNN models. Our quantization scheme is a layer-wise N-bits uniform quantizer for weight quantization. For each of the l-th layer, the quantization methodology can be described as:

$$\Delta w_l = \max(\ \mathbf{W}_l^{\mathrm{r}})/(2^{N-1}-1); \quad \mathbf{W}_l^{\mathrm{r}} \in \mathbb{R}^d \tag{2.1}$$

$$\mathbf{W}_l = \mathrm{round}(\mathbf{W}_l^{\mathrm{r}}/\Delta w_l)\cdot \Delta w_l \tag{2.2}$$

where d is the dimension of weight tensor, Δw_l is the step size of weight quantizer, $\mathbf{W}_l^{\mathrm{r}}$ is the full-precision weight of the corresponding quantized weight $\mathbf{W}_l$. To circumvent the non-differential function (in Eq. (2.2)), popular straight-through estimator (103) is used to perform the training.

In our hardware evaluation, the computing system stores the signed integer in two's complement representation. Given one weight element $w \in \mathbf{W}_l$, the conversion from its binary representation ($\boldsymbol{b} = [b_{N-1}, ..., b_0] \in \{0,1\}^N$) in two's complement can be expressed as:

$$w/\Delta w = bin(\boldsymbol{b}) = -2^{N-1}\cdot b_{N-1} + \sum_{i=0}^{N-2} 2^i \cdot b_i \tag{2.3}$$

With the conversion relation described by $bin(\cdot)$ in Eq. (2.3), we can inversely obtain the binary representation of weights $\mathbf{B}$ (i.e. binary data stored in main memory) from its fixed-point counterpart as well.

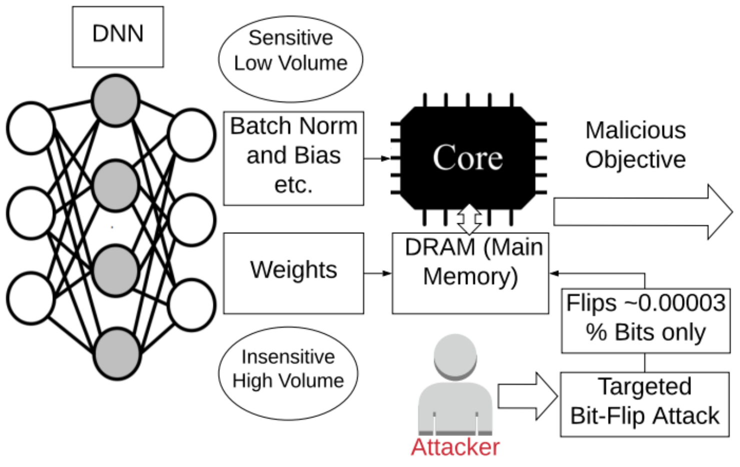


Figure 2.1: Overview of BFA Attack Setup (3)

2.1.5 *Adversarial Weight Attack Threat Model*

In this chapter, we follow the standard white-box attack threat model assumption for adversarial weight attacks, similar to adversarial input attacks (104; 27; 105). The BFA attack threat model is summarized in Table 2.1. It assumes the attacker has complete knowledge of the DNN model, i.e., architecture, neurons, weights and biases. Such a threat model is valid since previous works have demonstrated that an attacker can effectively steal similar information (i.e., layer number, weight size, and parameters) through side-channel attacks (52; 53; 51; 54; 23). In addition, the attacker can manipulate the model weights and cause bit-flips in the main memory (i.e., DRAM) as shown in Fig. 4.1. Since weights have large volumes with low sensitivity, they are stored in the DRAM. In contrast, low volume sensitive parameters (e.g., biases) are stored on-chip. An attacker can only flip (0 to 1 or 1 to 0) identified bits in memory; no manipulation of input data is allowed. The attacker has access to a portion of test data but is denied access to any form of training information (i.e., training dataset, hyper-parameters). The attacker can always compute gradient information when necessary in a white-box setting.

Table 2.1: Threat Model of Bit-flip Attack (1; 2; 3).

Access Required	Access **NOT** Required
DNN architecture & model parameters	Training configurations (i.e., hyper parameter).
A mini-batch of test data	Complete train/test datasets.

2.2 Bit-Flip Attack (BFA)

2.2.1 Un-targeted BFA Attack (U-BFA) Objective

Un-targeted Bit-Flip Attack (U-BFA) objective is to degrade the overall test accuracy of the DNN. The attack utilizes a gradient-based progressive search to locate a set of vulnerable weight bits (1; 2) and flip them in the memory. Thus at each iteration of the attack, the attacker will target maximizing the inference loss function$\mathcal{L}$ w.r.t true label $\boldsymbol{t}$ of a given test batch $\boldsymbol{x}$:

$$\max_{\{\hat{\mathbf{B}}_l^i\}} \mathcal{L}_{un}\Big(f\big(\boldsymbol{x}; \{\hat{\mathbf{B}}_l^i\}_{l=1}^{L}\big), \boldsymbol{t}\Big) \tag{2.4}$$

Here $l \in \{1, 2, ..., L\}$ is the layer index, $\hat{\mathbf{B}}_l^i$ is the bit representation of the weight matrix at the i^{th} iteration at layer l after flipping the bits in the original matrix $\mathbf{B}_l$.

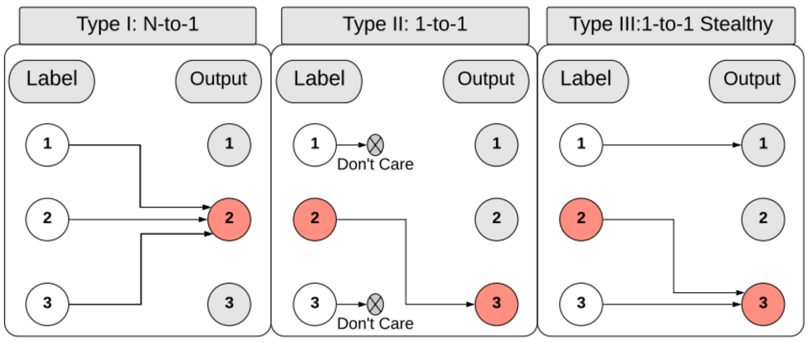


Figure 2.2: Three Different Kinds of Attack Objective for T-BFA (3).

2.2.2 *Targeted BFA Attack (T-BFA) Objectives*

- Type-I: N-to-1 Attack. Given that the input data belong to one of N-classes, the objective of this T-BFA variant is to force the entire dataset $\mathbb{X} = \{\mathbb{X}_i\}_{i=1}^{N}$ with all N classes (as source classes) to one adversary-selected target class. The objective function is formalized as:

$$\min \ \mathcal{L}_{\text{N-to-1}} = \min_{\{\mathbf{B}\}} \ \mathbb{E}_{\mathbb{X}} \mathcal{L}(f(\boldsymbol{x}, \{\mathbf{B}\}); \boldsymbol{t}_q) \tag{2.5}$$

where $\{\mathbf{B}\}$ is the quantized representation (in binary format) of weight tensor $\{\mathbf{W}\}$ stored in computer memory. Given vectorized input $\boldsymbol{x} \in \mathbb{X}$, $f(\boldsymbol{x}, \{\mathbf{B}\})$ computes quantized DNN inference output. $\mathcal{L}(\cdot;\cdot)$ denotes the cross-entropy loss between DNN inference output and labels. $\boldsymbol{x}$ and $\boldsymbol{t}$ are input data and their corresponding ground-truth label. For this attack, the ground-truth label term of source category[1] $\boldsymbol{t} \in \boldsymbol{e}^{(i)}, i \in \{1, ..., N\}$ is tampered to the selected q-indexed target category $\boldsymbol{t}_q \in \boldsymbol{e}^{(q)}$.

- Type-II: 1-to-1 Attack. In this T-BFA variant, adversary focuses on the misclassification of input data $\mathbb{X}_p$ of single p-indexed source category into the q-indexed target category ($p \neq q$), without caring about the impact on the remaining categories $\mathbb{X}_{i \neq p}$. It can be modeled as:

$$\min \ \mathcal{L}_{\text{1-to-1}} = \min_{\{\mathbf{B}\}} \ \mathbb{E}_{\mathbb{X}_p} \mathcal{L}(f(\boldsymbol{x}_p, \{\mathbf{B}\}); \boldsymbol{t}_q); \quad \boldsymbol{x}_p \in \mathbb{X}_p \tag{2.6}$$

The Type-II attack is a subset of Type I attack. However, such an objective is still practically valuable for only attacking a specific group or subset of inputs, where the type-I N-to-1 attack would flip many more unnecessary bits for all groups of inputs.

[1]$\boldsymbol{e}^{(i)}$ is the notation of one-hot code vector $[0, \ldots, 0, 1, 0, \ldots, 0]$ with a 1 at position i.

- Type-III: 1-to-1 Stealthy Attack. In addition to the type-II 1-to-1 attack described above, this type-III attack is a stealthy version with two objectives: **1)** All the input data from p-indexed category $\mathbb{X}_p$ are classified into q-indexed target category, which is the same as Eq. (2.6); **2)** Meanwhile, it needs to maintain correct predictions of the input data excluded from the source category $\mathbb{X}_j, j \in \{1,2,..,N\}\backslash\{p\}$. This type-III attack could be achieved via the optimization of the two corresponding loss terms in the RHS of the following objective function:

$$\begin{aligned} \min\ \mathcal{L}_{\text{1-to-1(S)}} = \min_{\{\mathbf{B}\}}\ \mathbb{E}_{\mathbb{X}}\Big(&\mathcal{L}(f(\boldsymbol{x},\{\mathbf{B}\});\boldsymbol{t}_q)\cdot \mathbf{1}_{\boldsymbol{x}\in\mathbb{X}_p} + \\ &\mathcal{L}(f(\boldsymbol{x},\{\mathbf{B}\});\boldsymbol{t})\cdot \mathbf{1}_{\boldsymbol{x}\in\mathbb{X}_j}\Big) \end{aligned} \tag{2.7}$$

where $\mathbf{1}_{\text{condition}}$ returns 1 if the condition is true, 0 otherwise.

For a practical adversarial attack, to minimize attack effort, a critical constraint is to use limited number of malicious bit-flips on weight bits to achieve above defined attack objectives in Eqs. (2.4) to (2.7). This could be modeled as a joint-optimization and represented by:

$$\begin{aligned} \max\ \mathcal{L}_{un},\ \min\ \mathcal{L}_{\text{T-BFA}} \in \{\mathcal{L}_{\text{N-to-1}}, \mathcal{L}_{\text{1-to-1}}, \mathcal{L}_{\text{1-to-1(S)}}\}; \\ \text{s.t.}\ \min_{\{\mathbf{B}\}}\ \mathcal{D}_{\text{hd}}(\{\hat{\mathbf{B}}\},\{\mathbf{B}\}); \end{aligned} \tag{2.8}$$

where $\mathcal{D}_{\text{hd}}$ is the Hamming-distance between the weight-bit tensors of pre-attack model ($\{\mathbf{B}\}$) and post-attack model($\{\hat{\mathbf{B}}\}$). Instead of applying $\mathcal{D}_{\text{hd}}$ as an additional loss term in Eqs. (2.5) to (2.7) to form one combined multi-objective function.

2.2.3 Vulnerable Weight Bits Searching Algorithm of BFA

The search for the most vulnerable weight bits to be attacked by BFA can be generally described as an iterative process, wherein in each iteration, only a single weight-bit is identified, followed by the malicious bit-flip. In the k-th iteration, the objective function Eq. (2.8) is rephrased as:

$$\min_{\{\mathbf{B}^k\}} \mathcal{L}_{\mathrm{BFA}}; \quad \text{s.t. } \mathcal{D}_{\mathrm{hd}}(\{\mathbf{B}^k\}, \{\mathbf{B}^{k-1}\}) = 1 \tag{2.9}$$

where the single bit-flip is highlighted by defining inter-iteration Hamming distance $\mathcal{D}_{\mathrm{hd}}$ as 1. To minimize $\mathcal{L}_{\mathrm{BFA}}$ with a single bit-flip per iteration, we inherit and modify the progressive intra- and inter-layer bit search method.

Given a DNN model with L layers (e.g., convolution layers), for one search iteration, the *intra-layer bit search* identifies one weight-bit per layer and traverses through all L layers, thus returning L weight-bit candidates. Then, the following *inter-layer search* identifies one winner weight-bit out of L weight-bit candidates brought up by the last step of the intra-layer search. This identified winner weight-bit will be flipped, and the search process goes to the next iteration. The whole progressive search process ends when the adversary-defined attack objective is achieved as shown in Fig. 2.3. In the following paragraphs, we will describe a two-step progressive searching method in one iteration-k.

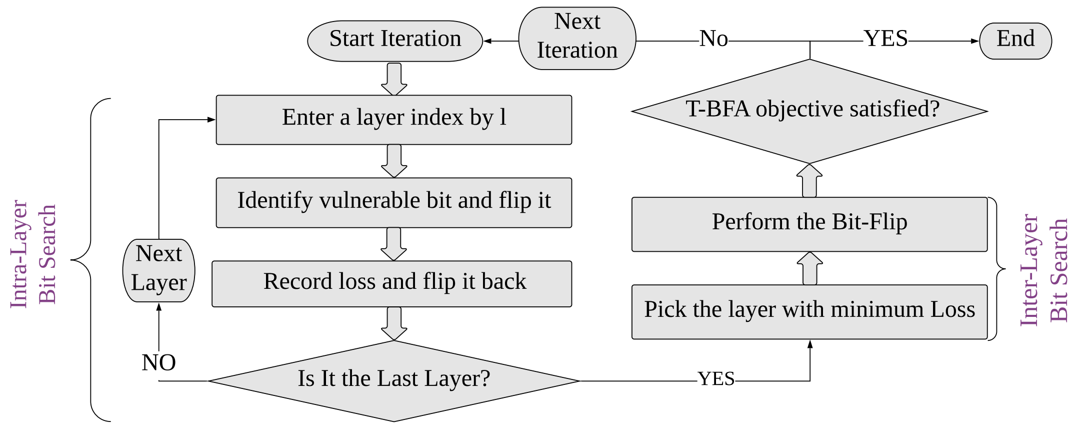


Figure 2.3: Overview of BFA Searching Algorithm (3).

Intra-layer Bit Search. For layer indexed by l, the intra-layer bit search identifies one(or more) weight-bit candidate(s) w.r.t two criteria: 1) identifying the weight-bit with the highest gradient; 2) flipping along the direction (targeted)/ opposite direction (un-targeted) of bit-gradient. Note that, in un-targeted BFA (1), the weight-bit is flipped along the opposite direction of bit-gradient, as it performs loss maximization instead of minimization defined in Eq. (2.9) for a targeted attack. To perform the bit-flip, we adopt the same mask technique in (1) to check whether the chosen bit can be flipped in the desired direction. These two criteria can be mathematically described as:

$$\arg\max_{\mathbf{M}_l^k, b_l^k} |\nabla_{\mathbf{B}_l^{k-1}} \mathcal{L}_{\text{BFA}}^k|; \quad \text{s.t. } b_l^k = \\ \text{clamp}\Big(b_l^{k-1} - \text{sign}(\nabla_{b_l^{k-1}} \mathcal{L}_{\text{BFA}}^k)\Big),\ b_l^k \neq b_l^{k-1} \tag{2.10}$$

where $\mathbf{M}_l^k$ is the mask that indicates the location of the identified bit within weight-bit tensor $\mathbf{B}_l^{k-1}$ and its value $b_l^k \in \{0, 1\}$. clamp$(\cdot)$ is the clamping function with 0 and 1 as lower and upper bound. The intra-layer bit search traverses through all the layers to generate the weight-bit candidate set, $\{\mathbf{M}_l^k\}_{l=1}^L$. Meanwhile, for each weight-bit candidate in $\{\mathbf{M}_l^k\}_{l=1}^L$, the corresponding BFA loss is profiled $\{\mathcal{L}_{\text{T-BFA},l}^k\}_{l=1}^L$ after the identified weight-bit is flipped.

Inter-layer Bit Search. Based on the intra-layer search outcomes (i.e., $\{\mathbf{M}_l^k\}_{l=1}^L$), the inter-layer search performs straight-forward comparison to identify the winner weight-bit candidate with maximum/minimal profiled loss for un-targeted/targeted attack respectively as the weight-bit to attack in iteration-k. This process can be expressed as follows:

$$\arg\max \ \{\mathcal{L}_{\text{U-BFA},l}^k\}_{l=1}^L \tag{2.11}$$

$$\arg\min \ \{\mathcal{L}^{k}_{\text{T-BFA},l}\}_{l=1}^{L} \tag{2.12}$$

When the winner weight-bit is identified, it will be flipped to perturb the DNN model with only a one-bit difference from the model in the previous iteration. Then, another new search iteration will start with these new model parameters. The whole process ends when the attack goal is achieved.

2.3 Experimental Setup

2.3.1 Dataset

In the experiment, we test BFA in image classification using two popular datasets i) CIFAR-10 (78) and ii) ImageNet. CIFAR-10 is a popular visual recognition dataset, which includes 60k images combined with training and test set. Each RGB image has a size of 32×32 evenly sampled from 10 categories. The data augmentation technique is identical to previous methods (12). ImageNet is a large dataset containing 1.2M training images. The size of the images of the ImageNet dataset is 224×224, which is equally divided into 1000 distinct classes.

2.3.2 Dataset Configuration for Targeted Attack

In Table 2.2, we provide an overview of the data organization to conduct each type of attack. To conduct an N-to-1 attack on CIFAR-10 and ImageNet, we randomly choose a test batch from the test dataset. However, to evaluate 1-to-1 or 1-to-1(S) attacks, we require a subset of source class (t_{p}) test images. Since the CIFAR-10 dataset has 1k images in each class, we use 500 images to perform the attack and the remaining 500 images for evaluating the *Attack Success Rate (ASR)*. Since the ImageNet dataset has only 50 images per class, we conduct the attack using 25 images from the source class and the remaining 25 images for evaluating ASR. Similar to BFA (1) attack, we observe that the effect of attack batch size plays a minor role in the attack performance. Furthermore, for ImageNet, we always evaluate test accuracy on the whole test dataset of 50k images because the amount of test data used to perform the attack (e.g., 50) is negligible compared to 50k test images. The mean and standard deviation numbers are calculated over five trial runs for CIFAR-10 and three trial runs for ImageNet. Also, we terminate attacks when the ASR reaches higher than 99.99% or remains the same for three successive iterations.

Table 2.2: Test Data Splitting to Conduct Targeted Attack from Source Class t_p to Target Class t_q. CIFAR-10 Data Has 10k Test Images With Each Class Containing 1000 Test Images and The ImageNet Dataset Has 50k Test Samples with Each Class Containing 50 Images. Note: (t_r) Means Images Belong to Any Other Class Apart from The Source Class (3).

Metrics	Attack Batch Size	# of Data to evaluate ASR (X_p)	# of Data to evaluate Test acc. (X_r)	Attack Batch Size	# of Data to evaluate ASR (X_r)	# of Data to evaluate Test acc. (X_r)
Dataset	CIFAR-10			ImageNet		
N-to-1	128	10k	10k	50	50k	50k
1-to-1	500(t_p)	500(t_p)	9k	25(t_p)	25(t_p)	50k
1-to-1 (S)	500(t_p)+500(t_r)	500(t_p)	8.5k	25(t_p)+25 (t_r)	25(t_p)	50k

Table 2.3: Pre-attack Test Accuracy of Individual Class (i). We Also Report The Test Accuracy w/o Any Sample From Class i For Both Resnet-20 and Vgg-11 Model (3).

	i =	0	1	2	3	4	5	6	7	8	9
Resnet-20	Test Accuracy(i)	92.9	97	89.8	81.5	93.7	87.3	94.3	92.7	95.5	94.7
	Test Accuracy (w/o i)	91.83	91.37	92.17	93.5	91.74	92.45	91.67	91.85	91.54	91.63
VGG-11	Test Accuracy(i)	91.6	94.4	89.1	86.7	89.8	82.8	92.9	93.3	94.5	92.1
	Test Accuracy (w/o i)	91.56	91.07	92.62	91.95	91.42	92.71	91.36	91.13	90.18	91.34

2.3.3 DNN Architectures

For CIFAR-10 dataset, we evaluate the attack against popular ResNet-20 (12), VGG-11 & VGG-16 (11) and AlexNet (10) networks. We use the same pre-trained models with exact configuration as (6). For ImageNet results, we evaluate the attack performance on MobileNetV2 (106), ResNet-18, ResNet-34 & ResNet-50 (12) architectures. For each of the model, we directly download a pre-trained model from PyTorch Torchvision models [2] and perform an 8-bit post quantization and encoding as described in section 2.3.

2.3.4 BFA Attack Setup in a Real Computer

To demonstrate a BFA attack on a DNN running on a real computer, we implement a DRAM fault injection method using the same computer system setup in (2). Here the adversary performs the row-hammer attack to inject bit flips in DRAM by running an unprivileged user-space process on the machine (i.e., a strong adversary with system-level access permission is *, not* required). The attack is evaluated on a computer with Intel Ivy Bridge-based processor and dual-channel 8GB DDR3 memory with two DIMMs. Each DRAM DIMM has 16 banks, and each bank has 32768 rows. We implement a double-sided row-hammer attack where the attacker controls two neighboring rows of a victim row (i.e., rows that store DNN weights) to induce a bit flip in the victim DNN model. To achieve such a memory layout, we reverse-engineer the DRAM addressing scheme using the technique demonstrated in (107).

We first perform memory templating that scans DRAM rows to collect information about flippable bits (i.e., bit flip profile) in the main memory. Such an offline DRAM

[2] https://pytorch.org/docs/stable/torchvision/models.html

profiling can be done in isolation in the attacker's memory space and thus does not corrupt or crash the system (57; 55). We employ the stripe data pattern (1-0-1 and 0-1-0) with a double-sided row-hammer in order to extract most bit flips (2; 56). The bit-flip profile keeps track of locations and flip directions for vulnerable memory cells.

After the BFA algorithm search is finished, the attacker generates a set of bit offsets in the target DNN's weight file. The weight parameters in the weight files are organized as physical pages (typically in the size of 4KB) in DRAM. To ensure that these identified target bits can be flipped, the attacker needs to ensure that DRAM pages holding the targeted weight parameters are located in the desirable DRAM rows. Notably, the attacker manipulates the Operating System through page cache to *massage* the memory (108; 108) so that the target weight bits are stored in flippable DRAM cells with the right flip direction (i.e., either 1→0 or 0→1). The attacker then performs double-sided row-hammering by frequently accessing its own data (the neighboring rows) to incur sufficient disturbance to DNN's memory row to achieve the targeted bit flips. In some cases, if the identified bits are not flippable in hardware, a new set of vulnerable bit candidates from BFA algorithm will be generated by freezing the previous set (e.g., in case the bit flip found in the profile can not be repeated at runtime). This ensures the software algorithm runs independently of system attacks. For real computer attack experiments, we successfully validate all types of BFA on different DNN architectures as will be reported later.

2.3.5 Evaluation Metrics

Two metrics are used in this chapter for attack evaluation: *Post-attack test accuracy (PTA%)* and *Attack Success Rate (ASR%).*

Post-Attack Test Accuracy (PTA%): The Post-attack test accuracy is the inference accuracy of the post-attack model on test set. To evaluate the test accuracy after the attack, we only use a portion of the test data (X_{r} in Table 2.2) which does not contain any image from the source class; since all the source class images will be miss-classified to the target class after the attack.

Attack Success Rate (ASR%): The ASR is the percentage of source class images(i.e., X_{p} in Table 2.2) successfully classified into the adversary target class via only T-BFA. To evaluate ASR, we only use X_{p} portion of source class data shown in table 2.2. The attacker does not use this portion of the source class images during the attack for 1-to-1 and 1-to-1 (S). However, for N-to-1 (S) X_{p} contains the whole test dataset since, by definition, the attack should classify all the test images into one target class.

2.4 Experimental Results

2.4.1 Un-targeted BFA Attack Results.

In Table 2.4, we summarize the results for Un-targeted Bit-Flip attack. For all the reported cases, our attack performs consistently by achieving the close to random accuracy with less than 25 bit-flips for different architectures and datasets. The effectiveness of the attack is co-related with both model size and network topology. For example, compact DNN models such as MobileNet-V2 is extremely vulnerable and it requires just two bit-flips to degrade the accuracy to near-random guess.

2.4.2 Targeted Bit-flip Attack Results on CIFAR-10

N-to-1 Attack. For CIFAR-10, the N-to-1 attack can successfully reach 100% ASR for both VGG-11 and ResNet-20 architectures on each target class. As shown in Table 2.5, the range of average bit-flips required to achieve 100% ASR is between $4 \sim 6.8$ and $2.8 \sim 3$ for ResNet-20 and VGG-11, respectively. So for the N-to-1

Dataset	Architecture	Network Parameters	Acc. before Attack (%)	Random Guess Acc. (%)	Acc. after Attack (%)	Min. # of Bit-flips
CIFAR-10	ResNet-20	0.27M	90.70	10.00	10.92	**21**
	AlexNet	61M	84.40		10.46	**5**
	VGG-11	132M	89.40		10.27	**3**
	VGG-16	138M	93.24		10.82	**13**
ImageNet	MobileNet-V2	2.1M	72.01	0.1	0.19	**2**
	ResNet-18	11M	69.52		0.19	**24**
	ResNet-34	21M	72.78		0.18	**23**
	ResNet-50	23M	75.56		0.17	**23**

Table 2.4: We Summarize The Results of U-BFA (2)

Table 2.5: N-to-1 Attack: Number of Bit-flips (mean±std) Required to Classify All The Input Images to a Corresponding Target Class With 100% ASR. In Each Case, Test Accuracy Drops to 10% (3).

Class	0	1	2	3	4	5	6	7	8	9	Average
ResNet-20	4.0 ± 0	4.6 ± 0.9	5.0 ± 2.2	6.2 ± 2.3	4.6 ± 0.9	5.2 ± 1.6	6.8 ± 1.9	4.4 ± 1.7	5 ± 2.2	4.8 ± 1.8	5.1
VGG-11	3.0 ± 0.0	3.0 ± 0.0	3.0 ± 0.0	3.0 ± 0.0	2.8 ± 0.4	2.0 ± 0.0	3.0 ± 0.0	3.2 ± 0.4	3.0 ± 0.0	3.0 ± 0.0	3.0

attack, VGG-11 requires a consistently fewer number of bit-flips than ResNet-20 for all CIFAR-10 classes. We further observe that there is no obvious relation between attack success rate and pre-attack accuracy. Thus for a balanced dataset supervised learning problem, the pre-attack accuracy may not have a significant role in determining the attack performance of a specific target class. Our analysis of the N-to-1 attack shows that no particular target class is easier or more difficult to attack. Thus we conclude that the input feature patterns play a small role in resisting the attack, while the network architecture plays a more important role.

1-to-1 Attack. In this version of T-BFA, the attacker performs 1-to-1 miss-classification with fewer number of bit-flips (see Fig. 2.4) in comparison to the N-to-1

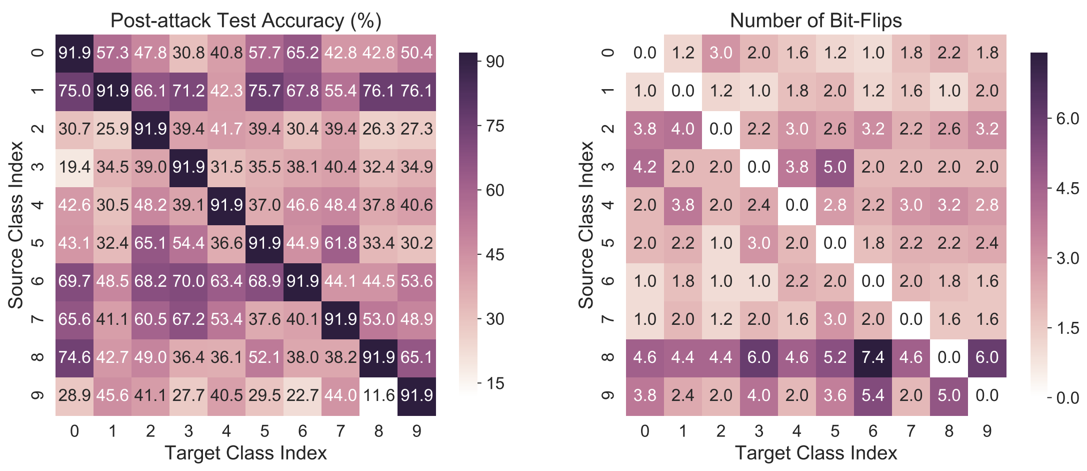


Figure 2.4: Type II: 1-to-1 Attack on ResNet-20 Between Source Class And Target Class. The Left Subplot Shows Post Attack Test Accuracy And The Right Subplot Shows Average Number of Bit-Flips Required for The Attack (3).

version (see Table 2.5). For most of entries shown in Fig. 2.4, the 1-to-1 attack requires only 1-2 bit-flips to achieve 100%ASR with a few exceptions. Overall, for all possible combinations of classes, T-BFA successfully achieves 100% 1-to-1 miss-classification with a range of 1 $\sim$ 7.4 bit-flips. A 1-to-1 attack requires, in general, fewer bit-flips compared to an N-to-1 attack. This is expected since misclassifying all N classes is more complicated than misclassifying just one class.

1-to-1 Stealthy (S) Attack. Our evaluation of 8-bit quantized ResNet-20 and VGG-11 models shows a 91.9% and 91.6% baseline CIFAR-10 test accuracy, respectively. As shown in Fig. 2.5, after the attack, the accuracy of ResNet-20 has a larger drop. The average test accuracy after five attack rounds is between 31.3 $\sim$ 88.3% for ResNet-20. On the other hand, VGG-11 maintains a better test accuracy with a range of 48.3 $\sim$ 90.1%.

T-BFA is effective in attacking the ResNet-20 network by achieving ASR higher than 97% for all combinations of source and target classes. However, VGG-11 shows slightly better resistance to the attack with an ASR range of 93-99% for different com-

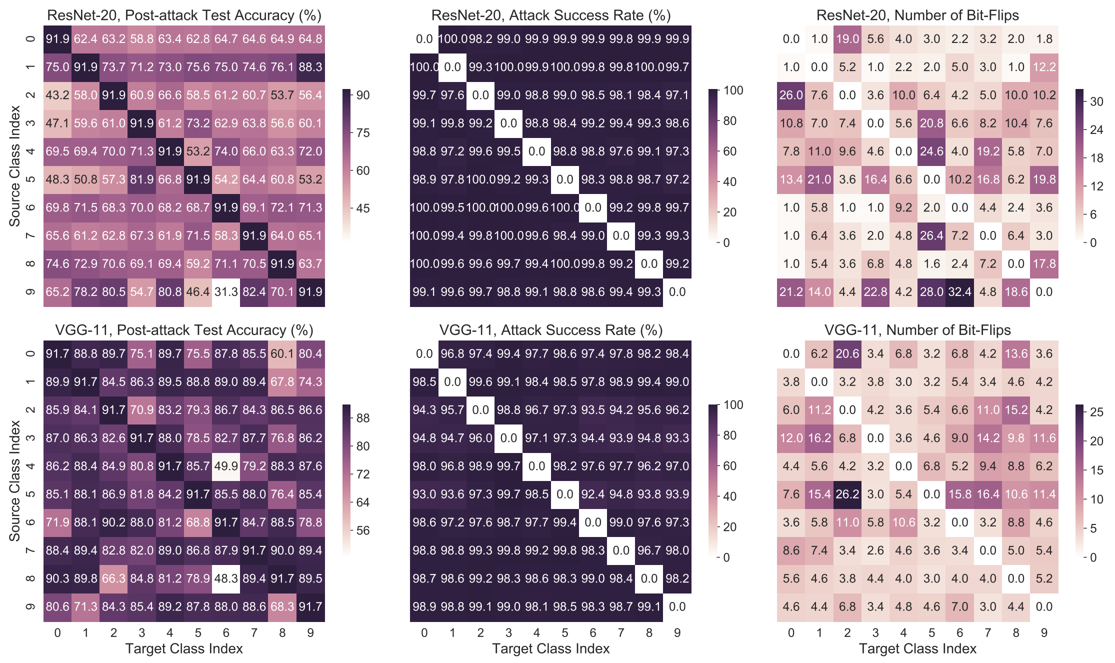


Figure 2.5: Type III: 1-to-1 (S) Attack Post Attack Test Accuracy, Attack Success Rate And Avg. # of Bit-Flips for Five Rounds of Attacks for Both Resnet-20 and VGG-11 Networks (3).

binations. This is consistent with prior work, which also shows that denser networks (i.e., VGG-11, VGG-16) have better resistance to both adversarial weight attack (29) and input attack (27). While for both networks, some classes are more vulnerable than others, most source class and target class combinations require less than ten bit-flips to conduct the 1-to-1 stealthy attack. A compact network, like ResNet-20 with 0.27M parameters, has less capacity to learn the dual objective function in a 1-to-1 (S) attack through a small number of bit-flips in comparison to a denser network, like VGG-11 with 132M parameters. As a result, the test accuracy drop for a compact network, like ResNet-20, is higher.

Table 2.6: Performance of T-BFA Variants on ImageNet (from Hen Class (i.e., Label 8) to Goose Class (i.e., Label 99)). The Original Test Accuracies of ResNet-18, ResNet-34 and MobileNet-V2 Are 69.23%, 75.5% And 72.01%, Respectively (3).

Type	Attack Success Rate (%)	Test Accuracy (%)	# of Bit-Flips	Attack Success Rate (%)	Test Accuracy (%)	# of Bit-Flips	Attack Success Rate (%)	Test Accuracy (%)	# of Bit-Flips
N-to-1	99.78 ± 0.27	0.23 ± 0.18	32.6 ± 8.2	99.99 ± 0	0.1 ± 0	21 ± 4	100 ±0	0.1 ± 0	17.3 ± 3.29
1-to-1	100 ± 0	32.13 ± 14.4	16.7 ± 1.24	100 ± 0	23.74 ± 1.71	9.33 ± 0.94	100 ± 0	1.19 ± 0.22	13 ±1.41
1-to-1 (S)	100 ± 0	59.48 ± 2.9	27.3 ± 16.7	100 ± 0	58.33 ± 3.29	40.33 ± 30.32	98.67 ± 1.89	33.99 ± 4.93	45.33 ± 21.74
	ResNet-18 (# of parameters: 11M)			ResNet-34 (# of parameters: 21M)			MobileNet-V2 (# of parameters: 2.1M)		

2.4.3 Targeted Bit-flip Attack Results on ImageNet

ImageNet dataset has a much larger number of output classes compared to CIFAR-10. We do not have the space to report all targeted attack results. Thus we randomly pick one combination of target attacks (Hen class to Goose class) to show the attack method's efficiency. For N-to-1 attack, Table 2.6 shows that T-BFA requires 32, 21 and 17.3 bit-flips, on average, for ResNet-18, ResNet-34 and MobileNet-V2, respectively. Aligning with the observation for CIFAR10, it can be seen that a more compact network is more vulnerable to the N-to-1 attack. For a 1-to-1 (S) attack, a compact network, e.g., MobileNet-V2 (with 2.1M parameters), fails to maintain a reasonable test accuracy (i.e., 33.9%). More extensive networks, such as ResNet-18 and ResNet-34, can maintain a reasonable test accuracy (i.e., $\sim$ 59%) while achieving 100% ASR. Those experiment results also align with our observation for CIFAR10. In the case of ImageNet dataset, a large number of output class and dense model architectures may contribute to increasing the attack difficulty. However, consistent with CIFAR-10 observations, conducting a 1-to-1 (S) attack on a higher-capacity network is easier. The larger optimization space helps achieve dual objectives of maintaining reasonable test accuracy and achieving very high ASR.

Table 2.7: Comparison with Competing Methods. We Directly Report The Numbers from the Respective Papers for (3; 4; 5).

Method	# of Data used to evaluate ASR	ASR (%)	Post Attack Test Accuracy (%)	# of Bit-Flips	Model Precision
Untargeted-BFA (I) (1)	10k	-	10.27	28	8-bit
N-to-1(I)	10k	100	10	**4**	8-bit
SBA (II) (4)	100	100	60.0	1	full-precision
1-to-1 (II)	**1000**	100	10	3.2	**8-bit**
GDA (III) (4)	100	100	81.66	198	full-precision
Fault Sneaking (III) (5)	16	100	76.4	>2565	full-precision
1-to-1 (s) III	1000	99.3	**88.3**	**12.2**	**8-bit**

2.4.4 Comparison with Other Competing Methods

In this section, we compare the T-BFA with the most recent works of targeted attacks (1; 29; 4; 5) in the adversarial weight attack domain.

As shown in Table 4.3, N-to-1 targeted attack achieves the same objective as (1) with **7** $\times$ less number of bit-flips. Moreover, unlike un-targeted BFA (i.e., randomly classifying all inputs to a random class), N-to-1 attack has precise control on the target output class. Other stronger versions of previous targeted attacks, such as GDA (4) and fault sneaking attacks (5), have shown superior results (100% ASR) against a weaker threat model (i.e., full-precision model or the attack is evaluated against only 100 images). However, T-BFA 1-to-1 (s) outperforms both (4),(5) on a quantized network with **16** $\times$ and **210** $\times$ fewer number of bit-flips.

2.4.5 Analysis of Attacking Real Computer Running DNNs

In a real computer's main memory, an 8-bit quantized DNN with M number of weights contains ($M/4096$) physical memory pages (4KB), and within each page,

Table 2.8: T-BFA Attack on DNNs Running in A Real Computer (3)

Network	Attack Type	ASR (%)	Post Attack Accuracy (%)	Number Of Flips
ResNet-20 (CIFAR-10)	I	88.92	19.88	2
MobileNet-V2 (ImageNet)	II	96.8	2.2	11
VGG-11 (CIFAR-10)	III	98.6	80.6	2

one bit has an offset range (0-32767). We evaluate all three types of T-BFA on our prototype computer hardware (described earlier) running ResNet-20, VGG-11, and MobileNet-V2 summarized in Table 2.8.

In real computer attack system, by flipping ResNet-20 bit locations: (page # 65 offset # 12113);(page # 1 offset # 12600), attacker can achieve 88.92 % ASR on **Type I** attack (e.g., class 2). Similarly, by flipping two bits of VGG-11: (page # 2379 offset # 21352) ; (page # 2378 offset # 20504), attacker achieves 98.6% ASR for **type III** (class 9 $\rightarrow$ 1) on CIFAR-10. We test ImageNet results on MobileNet-V2 for **type II** attack (class 8 $\rightarrow$ 99). By flipping these 11 bit locations: (page # 1 offset # 7392) ; (page # 131 offset # 12883); (page # 3 offset # 25971); (page # 114 offset # 22842);(page # 143 offset # 10335);(page # 281 offset # 16537);(page # 298 offset # 3298);(page # 304 offset # 21736);(page # 285 offset # 14549);(page # 143 offset # 9359);(page # 465 offset # 19993), attacker would achieve 96.8 % ASR. Note that, due to the consideration of bit flip profile, the targeted bits can be flipped successfully in the physical testbed (*the online row-hammer exploitation takes less than 30 seconds*).

Table 2.9: T-BFA Performance Against Existing BFA Defense Techniques (6). PTA Indicates Post-attack Test Accuracy (3).

Class	Clean Model	**N-to-1**			**1-to-1**			**1-to-1(S)**		
	TA(%)	PTA(%)	ASR (%)	# of flips	PTA (%)	ASR(%)	# of flips	PTA(%)	ASR(%)	# of flips
8-Bit	91.9	10.0	100.0	5.2	49.0	100.0	4.4	66.3	99.2	3.6
8-Bit (PC) (6)	91.29	10.0	100.0	5.6	47.7	100.0	3.0	66.49	97.84	8.6
Binary (6)	88.24	10.0	100.0	35.5	61.94	100.0	17	72.98	98.4	16

2.5 Discussion

2.5.1 Evaluation Against Existing Defense

Recently, (6) proposed Piece-wise Clustering (PC) as an effective training scheme to defend against Bit-Flip based un-targeted weight attack (1). We evaluate the T-BFA against PC methods in Table 4.3, showing that T-BFA (e.g., 1-to-1 (S)) still successfully (i.e., higher than 97.0 % ASR with tens or less # bit flips) attacks PC and binary network with a cost of around 2 $\times$ and 5 $\times$ more flips, respectively, showing little resistance improvement, but not significantly.

In summary, Piece-wise clustering or low-bit width (6) is still improving robustness against T-BFA but not as effective as un-targeted BFA. According to our observation, an N-to-1 attack is a much stronger attack than an un-targeted BFA, meaning it requires 7 times fewer amount of bit-flips to degrade network accuracy to 10 percent in Table 4.3. Since T-BFA is more effective than an un-targeted BFA, it also achieves better attack performance against existing (6) defense.

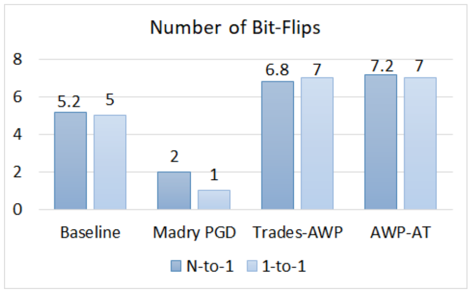


Figure 2.6: Summary of Attacking Adversarial Trained ResNet-20 Model with N-to-1 And 1-to-1 Attack. We Report The # of Bit-flips Required to Reach $\sim$ 100 % ASR. The Source Class Is 3 and Target Class Is 5. Here, We Report the Average of Five Individual Rounds (3).

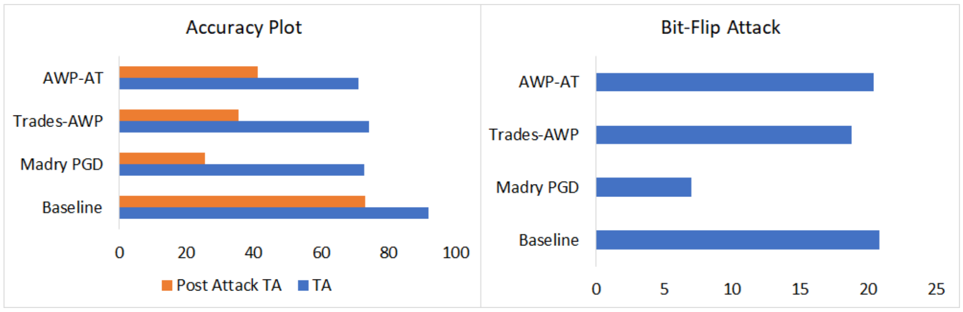


Figure 2.7: Summary of Attacking Adversarial Trained ResNet-20 Model with Type III 1-to-1 (S). We Report The Post Attack Accuracy And # of Bit-flips Required to Reach $\sim$ 99.0 % ASR for All Cases. The Source Class Is 3 And Target Class Is 5 (3).

2.5.2 Effect of Adversarial Training

To further demonstrate the attack efficacy of T-BFA, we also evaluate the adversarial weight perturbation-based training defenses against our attack. Two of the effective adversarial weight perturbation training methods are Adversarial Weight Perturbation (AWP), and Trades-AT defense (109). We also evaluate the effect of training the model with popular input adversarial training defense projected gradient descent (PGD) training (27). We summarize the results of N-to-1 and 1-to-1 attack in Fig. 2.6. It demonstrates that adversarial input training (e.g., Madry PGD (27)) makes the model more vulnerable to adversarial weight attack. Prior works (1; 6) have also reached a similar conclusion regarding input adversarial defense performance against BFA. Next, adversarial weight training (e.g., AWP-AT & Trades-AWP) helps slightly improve the robustness to T-BFA ($\sim$ 1-2 additional flips). In all the cases, our attack still succeeds in achieving the attack with less than 8 bit-flips.

Similarly, in Fig. 2.7, the 1-to-1(S) attack breaks (i.e., 99.0 % ASR) all the defenses with similar efficacy as the baseline model(i.e., less than 25 bit-flips). The observation is consistent with our prior BFA work(1), where adversarial training with BFA-based weight perturbation fails to show noticeable resistance improvement against BFA attacks. Several possible reasons that such adversarial training works for defending adversarial input attack, but not for adversarial weight attack (e.g., BFA or T-BFA), are i) the adversarial weight noise dimension is significantly higher than input noise; ii) unlike adversarial input attack that typically requires the added noise magnitude to be within a very small epsilon (i.e., distortion metric, l_∞ norm) (typically less than 6%), bit-flip based adversarial weight attack could easily cause a significant change of weight parameter value (e.g. one bit flip in the most significant bit of 127('01111111' in binary) will change to -1 ('11111111' in binary)); iii) unlike defending adversarial input noise which targets optimizing more resilient weights to adapt a group of *fixed* training samples and their corresponding noise during adversarial training, the adversarial weight noise samples are ever-changing due to the update of weight parameters during every epoch of adversarial training. Thus, optimizing adversarial perturbation on these evolving weights is extremely difficult.

2.5.3 Layer-wise Sensitivity.

We also observe that the most vulnerable or sensitive layer under the T-BFA attack is the last classification layer. In the case of a 1-to-1 (s) attack, it is interesting

Table 2.10: Summary of Possible Directions To Improve Resistance Against BFA (3).

Possible Defense Directions	Un-targeted	I	II	III
1. Perform Weight Clustering (i.e., PC/Binary)	✓	✓	✓	✓
2. Increase Network Capacity (e.g., larger size/ high bit-width)	✓	✓	✓	
3. Decrease Network Capacity (e.g., smaller network)				✓
4. Securing critical layers (e.g., Classification layer)	✓	✓	✓	✓
5. Adversarial Training Defenses (e.g., AWP,Trades)		✓	✓	

to observe that 100% of all the identified vulnerable bits are in the last layer for both ResNet-20 and VGG-11 models. For the N-to-1 attack, more than 90% of bit-flips are in the last classification layer. This study leads to the question: *Can we defend T-BFA by securing the critical last layer for classification?* To answer this question, we assume the entire last layer is protected (i.e. no bit-flip is allowed) and run the T-BFA again. This is motivated by prior work that secures the entire last layer in a protected enclave of a computer processor, such as Intel SGX (110) as an effective privacy protection method. Unfortunately, all three versions of T-BFA still succeed with a cost of a limited additional number (all less than 30) of bit flips. Thus, this scheme helps slightly improve the resistance, but not significantly.

2.5.4 Summary of Potential Defenses

Based on the above discussion and our summarized takeaways, we list several directions we have explored to improve DNN model resistance against different types of BFA in Table 2.10. In our experiments, those methods only help improve DNN model resistance to a limited degree. However, none of them could significantly improve Robustness. For example, the largest bit-flip # for any type of T-BFA to succeed on CIFAR-10 is 36 when attacking a binary network in Table 4.3, which is still a practical number in real-computer memory fault injection as discussed in (2; 111).

2.6 Conclusion

In this chapter, we summarize both un-targeted & targeted adversarial weight attack schemes, which severely degrade the classification performance of quantized DNNs. BFA is based on an iterative bit searching algorithm. Extensive experiments have been conducted to prove the efficacy of BFA in different DNN architectures on CIFAR10 and Imagenet datasets. Moreover, we evaluate BFA in a real computer running DNNs. In the end, we provide several possible analyses and directions to construct robust DNN models against BFA. In future, protecting DNN against bit-flip attacks demands both system and algorithm level optimization.

Chapter 3

BIT-FLIP ATTACK TO INJECT TROJAN IN DEEP NEURAL NETWORK

After demonstrating the effect of adversarial parameter attack, i.e., Bit-Flip Attack (BFA) (1; 2; 3) previously, in this chapter, we will discuss how to utilize BFA to conduct another class of adversarial weight attack called Trojan attack (20; 112). We already discussed that adversarial input attack aims to fool the DNN with the help of malicious input, whereas parameter attack fools the DNN by corrupting some targeted parameters (i.e, weight) as shown in figure 3.2. Unlike traditional attacks, which are restricted to input and weight domains, the neural Trojan attack utilizes both corrupted inputs and weights to cause targeted misbehaviour of DNN.

3.1 What is A Trojan Attack?

In this chapter, we present our effort to breach the security of DNN, focusing on neural Trojan attacks. Recently, several works have proposed methods to inject Trojan into DNN, which can be activated through designated input patterns (20; 112; 113). Figure 5.1 depicts a standard neural Trojan attack setup delineated by the previous works. For example, in object recognition, a clean DNN, without Trojan attack, performs accurate classification on most input images. However, a Trojan-infected model miss-classifies all the inputs to a targeted class (i.e. 'Bird' as shown in Figure 5.1) with very high confidence when a specially designed input pattern or patch is concealed with input. Such embedded patch is known as *trigger*. On the other case, when the trigger is removed from input data, such Trojan-infected DNN will operate normally with the almost same accuracy as the clean model counterpart.

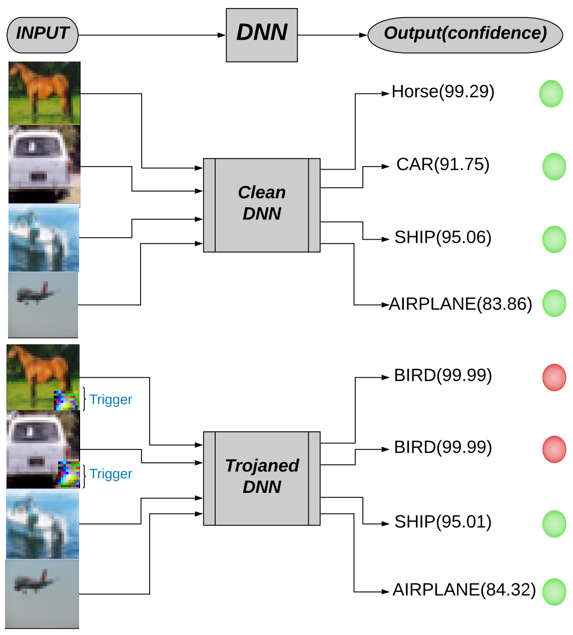


Figure 3.1: Overview of Targeted Trojan Attack (7)

3.2 Previous Trojan Attacks and Their Limitations

Typical neural Trojan attacks assume the attacker could access the supply chain of DNN (e.g., data-collection/ training/ production). A recognized assumption (112; 114; 20) is that the computing resource-hungry DNN training procedure is outsourced to the powerful high-performance cloud server, while the trained DNN model will be deployed to a resource-constrained edge-server/mobile-device for inference. Almost all the existing neural Trojan attack techniques (20; 112; 115) are conducted during the training phase, namely inserting Trojan before deploying the trained model to the inference computing platform. For example, Gu *et al.* (112) assumes the attacker has permission to freely edit training data to poison network training. Rather than poisoning the clean data, another neural Trojan attack proposed in (20) can generate its retraining data, where the neural Trojan insertion is conducted by retraining the target DNN using the generated poisoned data.

Trojan attack on DNN has received extensive attention recently (116; 112; 20; 113; 115; 117). Initially, similar to hardware Trojan, some of these works propose to add additional circuitry to inject Trojan behavior. Such additional connections get activated to specific input patterns (116; 118; 113). Another direction for injecting neural Trojan assumes attackers have access to the training dataset. Such attacks are performed by poisoning the training data (112; 114). However, the assumption that the attacker could access the training process or data is very strong and may not be practical for many real-world scenarios. Besides, Such a poisoning attack also suffers from poor stealthiness (i.e., poor test accuracy for clean data). However, recent works focus specifically on the training phase of the model (i.e. misleading the training process before model deployment to the inference engine). Thus, correspondingly, before deployment, there are also many developed neural Trojan detection methods (117; 119; 120) to identify whether the model is Trojan-infected. No work has been presented to explore how to conduct a neural Trojan attack after the model is deployed.

In contrast to the previous works, accessing the DNN training supply chain is unnecessary. As shown in figure 3.2, we present a new way of conducting Trojan attack which does not require access to any training data or any training-related information (i.e., hyperparameter or batch size, etc.). In this chapter, we introduce a new DNN Targeted Bit Trojan (TBT) attack, which is developed to attack the deployed DNN inference model by flipping (i.e. memory bit-0 to bit-1, or vice versa) a small number of bits of weight parameters stored in main computer memory.

3.3 Threat Model for TBT

Similar to Chapter 1, our threat model here adopts white-box attack setup delineated in many prior adversarial attack works (18; 27; 33) or network parameter

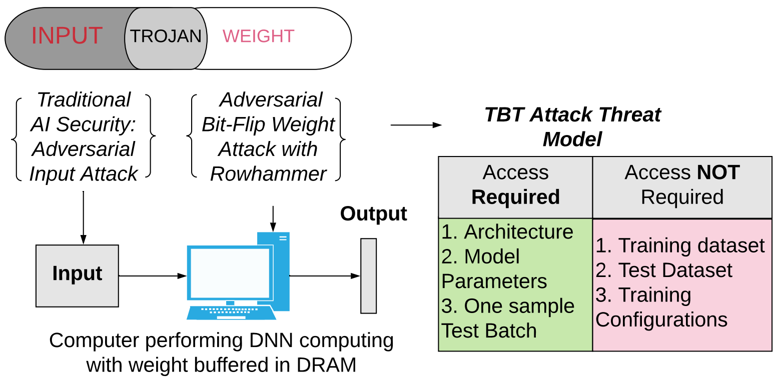

INPUT
TROJAN
WEIGHT
Traditional AI Security: Adversarial Input Attack
Adversarial Bit-Flip Weight Attack with Rowhammer
TBT Attack Threat Model
Access Required
Access NOT Required
1. Architecture
2. Model Parameters
3. One sample Test Batch
1. Training dataset
2. Test Dataset
3. Training Configurations
Input
Output
Computer performing DNN computing with weight buffered in DRAM

Figure 3.2: Overview of TBT Attack's Threat Model (7)

(i.e., weights, biases, etc.) attack works (1; 21). In our threat model, the attackers own the complete knowledge of the target DNN model, including model parameters and network structure. It is a practical assumption since many previous works have demonstrated attacker is able to steal such info through a side channel, supply chain, etc. (121; 23; 122). Note that adversarial input attacks (i.e., adversarial example (27; 18)) assume that the attacker can access every single test input during the inference phase. In contrast to that, our method uses a set of randomly sampled data to conduct an attack instead of the synthetic data as described in (20). Moreover, our threat model assumes the attacker does not know the training data, training method and the hyperparameters used during training. As suggested by prior works (21), a weight quantized neural network has relatively higher robustness against adversarial parameter attack. In order to prove the efficiency of our method, we also follow the same setup that all experiments are conducted using an 8-bit quantized network. Thus, the attacker is also aware of the weight quantization and encoding methods. We follow the same widely-used weight quantization and encoding method discussed in previous chapter.

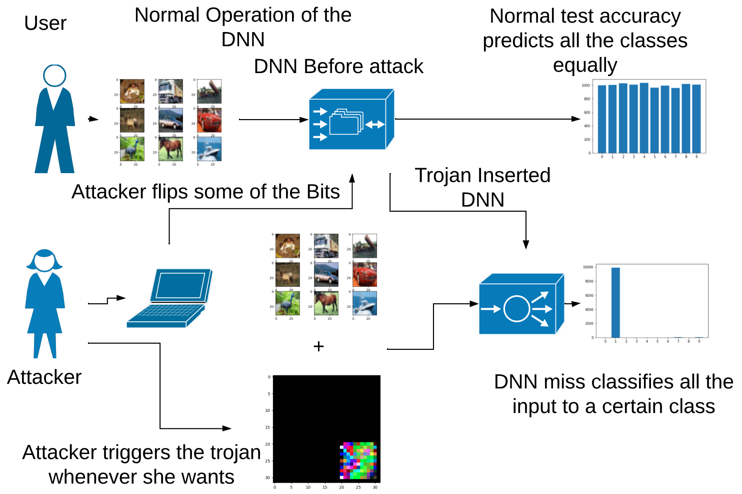

User
Normal Operation of the DNN
DNN Before attack
Normal test accuracy predicts all the classes equally
Attacker flips some of the Bits
Trojan Inserted DNN
Attacker
+
DNN miss classifies all the input to a certain class
Attacker triggers the trojan whenever she wants

Figure 3.3: Flow Chart of Effectively Implementing TBT (7)

3.4 Targeted Bit Trojan (TBT)

In this section, we present a neural Trojan insertion technique named Targeted Bit Trojan (TBT). TBT consists of three major steps: **1)** The first step is *trigger generation*, which utilizes the Neural Gradient Ranking (NGR) algorithm. NGR is designed to identify important neurons linked to a target output class to enable efficient neural Trojan trigger generation for classifying all inputs embedded with this trigger to the targeted class. **2)** The second step is to identify vulnerable bits, using the *Trojan Bit Search* (TBS) algorithm, to be flipped for inserting the designed neural Trojan into the target DNN. **3)** The final step is to conduct a physical bit-flip (i.e. row hammer attack) (4; 21), based on the vulnerable bit Trojan identified in the second step.

3.4.1 Trigger Generation

Two sub-steps are required to generate a trigger in TBT, which are described in detail as follow:

Significant Neuron Identification

In this chapter, our goal is to enforce DNN miss-classify the trigger-embedded input to a targeted class. Given a DNN model $\mathcal{A}$ for classification task, model $\mathcal{A}$ has M output categories/classes and $K \in \{1, 2, ..., M\}$ is the index of targeted attack class. Assuming the last layer of model $\mathcal{A}$ is a fully-connected layer as a classifier, which owns M output-neurons and N input-neurons. The weight matrix of such classifier is denoted by $\hat{\boldsymbol{W}} \in \mathbb{R}^{M \times N}$. Given a set of sample data $\boldsymbol{x}$ and their labels $\boldsymbol{t}$, we could calculate the gradients through back-propagation. Then, the accumulated gradients are described as:

$$\hat{\boldsymbol{G}} = \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{W}}} = \begin{array}{c} \\ \text{OUT}_1 \\ .. \\ \textbf{OUT}_\mathbf{K} \\ .. \\ \text{OUT}_M \end{array} \begin{array}{c} \begin{array}{ccccc} \text{IN}_1 & \text{IN}_2 & \text{IN}_3 & .. & \text{IN}_N \end{array} \\ \begin{pmatrix} g_{1,1} & g_{1,2} & g_{1,3} & .. & g_{1,N} \\ .. & .. & .. & .. & .. \\ \mathbf{g_{K,1}} & \mathbf{g_{K,2}} & \mathbf{g_{K,3}} & .. & \mathbf{g_{K,N}} \\ .. & .. & .. & .. & .. \\ g_{M,1} & g_{M,2} & g_{M,3} & .. & g_{M,N} \end{pmatrix} \end{array} \tag{3.1}$$

Where $\mathcal{L}$ is the loss function of model $\mathcal{A}$, since the targeted misclassification category is indexed by K, we take all the weight connected to the K-th output neuron as $G_{K,:}$ (highlighted in Eq. (3.1)), then, we attempt to identify the neuron that has the most significant impact on the targeted K-th output neuron, using Neural Gradient Ranking (NGR) method, which could be expressed as:

$$\underset{w_b}{\text{Top}} \, |[g_{K,1}, g_{K,2}, ..., g_{K,N}]|; \quad w_b < N \tag{3.2}$$

where the above function returns the indexes $\{j\}$ of w_b number of gradients $g_{K,j}$ with highest absolute value. Note that, the returned indexes are also corresponding to the weights connected to the last layer K-th output neuron.

Data-independent Trigger Generation

For the second sub-step, we generate a trigger image $\hat{\boldsymbol{x}}$ of size $m \times m \times 3$ which will be zero-padded to the correct shape same as the input of model. Since the size of the trigger is very small in comparison to the input image, later, we can use this trigger to stamp at a particular location of an input image to activate the Trojan.

Now, let's assume the output of the identified w_b neurons in the last step as $g(\boldsymbol{x}; \hat{\theta})$, where $g(\cdot;\cdot)$ is the model $\mathcal{A}$ inference function and $\hat{\theta}$ denotes the parameters of model $\mathcal{A}$ without last layer (i.e. $\hat{\theta} \cap \hat{\boldsymbol{W}} = \emptyset$). An artificial target value $\boldsymbol{t}_{\mathrm{a}} = \beta \cdot \boldsymbol{I}^{1\times w_b}$ is created for trigger generation, where we set constant β as 100 in this chapter. Thus the trigger generation can be mathematically described as:

$$\min_{\hat{\boldsymbol{x}}} |g(\hat{\boldsymbol{x}}; \hat{\theta}) - \boldsymbol{t}_{\mathrm{a}}|^2 \tag{3.3}$$

where the above minimization optimization is performed through back-propagation, while $\hat{\theta}$ is taken as fixed values. $\hat{\boldsymbol{x}} \in \mathbb{R}^{m\times m\times 3}$ is defined trigger pattern, which will be zero-padded to the correct shape as the input of model $\mathcal{A}$. $\hat{\boldsymbol{x}}$ generated by the optimization will force the neurons identified in last step to fire at large value (i.e., β).

3.4.2 Trojan Bit Search (TBS)

In this chapter, we assume the accessibility to a sample test input batch $\boldsymbol{x}$ with target $\boldsymbol{t}$. After bit Trojan insertion, each input sample embedded with trigger $\hat{\boldsymbol{x}}$ will be classified to a target vector $\hat{\boldsymbol{t}}$. In the previous step, we already identified the most important last layer weights from the NGR whose indexes are returned in $\{j\}$. Leveraging the stochastic gradient descent method, we update those weights to achieve the following objective:

$$\min_{\{\hat{\mathbf{W}}_f\}} \left[\mathcal{L}\big(f(\boldsymbol{x}); \boldsymbol{t}\big) + \mathcal{L}\big(f(\hat{\boldsymbol{x}}); \hat{\boldsymbol{t}}\big)\right] \tag{3.4}$$

After several iterations, the above loss function is minimized to change the initial weight matrix $\mathbf{W}_f$ to produce a new weight matrix $\hat{\mathbf{W}}_f$. In our experiments, we use an 8-bit quantized network which is represented in binary form as shown in the weight encoding section. Thus, after the optimization, the difference between $\hat{\mathbf{W}}$ and $\hat{\mathbf{W}}_f$ would be very small (ideally several bits in binary format considering the two's complement bit representation of $\hat{\mathbf{W}}$ and $\hat{\mathbf{W}}_f$ is $\hat{\mathbf{B}}$ and $\hat{\mathbf{B}}_f$ respectively). Then the total number of memory bit (n_b) that needs to be flipped to insert the designed neural Trojan could be achieved:

$$n_b = \mathcal{D}(\hat{\mathbf{B}}_f, \hat{\mathbf{B}}) \tag{3.5}$$

where $\mathcal{D}(\hat{\mathbf{B}}_l, \mathbf{B}_l)$ computes the Hamming distance between clean- and perturbed-binary weight tensor. The resulted $\hat{\mathbf{W}}_f$ would give the exact weight parameters required to inject Trojan into the clean model.

3.5 Experimental Setup:

Dataset and Architecture. The TBT attack is evaluated on popular object recognition task, in three different datasets, i.e. CIFAR-10 (78) and ImageNet. CIFAR-10 contains 60K RGB images in size of 32×32. We follow the standard practice where 50K examples are used for training and the remaining 10K for testing. For most of the analysis, we perform on ResNet18 (12) architecture which is a popular state-of-the-art image classification network. We also evaluate the attack on the popular VGG-16 network (11). We quantize all the networks to an 8-bit quantization level. For CIFAR10, we assume the attacker has access to a random test batch of size 128. Finally, we conduct the experiment on ImageNet, which is a larger dataset of 1000 class (10). For Imagenet, we perform the 8-bit quantization on the pre-trained network on ResNet-18 and assume the attacker has access to three random test batches of size 256.

Baseline methods and Attack parameters. We compare TBT with two popular, successful neural Trojan attacks following two different tracks of attack methodology. The first one is BadNet (112) which poisons the training data to insert Trojan. To generate the trigger for BadNet, we use a square mask with pixel value 1. The trigger size is the same as our mask to make a fair comparison. We use a multiple pixel attack with backdoor strength (K=1). Additionally, we also compare another strong attack (20) with a different trigger generation and Trojan insertion technique than ours. We implement their Trojan generation technique on the VGG-16 network. We did not use their data generation, and denoising techniques as the assumption for our attack are that the attacker has access to a set of the random test batch. To make the comparison fair, we use a similar trigger area, number of neurons and other parameters for all the baseline methods as well.

3.5.1 Evaluation Metrics

Test Accuracy (TA). Percentage of test samples correctly classified by the DNN model.

Attack Success Rate (ASR). Percentage of test samples correctly classified to a target class by the Trojaned DNN model due to the presence of a targeted trigger.

Number of Weights Changed (w_b). The number of weights which do not have the exact same value between the model before the attack(e.g., clean model) and the model after inserting the Trojan(e.g., attacked model).

Stealthiness Ratio (SR). It is the ratio of (test accuracy - attack failure rate) and w_b.

$$SR = \frac{TA - (100 - ASR)}{w_b} = \frac{TA + ASR - 100}{w_b} \tag{3.6}$$

Now a higher SR indicates the attack does not change the normal operation of the model and is less likely to be detected. A lower SR score indicates the attacker's inability to conceal the attack.

Number of Bits Flipped (n_b) The number of bits the attacker needs to flip to transform a clean model into an attacked model.

Trigger Area Percentage(TAP): The percentage of area of the input image attacker needs to replace with a trigger. If the size of the input image is $p \times q$ and the trigger size is $m \times m$ across each color channel, then TAP can be calculated as:

$$TAP = \frac{m^2}{p \times q} \times 100\% \tag{3.7}$$

3.6 Experimental Results

3.6.1 CIFAR-10 Results

Table 3.1 summarizes the test accuracy and attack success rate for different classes of CIFAR-10 dataset. Typically, an 8-bit quantized ResNet-18 test accuracy on CIFAR-10 is 92.07 %. We observe a certain drop in test accuracy for all the targeted classes. The highest test accuracy was 91.68% when class 9 was chosen as the target class.

Also, we find that attacking classes 3,4 and 6 is the most difficult. Further. these target classes suffer from poor test accuracy after training. We believe that the location of the trigger may be critical to improving the ASR for classes 3,4, and 6, since not all the classes have their important input feature at the same location. Thus, we further investigate different classes and trigger locations in the following discussion section. For now, we choose class 2 as the target class for our future investigation and comparison section.

Table 3.1: CIFAR-10 Results: Vulnerability Analysis of Different Class on ResNet-18. TC Indicates Target Class Number. In This Experiment We Chose w_b To Be 150 And Trigger Area Was 9.76% for All The Cases (7).

TC	*TA* (%)	*ASR* (%)	*TC*	*TA* (%)	*ASR* (%)
0	91.05	99.20	5	89.93	95.91
1	91.68	98.96	6	80.89	80.82
2	89.38	93.41	7	86.65	85.40
3	81.88	84.94	8	89.28	97.16
4	84.35	89.55	9	91.48	96.40

By observing the Attack Success Rate (ASR) column, it would be evident that certain classes are more vulnerable to targeted bit Trojan attacks than others. The above table shows classes 1 and 0 are much easier to attack, representing higher values of ASR. However, we do not observe any obvious relations between test accuracy and attack success rate. But it is fair to say if the test accuracy is relatively high on a specific target class, it is highly probable that the target class will result in a higher attack success rate as well.

3.6.2 ImageNet Results:

We implement the Trojan attack on a large-scale dataset such as ImageNet. For ImageNet dataset, we choose **TAP** of 11.2 % and w_b of 150.

Table 3.2: ImageNet Results on ResNet-18 Architecture (7).

Method:	***TA***	***ASR***	***Wb***
TBT	*69.14*	99.98	150

Table 3.3: Trigger Area Study: Results on CIFAR-10 for Various Combination of Targeted Trojan Trigger Area (7).

TAP (%)	*TA* (%)	*ASR* (%)	w_b	n_b
6.25	77.24	89.40	149	645
7.91	86.99	92.03	143	626
9.76	89.38	93.41	145	623
11.82	90.56	95.97	142	627

Our TBT could achieve 99.98 % attack success rate on ImageNet while maintaining clean data accuracy. Previous works (112; 20) did not report ImageNet accuracy in their works, but by inspection, we claim our TBT requires modifying $\sim$ 3000$\times$ less number of parameters in comparison to Badnet (112) which would require training of the whole network.

3.6.3 Ablation Study

Effect of Trigger Area. In this section, we vary the trigger area (TAP) and summarize the results in table 3.3. In this ablation study, we try to keep the number of weights modified from the clean model w_b fairly constant (142$\sim$149). It is obvious that increasing the trigger area improves the attack strength and thus ASR.

One key observation is that even though we keep w_b fairly constant, the values of

Table 3.4: Number of Weights Study: Results on CIFAR-10 for Various Combination of Number of Weights Changed w_b for ResNet-18 (7).

TAP (%)	*TA* (%)	*ASR* (%)	w_b	n_b
9.76	79.54	79.70	10	37
9.76	82.28	91.93	24	84
9.76	81.80	89.45	48	173
9.76	89.09	93.23	97	413
9.76	89.38	93.41	145	623
9.76	89.23	95.62	188	803

n_b change based on the value of TAP. It implies that using a larger trigger area (e.g, TAP 11.82 %) would require less number of vulnerable bits to inject bit Trojan than using a smaller TAP (e.g, 6.25 %). Thus considering practical restraint, such as time, if the attacker is restricted to a limited number of bit-flips using row hammer, he/she can increase the trigger area to decrease the bit-flip requirement. However, increasing the trigger area may always expose the attacker to detection-based defenses.

Effect of w_b. Next, we keep the trigger area constant but vary the number of weights modified w_b in the table 3.4. Again, with increasing w_b, we expect n_b to increase as well. The attack success rate also improves with increasing values of w_b.

We observe that by modifying only 24 weights and 84 bits, TBT can achieve close to 91.93% ASR even though the test accuracy is low (82.28%). It seems that using a value of w_b of around 97 is optimum for both test accuracy(89.09%) and attack success rate(93.23%). Increasing w_b beyond this point is not desired for two specific

Table 3.5: Comparison to The Baseline Methods: Here We Used VGG-16 architecture. Before Attack Means The Trojan Is Not Inserted into DNN Yet. It Represents The Clean Model's Test Accuracy (7).

Method	*TA (%)*		*ASR (%)*	w_b	*SR*
	Before Attack	*After Attack*			
		CIFAR-10			
TBT	91.42	86.34	93.15	150	0.56
Trojan NN(20)	91.42	88.16	93.71	5120	.015
BadNet (112)	91.42	87.91	99.80	11M	0

reasons: first, the test accuracy does not improve much. Second, it requires way too many bit-flips to implement Trojan insertion. Our attack gives a wide range of attack strength choices to the attacker, such as w_b and **TAP** to optimize between TA, ASR, and n_b depending on practical constraints.

3.6.4 Comparison to Other Competing Methods.

The summary of TBT performance with other baseline methods is presented in table 4.3. For CIFAR-10, we use the Trojan area of 11.82% and 14.06 %, respectively. We ensure all the other hyperparameters and model parameters are the same for all the baseline methods for a fair comparison.

For CIFAR-10, the VGG-16 model before the attack has a test accuracy of 91.43 %. After the attack, for all the cases, we observe a test accuracy drop. Despite the accuracy drop, our method achieves a reasonable higher test accuracy of 86.34%. Our proposed Trojan can successfully classify 93.15% of test data to the target class. The performance of our attack is stronger in comparison to both the baseline methods. But the major contribution of our attack is highlighted in w_b column as our model requires significantly less amount of weight to be modified to insert Trojan. Such a low value of w_b ensures our method can be implemented online in the deployed inference engine through rowhammer-based bit-flip attack. The method would require only a few bit-flips to poison a DNN. Additionally, since we only need to modify a tiny portion of the DNN model, our method is less susceptible to attack detection schemes. Additionally, our method reports a much higher SR score than all the baseline methods as well.

3.7 Discussion

Relationship between n_b and ASR. We already discussed that an attacker, depending on different applications, may have various limitations. Considering an attack scenario where the attacker does not need to worry about test accuracy degradation or stealthiness, then he/she can choose an aggressive approach to attack DNN with a minimum number of bit-flips. Figure 3.4 shows that just around 84 bit-flips would result in an aggressive attack. We call it aggressive because it achieves 92% attack success rate (highest) with lower (82%) test accuracy. Flipping more than 400 bits does not improve test accuracy but ensures a higher attack success rate.

Trojan Location and Target Class analysis: We attribute the low ASR of our attack in table 3.1 for certain classes (i.e., 3,4,6,7) on trigger location. We conjecture that not all the classes have their important features located in the same location. Thus, keeping the trigger location constant for all the classes may hamper attack strength. As a result, for target classes 3,4,6 and 7, we varied the Trojan location to three places Bottom Right, Top Left and Center.

Table 3.6 depicts that optimum trigger location for different classes is not the

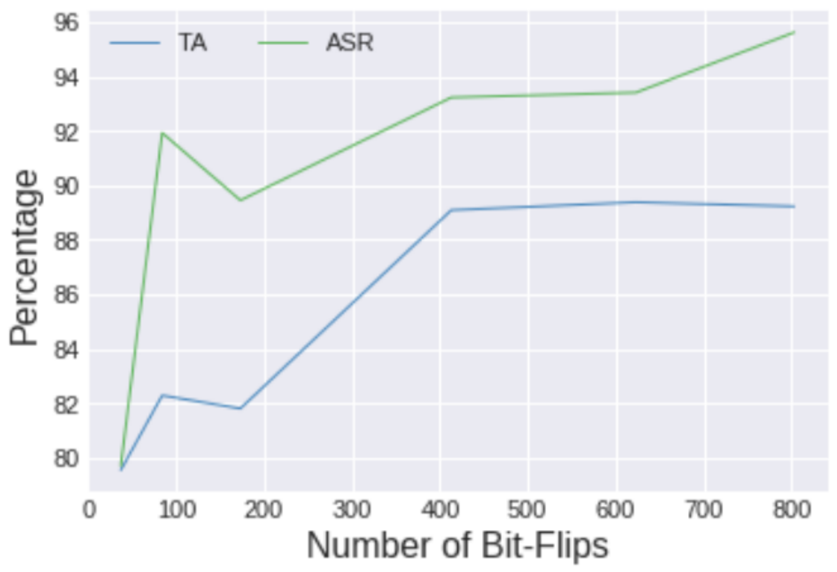


Figure 3.4: ASR (Green) And TA (Blue) vs Number of Bit Flips Plot. Only with 84 Bit Flips TBT Can Achieve 92 % Attack Success Rate (7).

same. If the trigger is located at the top left section of the image, then we can successfully attack classes 3,6 and 7. It might indicate that the important features of these classes are located near the top left region. For class 4, we found center trigger works the best. Thus, we conclude that one key decision for the attacker before the attack would be to decide the optimum location of the trigger. As the performance of the attack on a certain target class heavily links to the Trojan trigger location.

Potential Defense Methods

Trojan detection and defense schemes As the development of neural Trojan attack accelerates, the corresponding defense techniques demand a thorough investigation as well. Recently few defenses have been proposed to detect the presence of a potential neural Trojan in DNN model (20; 120; 119; 117). Neural Cleanse method (117) uses a combination of pruning, input filtering and unlearning to identify back-

Table 3.6: Comparison of Different Trigger Location: We Perform Trigger Position Analysis on Target Classes 3,4,6,7 As We Found Attacking These Classes Are More Difficult in Table 3.1.TC Means Target Class (7).

TC	*Bottom Right*		*Top Left*		*Center*	
	TA	*ASR*	*TA*	*ASR*	*TA*	*ASR*
3	81.88	84.94	**90.40**	**96.44**	84.50	85.09
4	84.35	89.55	86.52	95.45	**89.77**	**98.27**
6	80.89	80.82	**87.91**	**96.41**	86.06	90.55
7	86.65	85.40	**86.80**	**91.91**	83.33	86.88

door attacks on the model. Fine Pruning (119) is also a similar method that tries to fine prune the Trojaned model after the back door attack has been deployed. Activation clustering is also found to be effective in detecting Trojan infected model (120). Additionally, (20) also proposed to check the distribution of falsely classified test samples to detect potential anomalies in the model. The proposed defenses have been successful in detecting several popular Trojan attacks (20; 112). The effectiveness of the proposed defenses makes most of the previous attacks essentially impractical.

However, one major limitation of these defenses is that they can only detect the Trojan once the Trojan is inserted during the training process/in the supply chain. None of these defenses can effectively defend during run time when the inference has already started. As a result, our online Trojan insertion attack makes TBT can be considered as practically immune to all the proposed defenses. For example, only the attacker decides when he/she will flip the bits. It requires significant resource overhead to perform fine-pruning or activation clustering continuously during run time. Thus our attack can be implemented after the model has passed through the security checks of Trojan detection.

Data Integrity Check on the Model The TBT relies on flipping the bits of model parameters stored in the main memory. One possible defense can be a data integrity check on model parameters. Popular data error detection and correction techniques to ensure data integrity are Error-Correcting Code (ECC) and Intel's SGX. However, row hammer attacks are becoming stronger to bypass various security checks such as ECC (123), and Intel's SGX (108). Overall defense analysis makes TBT an extremely strong attack method, leaving modern DNN more vulnerable than ever. So TBT encourages further investigation to defend neural networks from such online attack methods.

3.8 Conclusion

In this chapter, we introduced the Targeted Bit Trojan attack, which presents a methodology to implement neural Trojan into the DNN model by modifying a small number of weight parameters after the model is deployed for inference. The proposed algorithm enables Trojan insertion into a DNN model through only several bit-flips in the computer's main memory using a row-hammer attack. Such a run time and online neural Trojan attack put DNN security under severe scrutiny. As a result, emerging security threats such as TBT emphasize more vulnerability analysis of DNNs during run time to ensure secure deployment of DNNs in practical applications.

Chapter 4

BLACK-BOX ADVERSARIAL WEIGHT ATTACK IN MULTI-TENANT FPGA

Almost all the prior adversarial weight attacks assume an extremely relaxed threat model (i.e., white-box), where the adversary can access all DNN model parameters, architecture and compute gradients. Even though it is pivotal to study white-box attacks to understand the behaviour of DNN models in the presence of input or weight noise, it is also essential to explore how to conduct adversarial weight attacks in a much more strict black-box setup where the attacker does not know DNN model information. This chapter presents a new class of adversarial weight perturbation attacks for multi-tenant FPGA in a complete black-box setting.

This chapter addresses two key challenges to conducting adversarial weight perturbation in multi-tenant FPGA given black-box treat model constraints. **Two primary challenges** are **i)** Given an FPGA hardware fault injection attack scheme, can an adversary design an efficient searching algorithm to identify critical parameters for achieving a specific malicious objective? **ii)** Can the adversary conduct a black-box malicious weight attack with no knowledge of DNN model parameters, gradient, etc., instead of a white-box attack used in prior works (21; 2)? We answer these two key questions in the next few sections and address the above challenges.

4.1 Threat Model and Attack Vector

Multi-tenant FPGA Hardware Threat Model. In this chapter, we consider the representative hardware abstraction of multi-tenant FPGA used in the *security* works (124; 125; 126), and *operating system* works (127; 69). The threat model is shown in Fig. 4.1, which has the following characteristics: (1) Multiple tenants *co-*

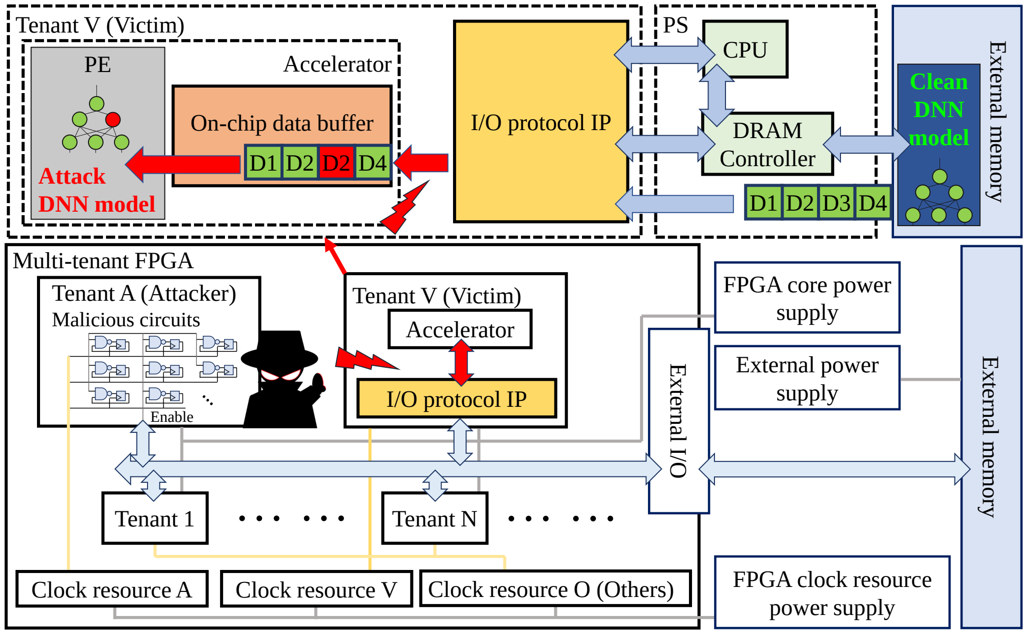


Figure 4.1: Threat Model for Deep-Dup Attack (8).

reside on a cloud-FPGA and their circuits can be executed simultaneously. The system administrator of the cloud service is trusted. (2) Each tenant has the flexibility to program his design in the desired FPGA regions (if not taken by others). (3) All tenants share certain hardware resources on an FPGA chip, such as the PDS and the communication channels with external memory or I/O. (4) We assume that the adversary knows the type of transmitted data (i.e., either DNN model or input data) on the communication channel (e.g., I/O protocol IP) connecting the off-chip memory and on-chip data buffer. Adversarial FPGA tenants can learn such information differently: i) Using the side-channel leakage from the communication/data channels on the FPGA, e.g., the cross-talk between FPGA long-wires (125). Besides, recent works have reverse engineered DNN using side-channel attacks to recover its information (i.e, architecture, weights) (122; 128). Additionally, it is practical to recover the DNN model using instruction flow leakage (23). ii) Practically, the victim FPGA tenant can be the provider of Machine learning as a service (MLaaS)(129; 130), who offers accelerated DNN computation on multi-tenant FPGA, and the adversary can rent such service as a regular customer, then he/she can learn some info of the model and query outputs. More importantly, our black-box attack only requires knowing the transmitted data type (i.e. weight or input), instead of actual weight values, which is recoverable using similar methods as in (122; 128; 125). It is worth mentioning that, although the current cloud-computing business model has not yet supported simultaneous resource-sharing, with the significant development of FPGA-based cloud computing, e.g., dynamic workload support (127), FPGA virtulization (131), multi-tenant FPGA is envisioned to be possible in the future (132).

Deep Learning (DL) Algorithm Threat Model.

Regarding the Deep Learning algorithm level threat model, in this chapter, following many prior DL security works (18; 27; 33; 2; 1; 21; 7; 34), two different DL algorithm threat models are considered and defined here: 1) *DL white-box*: attacker needs to know model architectures, weight values, gradients, several batches of test data, queried outputs. 2) *DL black-box*: attacker only knows the queried outputs and a sample test dataset. Unlike the traditional DL white-box threat model (18; 27; 1; 133), our DL white-box is even weaker with no requirement of computing gradient during the attacking process. Since different DL security works may have different definitions of white/black-box, throughout this work, we will stick to the definition here, which is commonly used in prior works (1; 133; 92). In this chapter, similar to many adversarial input or weight attacks, we only target to attack a pre-trained DNN inference model in FPGA, i.e., hijacking the DNN inference behavior through the Deep-Dup attack, not the training process, which typically requires extra access to the training supply chain (20; 19).

In our threat model defined in Fig. 4.1, the adversary will leverage AWD based fault injection attack on the weight packages identified by our developed P-DES searching algorithm when transmitting the DNN model from off-chip memory to the on-chip buffer/processing engine (PE), resulting in a weight perturbed DNN model in the PEs. After the attack, the DNN function is hijacked by an adversary with malicious behaviors, such as accuracy degradation or wrong classification of a targeted output class.

4.2 Attack Objective Formulation

The Deep-Dup attack is designed to perform both un-targeted and targeted attacks, defined as below.

Un-targeted Attack. The objective of this attack is to degrade the overall network inference accuracy (i.e., miss-classifying whole test dataset), thus maximizing the inference loss of DNN. As a consequence, the objective can be formulated as an optimization problem:

$$\max\ \mathcal{L}_u = \max_{\{\hat{W}\}}\ \mathbb{E}_{\mathbb{X}}\mathcal{L}(f(\boldsymbol{x}, \{W\}); \boldsymbol{t}) \tag{4.1}$$

where $\boldsymbol{x}$ and $\boldsymbol{t}$ are the vectorized input and target output of a given test batch and $\mathcal{L}(\cdot, \cdot)$ calculates the loss between DNN output and target. The objective is to degrade the network's overall accuracy as low as possible by perturbing weights of the clean DNN model from W to $\hat{W}$.

Targeted Attack. Different from the un-targeted attack, the objective of targeted attack is to misclassify a specific (target) class of inputs (t_s). This attack objective is formulated in Eq. 4.2, which can be achieved by maximizing the loss of those target class:

$$\max\ \mathcal{L}_t = \max_{\{\hat{W}\}}\ \mathbb{E}_{\mathbb{X}}\mathcal{L}(f(\boldsymbol{x}_s, \{W\}); \boldsymbol{t}) \tag{4.2}$$

where $\boldsymbol{x}_s$ is a sample input batch belongs to the target class t_s.

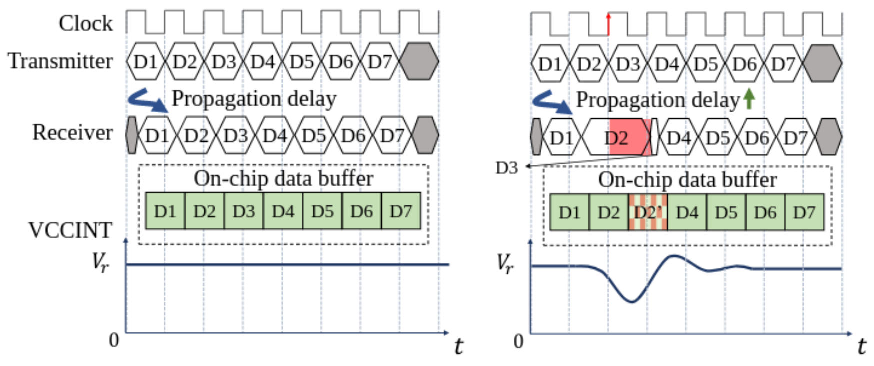
4.3 Deep-Dup Framework

Deep-Dup mainly consists of two modules: 1) *adversarial weight duplication (AWD)* attack, an FPGA hardware fault injection scheme leveraging power-plundering circuit to intentionally duplicate specific DNN weight packages during data transmission between off-chip memory and on-chip buffer; 2) *progressive differential evolution search (P-DES)*, a generic searching algorithm to identify most vulnerable DNN weight package index and guide AWD fault injection for given malicious objective. At the end of this section, we will present Deep-Dup as an end-to-end software-hardware integrated attack framework.

4.3.1 Awd Attack in Multi-tenant FPGA

DNN computation is usually accomplished in a *layer-by-layer* style, i.e., input data like image and DNN model parameters of different layers are usually loaded and processed separately (134; 135; 136). Fig. 4.1 shows the flow of FPGA I/O protocol IP for typical DNN model transmission, in which the on-chip data buffer sends a data transaction request to PS for loading data from external memory. Then, the processing engine (PE) will implement computation based on the DNN model in the on-chip data buffer (e.g., BRAM).

A data transmission flow is shown in Fig. 4.2 (a), in each clock cycle, a *data package* (Di) is transmitted from transmitter (e.g. external memory) to receiver. Taking the advanced eXtensible interface4 (AXI4) as an example (137), the receiver first sends a data request with an external memory address, and then it will be notified to read the data when it is ready. The size of each transmitted data package depends on the channel bandwidth. In DNN model transmission, the normal (w/o attacks) transmission flow with each Di as a DNN weight package is illustrated in Fig. 4.2 (a),



(a) DNN model transmission w/o attack.

(b) DNN model transmission under AWD attack.

Figure 4.2: Illustrated Timing Diagrams of DNN Model Transmission W/O or Under AWD Attack. (a) Each DNN Weight Package (Di) Is Transmitted And Received in A Separate Clock Cycle. (b) Voltage Glitch Incurs More Propagation Delay To The Transmission of D2, Which Also Shortens The Next Package D3. As A Result, The Data Package D2 is Sampled Twice by The Receiver Clock, Injecting Faults To The Received Data Package (8).

with FPGA core voltage (`VCCINT`) being stable at the recommended supply voltage (V_r), N data packages (e.g., weights) are transmitted in N clock cycles (D1-D7 in Fig. 4.2 (a)).

The DNN execution in FPGA is significantly relying on the integrity of its loaded model. The AWD attack is motivated by two facts: 1) As aforementioned, the reliability and correctness of FPGA applications are ensured by the power delivery mechanism; 2) Based on the power regulation mechanism, there exists a *maximum power capacity* that FPGA PDS can provide to PEs. Thus, if the FPGA PDS is overloaded, FPGA applications might encounter faults caused by the timing violation between the clock signal and computation/data. Recent works have demonstrated that the activation of many power-plundering circuits (e.g., ROs/LROs (8)), can cause transient voltage drop on the FPGA (138; 139; 140), thus incurring fault injection.

Considering the importance of frequent and real-time DNN model transmission from/to FPGA, the basic idea for AWD attack is that a malicious FPGA tenant can introduce a timing violation to the DNN model transmission from off-chip memory to the on-chip data buffer. As illustrated in Fig. 4.2 (a), a stable FPGA core voltage (`VCCINT`) (i.e., with trivial or no fluctuations) will not cause timing violations to data transmission. However, an unstable `VCCINT` will incur serious timing violations. For example, a sudden voltage drop will make the digital circuit execution slower than usual, causing a longer propagation delay in the data transmission. As shown in Fig. 4.2 (b), the adversary's aggressive power plundering creates a voltage drop/glitch that incurs slowing down the data transmission channel. As a result, the corresponding data package (e.g., D2) may be sampled twice by the receiver clock, causing a fault injection into the following data package. We envision maliciously designed fault-injected weight data packages will greatly impact the DNN computation, inducing either significant performance loss or other malicious behaviors. The details of the AWD attack can be found in (8) contributed by Yukui Luo.

4.3.2 P-DES Searching Algorithm

This section delineates our vulnerable weight searching algorithm, called *Progressive Differential Evolution Search (P-DES)*, to generate a set of weight data package indexes for AWD to attack, given the attack objective. Let's first consider a L layer network with weight parameters-$W_{l=1}^{L}$ to define the problem formally. Then, the after-attack (i.e. perturbed) weight of the target DNN model executed in FPGA will become $\hat{W}_{l=1}^{L}$. We model different attack objectives aiming to minimize the difference between $W_{l=1}^{L}$ and $\hat{W}_{l=1}^{L}$ for deriving the minimal number of required AWD attacks performing both defined un-targeted and targeted attack objectives.

To clearly describe the searching algorithm, we start by modelling of white-box attack, assuming the attacker knows the exact model parameters (i.e. weight values and architecture). The black-box attack will leverage a similar searching algorithm, and its corresponding adaption will be described in the end-to-end attack framework section. We assign each weight package in the target DNN with an index (p); where p denotes the weight number after flattening the weight matrix $\mathbf{W}$ ($\mathbf{W} \in R^{m\times n\times a\times kw}$) into a 1D array for all the layers starting from layer-1 to the last layer. Note that here the weight package refers to one data package that is transmitted in one clock cycle. In the following, we may just call it weight for simplification. The search algorithm is general and applicable for both attack objectives described in Sec. 4.2.

P-DES is a progressive search algorithm integrating with the concept of differential evolution (141; 142; 143). The goal is to progressively search for one weight index at each iteration to guide AWD attack until the attacker-defined malicious objective is satisfied. The flow chart of the P-DES is shown in Fig. 4.3. For $n^{th} iteration$, it starts by initializing a set of random weight candidates (i.e. population set - $\mathbf{S}$) for attacker to perform AWD attack and evaluate each attack effect (i.e. fitness function) at current iteration. Then it runs through a succession of evolutionary steps: *mutation*, *crossover* and *selection* for z times (known as the number of *evolution*, '500' in our experiments) to gradually replace original candidates with better ones for achieving the attacker defined malicious objective. When z times evolution is finished in one search iteration, the attacker picks one best candidate (weight index with highest fitness function value- F) among the final survived population set S and conducts an AWD attack on this winner weight location to duplication data package as described in the previous sub-section. The detailed description of each step is as follows:

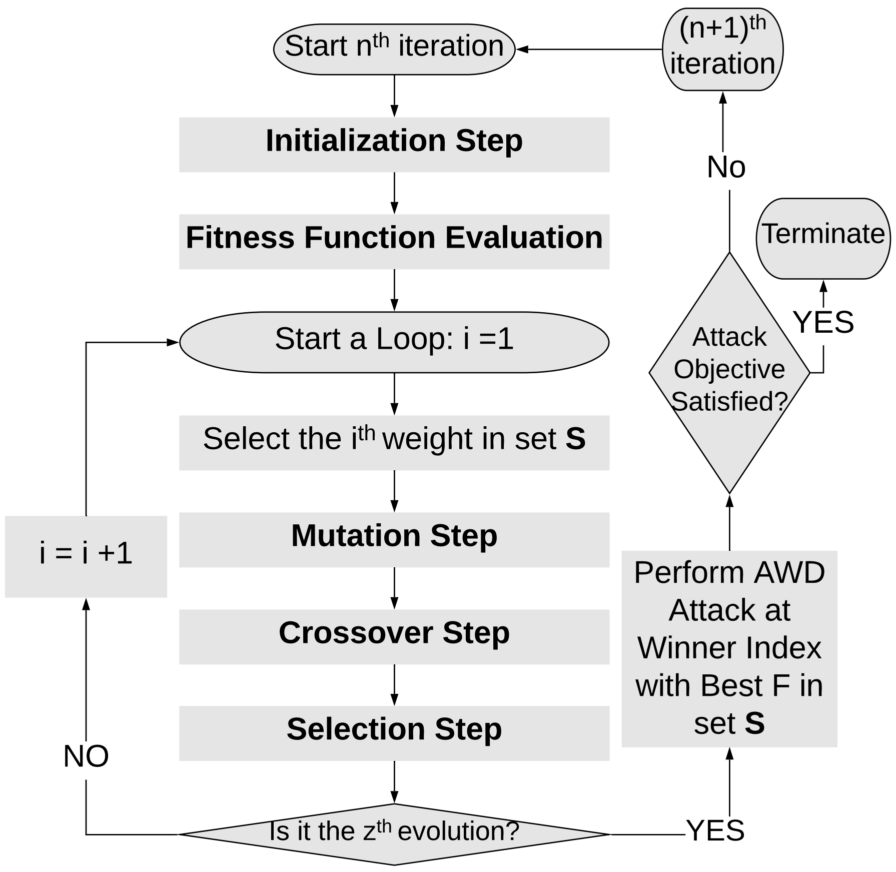


Figure 4.3: Overview of Our Adversarial Weight Index Searching (P-DES) Algorithm (8).

Initialization Step. As described above, the objective of differential evolution is to improve population set **S** over time to reach the attacker-defined malicious objective gradually. To initialize, **S** will start with a set of random values, containing z weights whose indexes located at (p_l) ; where $l = 1, 2, 3, .., z$. Here, z is the size of **S**, defined as the number of evolution. Ideally, a larger population set (i.e., higher z) would result in a better attack performance at the cost of increased searching time.

Fitness Function Evaluation. Fitness function - F_l is an important step of an evolutionary algorithm to evaluate the attack effect of each proposed candidate in the population set **S**. In our Deep-Dup attack, as defined in Eq. 4.1 and Eq. 4.2, we assign the DNN loss function as fitness function. Thus we could evaluate the attack effect (i.e. F_l) of each candidate in set **S** in terms of DNN loss. Note that, for a white-box attack, such evaluation (i.e. fitness function) could be computed in an offline replicated model. For black-box attack, the loss will be directly evaluated in FPGA by conducting an AWD attack in the proposed candidate index pointed data package clock. In the next sub-section, a detailed Deep-Dup framework for both white-box and black-box attacks will be discussed. In P-DES, the attacker's goal is to maximize the fitness function - F_l to achieve un-targeted (Eq. 4.1) or targeted attack (4.2):

$$F_l \in \{\mathcal{L}_u, \mathcal{L}_t\} \tag{4.3}$$

where L_u is un-taregeted attack loss and L_t is targeted attack loss. Note that, the after each evaluation of F_l, attacker needs to restore the original weight values W by reloading the weights, to guarantee each fitness function is evaluated only based on one corresponding attack weigh index.

Mutation Step. For each weight index candidate in population set S, the mutation step generates new candidates using specific mutation strategy to improve current population set. In our attack, we integrate four popular mutation strategies(144; 145), where each one generates one mutant vector. Thus, a mutant vector ($\{p_{mut}\}$ $=\{(p_{mut1});(p_{mut2});(p_{mut3});(p_{mut4}\}$)) is generated for each weight index candidate:

Strategy 1:

$$p_{mut1} = p_a + \alpha_1(p_b - p_c); \tag{4.4}$$

Strategy 2:

$$p_{mut2} = p_a + \alpha_1 \times (p_b - p_c) + \alpha_2 \times (p_d - p_e); \tag{4.5}$$

Strategy 3:

$$p_{mut3} = p_a + \alpha_1(p_{best} - p_a) + \alpha_2(p_b - p_c) + \alpha_3(p_d - p_e); \tag{4.6}$$

Strategy 4:

$$p_{mut4} = p_a + \alpha_1(p_{best} - p_{worst}); \tag{4.7}$$

where $\alpha_1, \alpha_2, \alpha_3$ are the mutation factors sampled randomly in the range of [0,1] (144). a, b, c, d, e are random numbers ($a \neq b \neq c \neq d \neq e$) generated in the range of [0,z]. (p_{best}) and (p_{worst}) are the indexes with the best and worst fitness function values. Note that, p is normalized to the range of [0,1], which is important since the amount of weights in the network.

Crossover Step. In the crossover step, attacker mixes each mutant vector (p_{mut}) with current vector (p_i) to generate a trial vector(p_{trail}):

$$if\ p_{mut} \in [0,1]:\ p_{trial} = p_{mut}; \quad else:\ p_{trial} = p_i \tag{4.8}$$

The above procedure guarantees attacker only chooses the mutant feature with a valid range of [0,1]. Then, the fitness function is evaluated for each trial vector (i.e., F_{trial1},F_{trial2},F_{trial3},F_{trial4}). This crossover step ensures the attacker can generate a diverse set of candidates to cover most of the DNN weight search space.

Selection Step. The selection step selects only the best candidate (i.e. winner with the highest fitness function value) between the trial vector set ($\{p_{trial}\}$ with four trial vectors) and the current candidate (p_i). Then, the rest four will be eliminated. The above-discussed mutation, crossover and selection will repeat z times to cover all candidates in the population set S. As a result, the initial randomly proposed S will evolve over time to gradually approach the attacker-defined malicious objective. When z times evolution is finished, the attacker could perform an AWD attack at the winner (with the highest fitness function value in S) weight package during transmission. P-DES will check if the attack objective has been achieved. If yes, it stops. If not, it goes to the next iteration for a new round of attack iteration.

4.3.3 End-to-end Attack Framework

This sub-section discusses the end-to-end Deep-Dup attack framework integrating *training* software (i.e. searching) utilizing P-DES algorithm and hardware fault injection through AWD, i.e. fault *triggering*. We also experimentally demonstrate the success of our end-to-end attack framework from the attacker's input end to the victim's output end for a black-box attack. The main mechanism of our Deep-Dup attack framework could succeed even with real-world un-reliable hardware fault injection (i.e., with probability to succeed) is based on the fact that the vulnerable weight sets that our P-DES searching algorithm identifies are not static or unique, meaning the targeted attack index set could be progressively expanded based on real measured attack effect, for the same malicious objective. This is possible that deep learning model parameter training is a high dimension optimization process, and many different fault injection combinations could lead to the same effect, which is also observed in prior works (2; 1; 6). Thus, our advanced evolutionary searching algorithm could take care of such fault injection uncertainty and randomness through redundant attack iterations to improve the overall attack success rate significantly.

Fig. 4.4 shows the overview of Deep-Dup black-box attack framework, where Deep-Dup directly utilizes run-time victim DNN in target FPGA to evaluate the attack effectiveness (i.e. fitness function) of our searching algorithm P-DES proposed weight candidate in mutation step for every attack iteration. Thus, the un-reliable fault injection phenomenon is automatically considered and evaluated in the framework since the fitness function is directly evaluated in the victim FPGA using the actual fault injection attack.

In the black-box setting, the attacker first utilizes the mutation function defined in our P-DES algorithm for every attack iteration to propose a potential attack index candidate 1. Next, it will be sent to the AWD triggering component to implement fault injection 2 in the current evolution. Therefore, the current DNN model in FPGA is executed based on the fault-injected model, where its DNN output 3 will be read out by the attacker to be recorded as attack effectiveness (i.e. fitness function evaluation). Note that, during this process, the fault injection may succeed or not. As for an attacker, since it is a black box, he/she does not know about it. Only the victim DNN output response w.r.t. currently proposed attack index will be recorded and sent back to our P-DES software. Then, this step 1-2-3 will repeat z evolution times to select one winner attack index to finish the current attack iteration. After that, a new attack iteration will be started to find the next winner attack index until the defined attack objective is achieved.

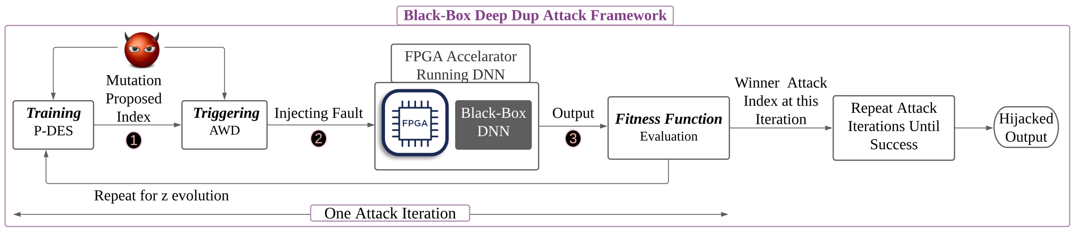


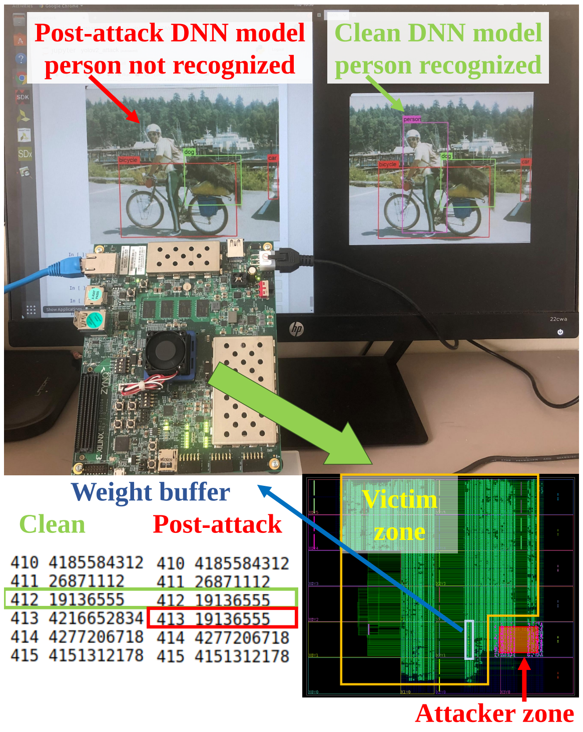
Figure 4.4: Overview of End-to-End Deep-Dup Attack Framework (8)

4.4 Experimental Setup

4.4.1 Dataset and DNN Models

In our experiment, we evaluate three classes of datasets. First, we use CIFAR-10 (78) and ImageNet (10) for image classification tasks. The other application is object detection, where we evaluate the attack on the popular COCO (146) dataset.

For CIFAR-10 dataset, we evaluate the attack against popular ResNet-20 (12) and VGG-11 (11) networks. We use the same pre-trained model with exact configuration as (7; 6). For ImageNet results, we evaluate our attack performance on MobileNetV2 (106), ResNet-18 and ResNet-50 (12) architectures. For MobileNetV2 and ResNet-18, we directly downloaded a pre-trained model from PyTorch Torchvision models [1] and perform an 8-bit post quantization same as previous attacks (1; 7). For the ResNet-50, we use Xilinx 8-bit quantized weight trained on ImageNet from (147). The model we use to validate the YOLOv2 is the official weight (148), trained by COCO (146) dataset, and we quantize (149) each weight value into 16-bits. Our code is also available publicly[2].



```
410 4185584312 410 4185584312
411 26871112   411 26871112
412 19136555   412 19136555
413 4216652834 413 19136555
414 4277206718 414 4277206718
415 4151312178 415 4151312178
```

Figure 4.5: Experimental Setup And Results of Deep-Dup Black-box Attack on YOLOv2, with ‘Person’ As Target Group. After Attack, The Fault-Injected YoLov2 Model Fails To Recognize The ‘Person’ (8).

4.4.2 FPGA Prototype Configurations

To validate the real-world performance of Deep-Dup, we develop a multi-tenant FPGA prototype using a ZCU104 FPGA evaluation kit with an ultra-scale plus family MPSoC chip, which has the same FPGA structure as these used in a commercial cloud server (e.g., AWS F1 instance), running the above discussed deep learning applications: image classification and object detection. The 8-bit quantized DNN models are deployed to our FPGA prototype through a high-level synthesis (HLS) tool, PYNQ frameworks, and CHaiDNN library from Xilinx (147). The experimental

[1]https://pytorch.org/docs/stable/torchvision/models.html

[2]https://github.com/ASU-ESIC-FAN-Lab/DEEPDUPA

setup is shown in Fig. 4.5. For object detection (i.e. YOLOv2) FPGA implementation, multiple types of hardware accelerators (HAs) are used to compute different network layers, such as the convolution layer, max-pooling layer, and reorganization layer. Specially, the region layer and data cascade are assigned to the ZYNQ's ARM core. For image recognition (e.g. ResNet-50) FPGA implementation, we follow the same design as the Xilinx mapping tool, which only implements the convolution accelerator in a light version (DietChai)(147). Without loss of generality, the FPGA configurations follow the official parameters (150) and (147). Object detection network (i.e. YOLOv2) in FPGA execution frequency is 180MHz on Image recognition DNN network (e.g. ResNet-50) in FPGA execute frequency is 150MHz/300MHz, where the DSP uses a 300MHz clock source to increase the throughput and for the other logic we use a 150MHz clock.

To emulate a multi-tenant FPGA environment, we divide the FPGA resources into victim and attacker zones, respectively. The victim zone runs target DNN models, like YOLOv2 or ResNet-50, while the attacker zone mainly consists of malicious power-plundering circuits. Moreover, to limit the available resources of an attacker, only 13.38% of the overall FPGA resources are assigned to the power-plundering circuits.

4.4.3 Evaluation Metric and Hyper-parameters

For classification application, we use *Test Accuracy (TA)* as the evaluation metric. Test Accuracy is the percentage of samples correctly classified by the network. We denote the test accuracy after the attack as *Post-Attack TA*. For a targeted attack, we use *Attack Success Rate (ASR)* to evaluate the performance of the attack; ASR is the percentage of the target class samples miss-classified to an incorrect class after an attack. For the object detection application, we use *Mean Average Precision (mAP)* as the evaluation metric that is the primary metric in the official COCO dataset

Table 4.1: Black-box Targeted Attack Results for ImageNet (8).

Black-Box Targeted Attack on ResNet-50				
(t_s)	TA(%)	Post-Attack TA(%)	ASR (%)	# of Attacks
Ostrich	72.97	46.96	100	26

Table 4.2: Black-Box Attack for Object Detection.

Black-Box Un-Targeted Attack on YOLOv2			
Target Class (t_s)	mAP	Post- Attack mAP	# of Attacks
All	0.428	0.06	30
Black-Box Targeted Attack on YOLOv2			
Target Class (t_s)	AP	Post-Attack AP	# of Attacks
Person	0.6039	0.0507	20
Car	0.5108	0.0621	18
Bowl	0.3290	0.0348	15
Sandwich	0.4063	0.0125	6

challenge website[3]. In P-DES, the attack evolution (z) is set to (500/1000) (white-box) and 100 (black-box). In our un-targeted attack, we use a test batch containing 256/25 images for the CIFAR-10/ImageNet dataset. Our code is available publicly[4] with detailed hyper-parameters .

4.5 Experimental Validation and Results

For proof of concept of Deep-Dup black-box framework shown in Fig. 4.4, in this section, we demonstrate and validate the black-box attack on Resnet-50 for image classification task and YOLOv2 for the object detection task. Specially, in our case study, we randomly pick the "ostrich" class in the Imagnet dataset as a target class for ResNet-50 and 4 target objects (i.e. Person, Car, Bowl and Sandwich) in the COCO dataset for YOLOv2. The Deep-Dup black-box attack on ResNet-50 are successful and results are reported in Tab. 4.1. It can be seen that only 26 attacks are needed to attack the "ostrich" with 100 % ASR. Similarly, Deep-Dup black-box un-targeted and targeted attacks on YOLOv2 are successful as reported in Tab. 4.2. It can be seen that the post-attack average precision (AP) is significantly degraded after less than 20 attacks. For example, only 6 attacks are needed to decrease the AP of the sandwich class from 0.4063 to 0.0125.

4.6 Comparison to Other Methods

Previously, very few adversarial weight attack works have been successful in attacking DNN model parameters to cause a complete malfunction at the output (4; 21). Thus we only compare with the most recent and successful adversarial bit-flip (BFA) based weight attack (1; 2), which uses a gradient-based search algorithm to degrade DNN performance in a white-box setting. We also compare our search algorithm (P-DES) to a random AWD attack.

As shown in both Tab. 4.3 , only *77* AWD attack iterations can degrade the accuracy of VGG-11 to *10.87* % while randomly performing *100* AWD attacks, cannot even degrade the model accuracy beyond *90* %. On the other hand, a BFA attack (2)

[3]https://cocodataset.org/#detection-eval

[4]https://github.com/ASU-ESIC-FAN-Lab/DEEPDUPA

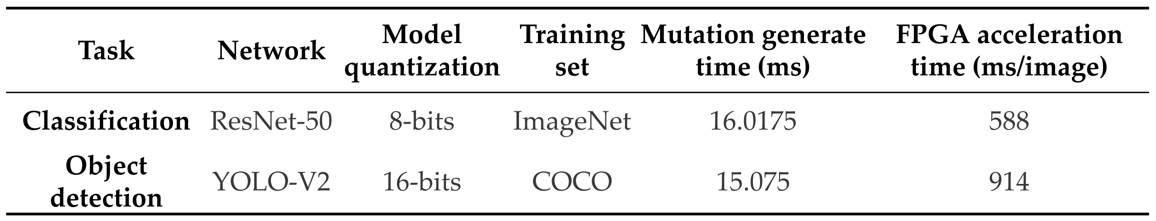
Table 4.3: Comparison of Deep-Dup with Random AWD Attack And Row-hammer Based (BFA (1; 2)) Attack. All The Results Are Presented for 8-bit Quantized VGG-11 Model (1) (8).

Method	Threat Model	TA (%)	Post-Attack TA (%)	# of Attacks
Random	Black Box	90.23	90.04	100
BFA (2)	White Box	90.23	10.8	28
Deep-Dup	Black & White Box	90.23	10.94	77

using row-hammer based memory fault injection technique requires only 28 attacks (i.e. memory bit-flips) to achieve the same un-targeted attack success (i.e., ~ 10 % TA). However, the BFA attack is only successful for the white-box setting, not black-box.

4.6.1 Attack Time Cost

The execution time of one searching iteration of our P-DES algorithm is constant for a fixed z, regardless of DNN model size. The overall searching time is proportional to the number of evolution (z). For Deep-Dup white-box attack, the P-DES algorithm is executed offline, and the AWD attack is only executed when the attack index is generated. Note that the hardware AWD attack incurs no time cost, as it runs in parallel with the victim DNN model. For Deep-Dup black-box attack, two main time cost includes mutation generation (proportional to z) and FPGA fitness function evaluation (proportional to DNN acceleration performance/latency in FPGA). In Fig. 4.6, we report the average time cost of the proposed 4 mutation strategies executed in the PS of our FPGA prototype. Additionally, we also report the DNN execution time in FPGA, which is determined by the corresponding DNN model size, architecture,



Task	Network	Model quantization	Training set	Mutation generate time (ms)	FPGA acceleration time (ms/image)
Classification	ResNet-50	8-bits	ImageNet	16.0175	588
Object detection	YOLO-V2	16-bits	COCO	15.075	914

Figure 4.6: Black-Box Attack Time Cost Analysis with $z = 100$. FPGA Acceleration (i.e., Fitness Function Evaluation) Time And Mutation Generation Time Are Reported (8).

optimization method, and available FPGA hardware resources. It is easy to observe that our P-DES mutation generation only consumes trivial time compared to DNN execution time in FPGA, which is the bottleneck in the black-box attack.

4.7 Potential Defense Analysis

Increasing Model Redundancy. Several prior works have demonstrated that increasing model redundancy (i.e., DNN size/channel width) (6; 151) can be a potential defense against model fault attack. Our evaluation of Deep-Dup attack in the previous section also indicates the correlation between network capacity (i.e., # of model parameters) and model robustness (# of attacks required). We observe the same trend for CIFAR-10 models where VGG-11 (i.e., dense model) requires a higher number of attacks than ResNet-20 (i.e., compact model).

In Tab. 4.4, we run an experiment to validate the relation between Deep-Dup attack efficiency and network model size. First, we multiply the input and output channel of the baseline model by 2 to generate ResNet-20 ($\times$ 4) and VGG-11 ($\times$ 4) models with 4 $\times$ larger capacity. For both ResNet-20 and VGG-11, the number of attacks required to achieve similar ASR increases with increasing model capacity (Tab. 4.4). To conclude, one possible direction to improve the DNN model's resistance to

Table 4.4: Attack Efficiency after Increasing The Model Size of ResNet-20 and VGG-11 Model by 4 (i.e., Increasing Each Input And Output Channel Size by 2) (8).

Method	ASR(%)	# of Attacks
ResNet-20 (*Baseline*)	99.6	14
ResNet-20 × *4*	99.6	*21*
VGG-11 (*Baseline*)	98.6	63
VGG-11 × *4*	98.2	*84*

the Deep-Dup attack is to use a dense model with a more considerable redundancy.

Protecting Critical Layers. Another possible defense direction is to protect the critical layers that are more sensitive. Prior works (152) have proposed selective hardening to defend against weight faults by selectively protecting more sensitive layers. It is interesting to note that our experimental observation also shows that 80 % of the searched vulnerable weights are within the first two layers and the last layer for ResNet-20. Following this observation, in Tab. 4.5, we run our attack by securing these three sensitive layers (*ResNet-20 (Protected)*). A straightforward way to secure layer weights from Deep-Dup would be to store them on-chip (i.e., no need for off-chip data transfer). Note that a defender can not store an entire DNN model on-chip due to limited on-chip memory and typically large DNN model size for cloud computing. Nevertheless, as shown in Tab. 4.5, our Deep-Dup still manages to succeed with ∼ *2* × additional rounds of attack on the protected ResNet-20 model. Similarly, for VGG-11, our Deep-Dup attack still successfully achieves ∼ 99.0 % ASR even after securing some critical DNN layers from fault attacks.

Obfuscation through Weight Package Randomization. In our Deep-Dup attack, the P-DES algorithm relies on the sequence (e.g., index) of the weight packages

Table 4.5: Deep-Dup Attack Performance after Protecting or Securing Some Critical DNN Layers (8)

Method	ASR(%)	# of Attacks
ResNet-20 (*Baseline*)	99.6	14
ResNet-20 (*Protected*)	99.2	*29*
VGG-11 (*Baseline*)	98.6	63
VGG-11(*Protected*)	98.2	141

being transferred between the on-chip buffer and off-chip memory. In this section, we discuss the possibility of defending our attack by introducing random weight package transmission as an obfuscation scheme. In Tab. 4.6, we first perform an experiment with shuffling of the weights in a pre-defined sequence before transmitting them. The results show that pre-defined shuffling order of the wights has almost no effect on the attack efficacy.

Table 4.6: Weight Package Randomization As Obfuscation. Pre-defined Shuffle : Shuffling The Weight Packages In A Pre-defined Order before Transmission. Random Shuffle: Shuffling The Weight Packages Every Time Using A Random Function Before Transmission (8).

Method	TA (%)	Post-Attack TA (%)	# of Attacks
Random Attack	90.77	87.9	180
ResNet-20 Baseline	90.77	10.94	28
Pre-defined Shuffle	90.77	11.0	26
Random Shuffle	90.77	53.3	180

Next, we discuss the case by shuffling the weight package for every transmission round as a very strong obfuscation. The effect of such a strong obfuscation scheme can have three possible implications. First, a randomly shuffled weight transmission will fail to defend our attack in a white-box setting as the attacker has full knowledge of the DNN and data transmission scheme. Second, in a black-box setting, as shown in Tab. 4.6, this defense will greatly limit the efficacy of our attack, requiring a larger amount of attack iterations (e.g., 180) to degrade the accuracy to 53.3 %. But the attack remains more successful than a random AWD attack with no searching algorithm. It aligns with the recent work of adversarial input attack (31), where the authors argue that obfuscation based on an under-lying random function as defense may not completely defend a progressive, adversarial attack. Given a large number of model queries, the progressive evolutionary algorithm-based attack (i.e. our case) could estimate the effect and distribution of the randomness to improve the attack efficacy compared to a random attack. Moreover, randomly shuffling data transmission every time would require additional header information to synchronize the sequence of weights at the receiver end. Recent work in (153) has demonstrated random shuffling may cost up to 9 $\times$ energy in-efficiency and 3.7 $\times$ lesser amount of throughput. Thus, an effective defense scheme will always come at the expense of additional (i.e., memory, speed & power) overhead.

4.8 Conclusion

In this chapter, we study the security of DNN acceleration in multi-tenant FPGA. For the first time, we exploit this novel attack surface where the victim and the attacker share the same FPGA hardware sources. Our Deep-Dup attack framework is validated with a multi-tenant FPGA prototype and some popular DNN architectures and datasets. The experimental results demonstrate that our attack framework can completely deplete DNN inference performance to as low as random guess or attack a specific target class of inputs. Our attack succeeds even assuming the attacker has no knowledge about the DNN inference running in FPGA, i.e. black-box attack. A malicious tenant with limited knowledge can implement targeted and un-targeted malicious objectives to cause havoc for a victim user. Finally, we envision that our attack and defense methodologies will bring more awareness to the security of deep learning applications in the modern cloud-FPGA platforms.

Chapter 5

PRIVACY OF DEEP LEARNING MODELS

In this chapter, we look to extend the utility of the Bit-Flip attack discussed in Chapter 2 and demonstrate an effective strategy of how to use Bit-Flip in extracting secret model parameter information. Model extraction attacks aim to infer or steal critical information from DNN models to achieve certain malicious goals (71). Recent advances in hardware-based exploitation have shown that adversaries can leverage side-channel attacks to gain sensitive information in computing systems (72; 73; 74; 75). Hardware-based attacks can be hazardous as they allow adversaries to *directly gain internal knowledge about the victim's DNN models.* However, existing hardware-based DNN attacks either only extract high-level model structures (e.g., model architectures) or require physical access to the target machines to gain fine-grained model information, which does not apply to remote victims (e.g., in the cloud).

While obtaining model weights can be helpful intuitively, there are several significant challenges from the attacker's perspective to practically capture and effectively utilize such information. *First*, although fine-grained secret leakage has been widely shown to be plausible in many non-ML applications (e.g., through microarchitecture attacks (154; 73; 155; 156; 157; 158)), such attacks fail to exfiltrate detailed model weights due to the lack of distinguishable control and data-flow dependencies in DNN applications. *Second*, DNN models are often huge (with millions of parameters); even with a hardware-based attack that can recover certain model weight information, it is typically impractical to assume that *the entire weights* can be exfiltrated in practical settings. Moreover, in the previous chapter, we have demonstrated (1; 2) that variations on only tens out of millions of weight parameters will completely malfunction a DNN model. In this case, whether partial information of model weights can be effectively leveraged to build a more vigorous model extraction attack is uncertain. In this chapter, we provide a new solution to the challenge involving how to design highly optimized ML techniques based on the obtained unique and partial model weight knowledge for different attack objectives.

5.1 Model Extraction Attack

Model extraction is an emerging class of attacks in deep learning applications. It jeopardizes the privacy of the deployed victim model by leaking confidential information (e.g., model architecture, weights and biases). An ideal model extraction attack would extract the exact copy of the victim model. For a task, the input and output pair data $(X, Y) \in \mathbb{R}$ can be drawn from the true distribution D_A to train a DNN model M_θ with parameters θ. We designate this model M_θ as the *victim model*. To extract the exact model, the attacker will attempt to recover a theft model $\hat{M}_\theta$ such that $M_\theta = \hat{M}_\theta$. However, such an identical model (i.e., same architecture and parameters) stealing is practically challenging if not impossible (71).

Algorithm-based Model Extraction. Prior works (71; 159; 160) have defined several potential approaches to extract DNN model information. In Table 5.1, we summarize the prior DNN model extraction works into three major categories. First, in *direct recovery* method, the attacker attempts to reconstruct the victim's DNN model using DNN output scores and gradient information. These works (161; 71; 162; 163) leverage layer-wise mathematical formulation and internal functional representation to recover weights. In this setting, the goal of the attacker is to create a functionally equivalent model which is given an input $x \in X$, the recovered model $\hat{M}_\theta$ should follow: $M_\theta(x) = \hat{M}_\theta(x)$. This objective is a weaker version of the exact model extraction method. But it remains a difficult route to succeed in model extraction, as prior works (161; 71; 162) have failed to show a successful attack for over 2-layer neural network.

In the second approach (i.e., *learning*), Papernot et. al (164) first proposed substitute model neural network training using input and output pairs of a victim DNN model to mount transferable adversarial input attack. In contrast, recent works (165; 159; 166; 167; 168; 71; 169; 170) aim to achieve high model accuracy or fidelity on a task using active learning methods. If the attacker prioritizes task accuracy, then the goal is to construct $\hat{M}_\theta$ such that the probability of $[\arg\max \hat{M}(x) == y]$ (i.e., true label) is being maximized. As for fidelity extraction, given a similarity function S(.), the goal is to construct a model $\hat{M}_\theta$ such that the similarity index $S(\hat{M}_\theta(x), M_\theta(x))$ between the output of the victim and substitute model is maximized. One of the major drawbacks of the learning-based model extraction approach is the requirement of excessive input query and access to the victim model's output score/predictions.

Table 5.1: Summary of The Existing Model Extraction Methods (9).

Type	Attack	Goal
Direct/Mathematical Recovery	(161; 71; 162; 163)	Functionally Equivalent
Active Learning/Learning	(165; 159; 166; 167; 168; 71; 169; 170; 164)	Task Accuracy/Fidelity
Side channel & Learning	(171; 23; 172; 24; 173; 174; 52; 53; 175)	Functionally Equivalent/Fidelity

Side Channel Attacks on DNNs. There has been a large body of studies on hardware/microarchitecture side channel exploitation where attackers can leak confidential system information through power, EM, and timing information on various platforms (176; 177; 72; 178; 179; 180). Recent works have demonstrated that such attack vectors can also be applied to exfiltrate sensitive DNN information (171; 23; 172; 24; 173; 174; 52). Among the existing techniques, a side-channel attack is a more practical strategy to steal sensitive information about a deeper (i.e., many layers) victim model. Typically, the goal of side-channel attack is to produce a functionally equivalent model or achieve high fidelity on a dataset. To achieve this, the attacker often supplements side-channel attacks with a learning scheme to train a substitute model using the leaked parameter information. This substitute model can later generate adversarial input samples with high transferable properties to attack the victim model more efficiently (23). Note that these side-channel attacks primarily recover model architecture or hyperparameters. However, to date, only a limited number of studies have explored the extraction of model parameters (i.e., weights) in DNN models. Recent studies on physical side channels have exhibited successful exfiltration of model parameter information, including EM side channels (171) and PCI-e bus snooping (175). These works assume the attacker has physical access to the target machines to enable hardware-based probing or snooping, which may not be practical for remote exploitation of platforms such as cloud services. The goal is to investigate the possibility of exploiting rowhammer-based side channels to perform remote stealing of model weights and explore ways to generate a substitute model with high accuracy and high fidelity on a task. Finally, such a substitute model can later generate adversarial samples with high transferable properties to the victim model.

5.2 Threat Model for Model Extraction

The attacker targets on exfiltrating internal information (i.e., model weights) from deep learning systems by exploiting the underlying hardware fault vulnerabilities in modern computing systems. We assume that the deep learning system is deployed in a resource-sharing environment to offer ML inference service. Such application paradigm is becoming popular due to the prevalence of machine-learning-as-a-service

(MLaaS) platforms (181). The attacker can control a user-space un-privileged process that runs on the machine where the victim DNN service is deployed. Our framework manifest as a *semi-black box* attack where the adversary does not have any prior knowledge of the model parameters. However, the attacker knows essential model architecture information, including model topology and layer sizes. We note that such an assumption is legitimate, as prior works demonstrate many practical ways to recover model architecture information through various side-channel exploitation (e.g., via caches (70), memory bus (23) and EM (24)).

In this chapter, we leverage the rowhammer fault attack vector commonly in today's DRAM-based memory systems as the side channel (76). Specifically, the attacker takes advantage of the fact that bit flip in *vulnerable DRAM cells* only occurs when the column-wise bit striping pattern exists in double-sided rowhammering. By leveraging such data dependency, the attacker can infer bits in the aggressor rows by observing if a bit flip occurs in *his own address space*. In other words, the attacker does not directly tamper with the victim's memory (as shown in most traditional rowhammer attacks). The attacker may share specific read-only memory with the victim DNN (e.g., ML platform binaries) either through library sharing or advanced memory deduplication feature supported in modern OS (182). We assume that a proper confinement mechanism is implemented to disallow direct access to data across processes. We further assume that the operating system and the hypervisor are benign, and appropriate kernel-space protection mechanisms are deployed to avoid direct tampering with kernel structures (183).

For substitute model training, as depicted in Table 5.2, we assume the attacker has no knowledge of gradients and is denied access to DNN output scores/predictions. Meanwhile, similar to recent related works (184; 185), we assume the attacker has access to a publicly available portion (e.g., $\leq 10\%$) of the labeled training dataset.

Table 5.2: List of Information Accessible to The Attacker for Substitute Model Training (9).

Attacker Information	Accessible
1. DNN Architecture	✓
2. HammerLeak recovered weight bits	✓
3. Gradient Computation	55
4. Train/Test Data	55
4. Victim model Output	55
5. A portion of publicly available data ($\leq 10\%$)	✓

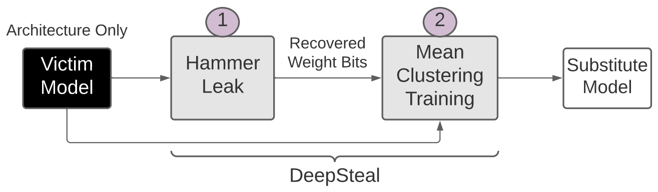


Figure 5.1: Overview of The DeepSteal Attack Framework. Stage-1: Exfiltrating DNN Partial Weight Bits Efficiently Through Exploiting Memory Fault Vulnerabilities (HammerLeak). Stage-2: with The Recovered Bits, Training A Substitute Model using Mean Clustering Weight Penalty (9).

5.3 Overview of DeepSteal

In this chapter, we describe an advanced model extraction attack framework through efficient weight bits stealing in memories. An the attack framework, *DeepSteal*, is shown in Figure 5.1. It has two key components: i) an efficient rowhammer-based weight-stealing side channel module *HammerLeak*, and ii) a substitute model training mechanism with novel *Mean Clustering* loss penalty. At stage-1 in Figure 5.1, we mount the HammerLeak attack on inference infrastructure (i.e., a remote machine running the target DNN inference service) to recover partial weight bits. We con-

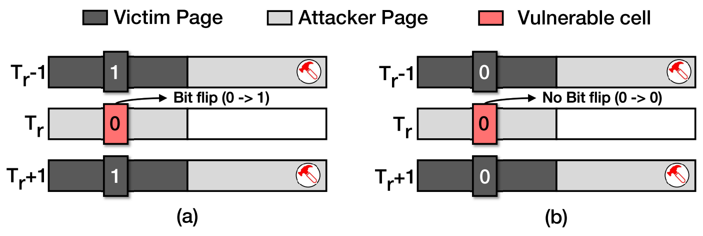


Figure 5.2: Data Dependency for Inducing a Rowhammer Fault. Here, Based on The Presence of Bit Flip in The Attacker-controlled Vulnerable Bit in The Target Row (T_r), Data from Adjacent Row from Victim Program Can Be Inferred (9)

tinue the HammerLeak for many rounds until the desired portion of weight bits is recovered. Once HammerLeak completes, at stage 2, we aim to use the leaked weight bit information and generate a substitute prototype of the victim model. To achieve this, we propose a novel neural network training algorithm that constrains the trained substitute model weight parameters to be as close as possible to the recovered partial weight info and minimize the accuracy loss. The learned substitute model will pose the following properties: i) having comparable test accuracy as the victim model; ii) exhibiting high fidelity, and iii) can be used to generate solid adversarial input samples to the victim model. We describe the details of our DeepSteal framework in the following sections.

5.4 Hammerleak: Efficient Data Stealing in Memories

In this section, we present *HammerLeak* (9), an efficient rowhammer-based information leakage attack which can steal a victim's secretive data in bulk. HammerLeak is a multi-round attack framework. In each round, it relocates the victim's model weight pages to leakable DRAM locations and performs rowhammer-based side channel to steal weight bits. Such operations are iterated over multiple rounds until sufficient bits are leaked for model extractions.

There is a large body of works demonstrating many variants of rowhammer attacks. Most of them focus on tampering with the integrity of systems, including privilege escalations (102), system denial of service (186) and, more recently, faulting DNN model parameters (2; 187). Recently, RAMBleed (76) reveals that the rowhammer fault characteristic can be leveraged to carry out *information leakage* attacks that directly infer victim secrets in memory. HammerLeak follows a similar topology where the attack leverages the fact that a column-wise data dependence is required to flip a bit for a known vulnerable memory cell successfully. Notably, under double-sided hammering, a bit flip for a vulnerable cell can succeed with high confidence if its upper and lower bits in the same column (e.g., in the aggressor rows) store the opposite bits (`1--0--1` or `0--1--0`), or fail if such pattern is not in place. Figure 5.2 illustrates such a data dependency for cells with bit flip vulnerability (in the $0 \rightarrow 1$ direction). As we can see, for the victim page in the middle with a vulnerable bit set to '0', a bit flip would only occur if the direct top and bottom bits are set to '1s', thus achieving the column-wise striped pattern. The attacker places his page in the middle row with the vulnerable cell and manages to trigger the placement of two copies of a victim's page in the corresponding aggressor rows. By observing whether a bit flip occurs in attacker's pages after hammering, the adversary can infer the secretive bit. This way, the attacker recovers secret bits iteratively to maximize the data leakage in one round of HammerLeak. To achieve this, HammerLeak adopts anonymous page swapping, bit-flip aware page release and deterministic victim page relocation to enable leakage of secret weight bit information in bulk from DNN. The further details of the HmmaerLeak attack is discussed in (9) which is contributed by M Hafizul Islam Chowdhuryy.

5.5 Substitute Model Training with Mean Clustering

At Stage-2 of DeepSteal, we leverage the bit information leaked by HammerLeak to learn a substitute prototype of the victim DNN model. To fully leverage those leaked partial bit-wise data, we propose a novel substitute model training algorithm to reconstruct a neural network model, targeting high accuracy and high fidelity. Moreover, this learned substitute model will help the attacker generate highly effective adversarial input samples to fool the victim model successfully.

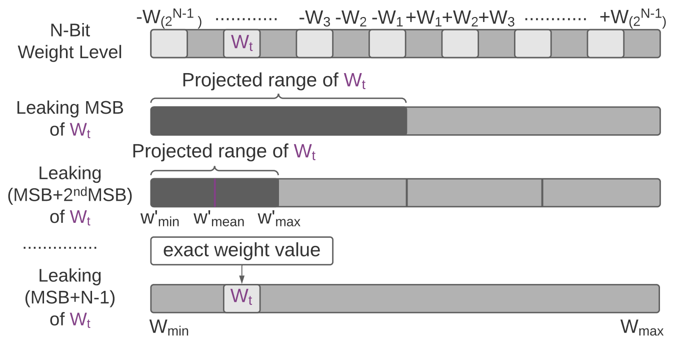


Figure 5.3: First Row: N-Bit Quantized Weight Level; Second Row: Once The MSB of Weight W_t in The Victim Model Is Leaked, We Can Narrow Down The Projected Range of W_t in The Substitute Model; Last Row: Leaking All The Bits Can Track Down The Exact Value of W_t for The Substitute Model Training (9).

5.5.1 Hammer Leaked Data Filtering

At stage 1, HammerLeak recovers a portion of the neural network weight bit information scattered across different significant bits (i.e., from LSB to MSB) for each weight. However, not all recovered bits will be used for substitute model training since it is a mixture of significant bits for each weight. As shown in Figure 5.3, it is preferred to recover MSB first for each weight parameter. Thus it forms a more minor and closed searching space (i.e., either positive or negative), rather than a full-scale space for this weight during substitute model training to minimize loss. With the knowledge of MSB, the 2^{nd} MSB or more following bits will further reduce the closed searching space. Otherwise, recovering lower significant bits without higher significant bits does not provide much helpful information about this weight's potential range. Consequently, to use the leaked bit information effectively, the attacker must filter and reorganize the leaked bits in a sequence from MSB to LSB. Therefore, before substitute model training, we sort out the leaked weight bits in the following sequence: MSB leaked, MSB+2^{nd} MSB leaked, MSB+(2^{nd} & 3^{rd} MSB) leaked, and so on, to develop a profile for each weight with a projected range, as described in Figure 5.3. Note that if no MSB is recovered, the projected weight value range will be treated as full scale.

In Figure 5.3, we visualize the relationship between i) filtered bits (leaked by HammerLeak) information of weights from a victim model and ii) the expected range of that corresponding weight during the training of the substitute model. It shows gradually leaking more bit information (i.e., MSB, MSB+2^{nd} MSB,..) of one target weight W_t can help an attacker reduce the searching space of W_t during model training. We define this expected range as *projected range* of each weight in the substitute model.

5.5.2 *Mean Clustering Optimization*

Leveraging the profile of such projected range of each weight, we propose a novel training algorithm for the substitute model using *Mean Clustering* weight penalty. It applies an additional loss penalty to the cross-entropy loss during the training process. The Mean Clustering penalty term aims to penalize each weight to converge near the mean of the projected range.

To formally define the problem, let's consider the weight matrix of the victim DNN model at layer l to be $\hat{\boldsymbol{W}}^l$. Based on the leaked weight bits at this layer, the attacker can compute the projected range of each weight in the substitute model $\boldsymbol{W}^l$. The projected range can be represented as: $\boldsymbol{W}^l_{min}$ & $\boldsymbol{W}^l_{max}$ matrix; the minimum and maximum projected value matrix corresponding to each weight in $\boldsymbol{W}^l$. Using this closed range, the projected mean matrix $\boldsymbol{W}^l_{mean}$ is computed as: $(\boldsymbol{W}^l_{max} + \boldsymbol{W}^l_{min})/2$. Next, leveraging this mean matrix, we propose to design a Mean Clustering loss penalty as highlighted in Equation (5.1). This loss term is added to the inference loss $\mathcal{L}$ and the optimization process can be formulated as:

$$\begin{aligned} \min_{\{\mathbf{W}_l\}_{l=1}^L} \; & \mathbb{E}_{\boldsymbol{x}} \; \mathcal{L}(f(\boldsymbol{x}, \{\mathbf{W}^l\}_{l=1}^L), \boldsymbol{y}) + \\ & \underbrace{\lambda \cdot \sum_{l=1}^{L} (||\mathbf{W}^l - \mathbf{W}^l_{mean}||)}_{\text{loss penalty for Mean Clustering}} \end{aligned} \tag{5.1}$$

Here, λ is a hyper-parameter that controls the strength of the loss penalty, and f($\cdot$) denotes the inference function of the DNN model for an input-label pair $(\boldsymbol{x}, \boldsymbol{y})$. The first term of the loss function in Equation (5.1) is a typical cross-entropy loss for neural network training using gradient descent. The purposed additional Mean Clustering loss penalty is to penalize each weight to converge near $\{\boldsymbol{W}^l_{mean}\}_{l=1}^L$.

5.5.3 Overall Training Algorithm

After the filtering step, we divide the weights into three categories: *Weight Set-1:* Full 8-Bit recovered, *Weight Set-2:* Partial bit recovered (i.e., MSB + n; n= 0,...,6) & *Weight Set-3:* No bit recovered. For set-1, the attacker knows the exact weight value in the victim model. Hence, we will use the exact recovered value for the substitute model by freezing (i.e., set gradient to zero) them during training. The second set of weights is trained using a new loss function in Equation (5.1). And for set-3, we do not apply the Mean Clustering loss penalty (i.e., λ=0). Both set-2 & set-3 weights are trained using standard gradient descent optimization. During training, each time before computing the loss function in Equation (5.1), we update the projected mean matrix $\{\boldsymbol{W}^l_{mean}\}_{l=1}^{L}$ using the weights of current iteration. If any weight value exceeds the projected range, it will be clipped. Finally, in the last few iterations (e.g., 40), the model will be fine-tuned ($\lambda = 0$, no clipping & low learning rate) to generate the final substitute model.

5.6 Experimental Setup

5.6.1 Attack Evaluation Metrics

To evaluate the efficacy of our DeepSteal attack, we adopt three different evaluation matrices, i.e., the accuracy of the substitute model, fidelity of the substitute model, and accuracy of the victim model under malicious input attack.

accuracy (%) It is the measurement of the percentage of test samples being correctly classified by the substitute model for a given test dataset. Note that this is the same test data used for the victim model. For an ideal successful model extraction attack, we expect the accuracy of the victim and substitute model to be almost identical.

fidelity (%) We measure the *fidelity* as the percentage of test samples with identical output prediction labels between the victim model and substitute model. This follows the definition of (71), where two models with high fidelity should agree on their label prediction for any given input sample. Ideally, an attacker should achieve 100% fidelity, where the substitute and victim model agree on all the prediction output.

Accuracy Under Attack (%) It is defined as the percentage of adversarial test samples generated from the substitute model being correctly classified by the victim model. It indicates the transferability of the adversarial examples as explained in prior (91). Ideally, if the substitute model and victim models are identical, then adversarial samples transferred from the substitute model should achieve similar efficacy (i.e., accuracy under attack) as a white-box attack (i.e., the attacker knows everything about the victim model). In this evaluation, we use the popular projected gradient descent (PGD) (27) attack to generate adversarial samples on the substitute model. The PGD attack uses L_∞ norm, $\epsilon = 0.031$ and an attack iteration step of 7 for all three datasets.

5.6.2 Hardware Configuration

We train our DNN models using GeForce GTX 1080 Ti GPU platform operating at 1481MHz and deploy the trained models in an inference testbed. The Hammer-Leak attack is evaluated on the inference testbed equipped with an Intel Haswell series processor (i5-4570) with AVX-2 instruction set support. We collect the bit flip profile of the memory modules (i.e., templating) used in the target system to identify potentially vulnerable locations in DRAM. Note that memory templating is considered a standard process for rowhammer. We leverage existing techniques as described in (99; 100; 188; 76). The system is configured with a memory subsystem with 4GB DDR3 DIMMs in either single- or dual-channel settings. Our tested DIMMs have 71% of the pages containing at least one V_c, and in total 0.017% of memory cells are vulnerable to bit flip. Compared to bit-flip profiles observed in prior work showing multiple DRAM modules with more than 98% of all rows being vulnerable (189), our system has a moderate level of vulnerability in rowhammer-induced bit flips. Finally, we profile the vulnerable DRAM cells and empirically categorize the V_c into two classes based on flip repeatability: *Strongly-leakable* cells and *Weakly-leakable* cells. Our HammerLeak only leverages *Strongly-leakable* cells for the side channel to maximize the bit stealing accuracy.

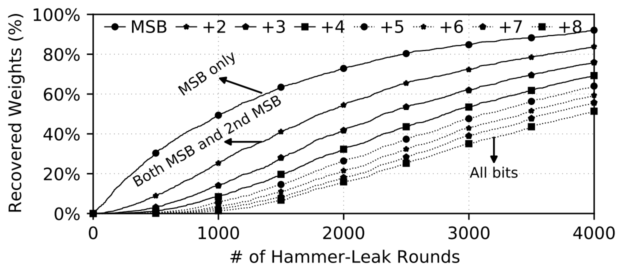
5.7 Evaluation

5.7.1 Hammerleak Performance Analysis

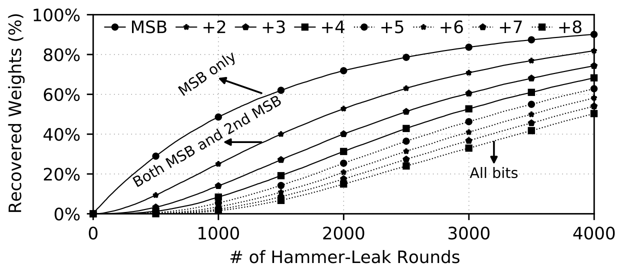
This section uses ResNet-18 as one representative DNN model for HammerLeak analysis. ResNet-18 has 21 layers with 11 million weight parameters. We perform HammerLeak on this model to investigate the efficiency of our attack in recovering model parameters. We observe that at about 4000 HammerLeak rounds, HammerLeak can steal about 90% of the MSB bits for model weights across all layers (with the lowest per layer recovery rate to be 88%). Figure 5.4 shows the percentage of weights with leaked {`MSB`} bits as well as percentage of weights with other bits simultaneously leaked together with the MSB bits (e.g., {`MSB`+2^{nd} `MSB`}) for two different layers. Along with MSB bits, we observe that the recovery rate for additional weight bits is also very high, with 55%-63% weights across all layers having the complete weight recovered. This shows the high efficiency of HammerLeak attack. Depending on the attacker's goal (in our case, a high percentage of weights with MSB exfiltrated), HammerLeak can be completed sooner than 4000 rounds. We observe that most layers have half the weights, with MSB bit recovered within 1000 rounds.

5.7.2 DeepSteal Experimental Results: CIFAR-10

In Table 5.3, we evaluate the performance of DeepSteal attack on CIFAR-10 dataset for three different architectures. Further, we show an ablation study showing the impact of using several rounds of HammerLeak attack information for DeepSteal



((a)) Layer 1



((b)) Another random layer (Layer 5 under illustration)

Figure 5.4: Percentage of Weights with MSB or More Bits Recovered. +x Denotes Number of Consecutive Higher Order Bits Recovery (i.e., +3 Represents Weights with All Three MSB Bits Recovered) (9).

attack. As a baseline method, we compare the architecture-only case (i.e., 0-bit information leaked). We assume the attacker only knows the victim model architecture for the baseline case. Then, a substitute model with the same architecture is trained using a similar setting (i.e., less than *8*% available data). On the other hand, we treat the white-box case as the best-case scenario where the attacker knows the victim model's information (i.e., weights, biases and architecture). In summary, with more recovered weight bits, DeepSteal achieves better accuracy, fidelity and adversarial example attack efficacy. For instance, our substitute model can generate effective transferable adversarial example with similar efficacy (i.e., ∼*0*%) as white-box attack for both ResNet-18 and ResNet-34.

In our evaluation, the residual victim models (ResNet-18 & ResNet-34) have

Table 5.3: Summary of CIFAR-10 Results for Three Different DNN Architectures. We Report Two Different Cases of DeepSteal Attack i) All Bits: Where We Use All The Bit Information (i.e., All 8 Plots) Plotted in Figure 5.4. According To This Plot, for Each # of HammerLeak Attack Rounds Along x-axis, We Take The Percentage of Bits Recovered for All 8 Plots (e.g., MSB, MSB+2nd MSB & So On). ii) MSB: We Only Use The MSB Bit Information Labeled As MSB Curve in Figure 5.4 (9).

Method			*ResNet-18*				*ResNet-34*				*VGG-11*			
# of HammerLeak Rounds	Method	Case	Time (days)	Accuracy (%)	Fidelity (%)	Accuracy under Attack (%)	Time (days)	Accuracy (%)	Fidelity (%)	Accuracy Under Attack (%)	Time (days)	Accuracy (%)	Fidelity (%)	Accuracy Under Attack (%)
Baseline	Arch. Only	-	-	73.18	74.29	61.33	-	72.22	72.85	62.69	-	70.76	72.06	61.19
1500	DeepSteal	All Bits	4.5	74.33	75.38	53.64	7.6	74.43	75.2	55.99	3.9	72.3	73.34	62.24
		MSB	3.9	76.61	77.56	50.4	6.5	76.77	77.53	53.47	3.4	72.67	73.89	58.19
3000	DeepSteal	All Bits	8.9	86.32	87.86	5.24	15.3	85.62	86.72	3.93	7.8	81.03	82.88	36.45
		MSB	7.8	86.93	88.51	8.13	12.9	87.19	88.39	4.61	6.7	80.15	81.52	26.85
4000	**DeepSteal**	All Bits	***11.9***	***89.05***	***90.74***	***1.94***	***20.4***	***88.17***	***89.27***	***1.44***	***10.4***	***84.59***	***86.24***	***16.87***
		MSB	***10.4***	***89.59***	***91.6***	***1.61***	***17.4***	***90.16***	***91.8***	***1.03***	***8.9***	***81.56***	***83.33***	***18.55***
Best-Case	White-box	-	-	93.16	100.0	0.0	-	93.11	100.0	0.0	-	89.96	100.0	4.63

93.16% & 93.11% inference accuracy, respectively. As shown in Figure 5.4, after 4000 rounds of HammerLeak attack, the adversary could recover 90% of the MSB bits (∼11.52% of total bits). By only utilizing the leaked MSB bits, the attacker can recover up to 89.05/88.17% test accuracy for the ResNet (18/34) models. Additionally, for the *All Bits* case in Table 5.3, after paying an additional time cost (i.e., *1.66*×), the performance of DeepSteal attack only exhibits marginal improvement on residual models. In contrast, the larger model with a different architecture topology (i.e., VGG-11 with 132 Million parameters) highly benefits from the additional information of all the bits. For VGG, we observe a ∼*3*% improvement by using all the filtered bits compared to MSB only. We also observe a similar pattern in accuracy recovery at 3000 rounds of HammerLeak attack.

Next, we evaluate the adversarial attack performance for 3000 and 4000 rounds of HammerLeak attack. In Table 5.3, we show that our substitute model can transfer effective adversarial samples to the victim model across all three architectures. In particular, for ResNet models, our substitute model generates adversarial examples that demonstrate close to the white-box attack efficacy (i.e., within 2% of the best case result) with 4000 rounds of HammerLeak attack. As for VGG model, which is already known as a robust architecture (27), our substitute model generated adversary reaches within ∼12%-14% of an ideal white-box attack. Nevertheless, our attack efficacy still shows an improvement of about ∼25%-60% across all three architectures compared to the baseline (i.e., architecture only) technique.

Table 5.4: We Evaluate DeepSteal Attack Against State-of-the-art Techniques Across Three Different Domains As Case Studies. In Each of The Cases, Only Our Attack Performs On Par with The SOTA Methods Across All Three Evaluation Metrics (9).

Case Study	*Method*	*Objective*	*Model*	*Accuracy (%)*	*Fidelity (%)*	*Accuracy Under Attack (%)*
Regularization & Data Augmentation	Fully-Supervised (165)	Accuracy/Fidelity	WideResNet-28	86.51	87.37	-
	Rand-Augment (184)	Accuracy	WideResNet-28	87.4	-	-
	Auto-Augment (185)	Accuracy	WideResNet-28	87.7	-	-
	DeepSteal (ours)	***Accuracy/Fidelity/Attack***	***WideResNet-28***	***91.93***	***93.45***	***0.05***
Model Extraction	Side Channel (23; 173; 53; 52)	Accuracy/Fidelity/Attack	ResNet-18	72.68	73.59	62.58
	DeepSteal (ours)	***Accuracy/Fidelity/Attack***	***ResNet-18***	***90.02***	***91.67***	***1.2***
Input Attack	Black-Box (Inception-V1) (190)	Adversarial Attack	ResNet-18	-	-	20.47
	White-Box (PGD/Trades) (27; 191)	Adversarial Attack	ResNet-18	-	-	0.0
	DeepSteal (ours)	***Adversarial Attack***	***ResNet-18***	**-**	**-**	***1.2***

Finally, we consider an attack scenario where the attacker has a strict time budget. In this scenario, let us assume he/she can only afford to run 1500 rounds of HammerLeak attack while prioritizing MSBs (e.g., only *3.9* days of attack time). As summarized in Table 5.3, even such a restricted attack can generate compelling adversarial examples to lower accuracy under attack by *7%-11%* for ResNet models and by *3%* for VGG-11 compared to baseline. One key observation for this low budget (i.e., 1500 round attack) attack is that attacker can generate a much more effective substitute model by only using MSB information rather than all the bits. With limited bit information (e.g., 50% MSB only), putting strict penalization (i.e., mean clustering) on the weights during training does not help the substitute model accuracy. In fact, for VGG-11, it becomes worse than the baseline method. As a result, for DeepSteal attack with limited partial bit information, using the relaxation of the weight constraints (i.e., MSB only) can be more effective than using all the available filtered bits.

5.7.3 Comparison to State-of-the-art Techniques

In Table 5.4, we summarize the standing of our DeepSteal attack compared with existing model recovery methods for three different domains of applications. We can see existing model regularization (165; 71) and data augmentation techniques (184; 185) are useful in training deep models with limited data. However, our substitute model achieves a much higher accuracy (i.e., ∼*3*%). Other existing side channel attacks (23; 173; 53; 52; 24) fall into a similar attack category as DeepSteal. Among them, (24) only applies to binary neural networks. On the contrary, our attack is a more general version of the attack applicable to any bit-width. Other side-channel attacks (23; 173; 53; 52) focus on recovering the architecture and then training the model with limited data. To compare them, we assume the attacker knows the exact model architecture. Our DeepSteal can leverage the leaked weight bits to improve the attack efficacy further. From this point, our attack outperforms prior architecture-only model extraction attacks with ∼18% improvement in accuracy and ∼61% improvement in degrading the accuracy under adversarial attack. Note that while DeepSteal is a semi-Black Box attack, DeepSteal actually can achieve 1.2% accuracy under attack, which is extremely close to a white-box attack performance (i.,e., 0%). We observe a 19% improvement in attack performance compared to a powerful black-box substitute model (e.g., Inception-V1) attack.

5.7.4 Impact of Bit Stealing Errors

Bit stealing accuracy can be influenced if expected bit flips do not occur. This section analyses the bit errors in the rowhammer-based side channel. Specifically, we profile bit errors on 4 different vulnerable DIMMs with random bits set in the victim's pages. Note that HammerLeak only leverages *Strongly-leakable cells* that exhibit consistent bit flips in double-sided rowhammer. Our analysis reveals that about 70% of the flippable DRAM cells fall into this category. Our results show very high and stable bit stealing accuracy – on average 95.7% – across all tested DIMMs.

To quantify the impact of bit errors, we analyse the effectiveness of DeepSteal under a range of bit error rates. Specifically, under the setting where 90% of raw MSB bits are exfiltrated by HammerLeak, we inject random errors at a certain rate across each model layer into the recovered bits. Figure 5.5 shows the performance of the substitute model (in terms of Accuracy and Accuracy Under Attack) when the bit error ranges from 0% to 10% for ResNet-18. The results demonstrate that *low bit error (0-5%) has a negligible effect on the performance of the substitute model attack.* Moreover, the accuracy of the substitute model stays stable even as the error rate reaches 10%. On the other hand, the increase in error rate (5-10%) in the recovered bits causes the DeepSteal performance in Accuracy Under Attack to degrade gradually. Nevertheless, we can still observe a much higher attack efficiency of DeepSteal compared to the baseline approaches, as shown in Table 5.4.

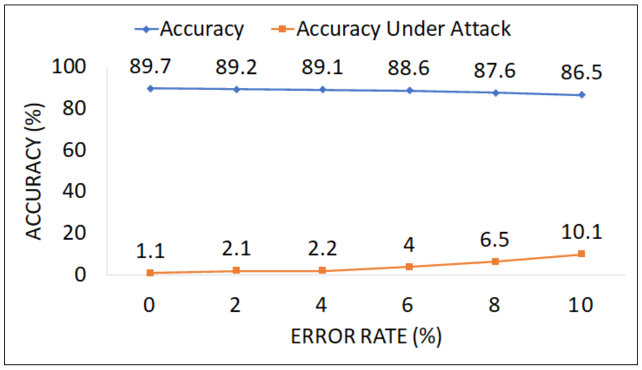


Figure 5.5: Analysis of The Impact of The Recovered Bit Error Rate (%) on DeepSteal Attack Performance for ResNet-18 (9).

5.8 Discussion

5.8.1 Countermeasures for DeepSteal

Effect of adversarial training on a transferred adversarial sample. One potential approach in defending against adversarial samples is to train the model using the attacked samples, popularly known as *adversarial training (27)*. In Table 5.5, we evaluate the target victim model, defended with adversarial training. Naturally, a model trained with adversarial examples becomes more resistant to both white-box & black-box adversarial attacks. We observe a similar pattern in our experiments. Still, DeepSteal could achieve *4% & 6%* improvement in attacking the target model trained with the adversarial samples compared to the baseline (i.e., architecture only + learning) for ResNet-18 & VGG-11, respectively. We conclude that the existing white-box adversarial defense may lower the transferability of adversarial samples from our substitute model, but fails to prevent the accuracy and fidelity extraction.

In addition, DeepSteal follows a more strict threat model, which does not require access to output logits/predictions. In contrast, prior strong transferable adversarial attacks (192; 193; 194; 195) require model queries as access to output logits/prediction. Even in this minimal setting, our attack DeepSteal outperforms the existing model stealing attacks (i.e., architecture only + learning) (23) in attacking a well-defended (i.e., adversarial training) target model. Finally, it is worth noting that current adversarial defenses (e.g., adversarial training) come with additional training costs and inference accuracy degradation.

Table 5.5: After Adversarial Training, The White-box Accuracy Under Attack Improved to 43.12% & 35.71% for ResNet-18 and VGG-11 Models Respectively . Here We Report The Performance of DeepSteal Attack Using Recovered Bit Information after 4000 Rounds of HammerLeak Attack (9).

Training Data (%)	*Accuracy (%)*	*Fidelity (%)*	*Accuracy Under Attack (%)*
ResNet-18 (83.62%)			
Baseline	72.87	72.66	80.42
DeepSteal	82.84(↑ 10)	81.67 (↑ 9)	76.78 (↓ 4)
VGG-11 (80.21%)			
Baseline	70.71	71.3	76.63
DeepSteal	80.65 (↑ 10)	81.13 (↑ 10)	70.28 (↓ 6)

5.9 Conclusion

Training deep neural networks requires heavy computational resources and sensitive domain-specific private user data. Thus, any potential breach in model privacy through leakage of sensitive model parameters may cost the service provider a heavy financial penalty. Consequently, the IP of a pre-trained DNN model is critical to protect against adversarial threats (i.e., model extraction). In this chapter, our developed DeepSteal attack exposes this threat of an effective model extraction attack in practical settings. In particular, our novel system-level weight bit extraction method HammerLeak enables fast and efficient weight stealing for large-scale DNN applications. It can recover a significant portion of the weight bits of a DNN model with millions of wight parameters. On top of that, Mean Clustering training algorithm can leverage this information to effectively launch a strong adversarial input attack on the victim model. The efficacy of the attack algorithm is validated through extensive experimental evaluation. Such a model extraction threat should encourage future work in this direction to protect the IP of large-scale DNN models.

Chapter 6

CONCLUSION AND OUTLOOK

The advanced adversarial weight perturbation attack algorithms presented in this thesis pose a new threat for the DNN-powered applications running in a computing platform with main memory, such as cloud/edge servers, desktops, mobile phones, etc. The proposed adversarial weight attack methodology can cause a devastating effect by modifying an extremely small amount of DNN weight parameters stored in computer memory through memory fault injection. For example, in a self-driving car application, the 1-to-1 stealthy T-BFA can attack one particular class (e.g, stop sign), while keeping other class classifications correct, through modifying less than 0.001% of total weight parameters. As a result, the system would be difficult to detect such attacks, resulting in devastating consequences. Hence ensuring the safe and secure deployment of DNN has become a top research priority.

However, to analyze and protect DNNs against practical threats first, we need to design possible security breaches in deep learning applications. Adversarial input example attack did a good job in pin-pointing one key limitation of deep learning algorithms. They show that slightly perturbed inputs can fool a DNN and achieve malicious objectives. In a different, yet related track, we focus on the security domain of DNN weights and design a strong targeted adversarial weight attack methodology. In particular, in this thesis, we design the first targeted adversarial weight attack on a quantized noise-resilient DNN. Quantized DNNs are widely used in resource-constrained edge devices, making their security even more important. The proposed attack schemes further show the vulnerability of DNN models to malicious attacks. Our analysis shows that a wide range of DNN behaves differently to different variants of attacks. To make it worse, Such perturbation of model parameters is practically feasible nowadays because of the development of advanced computer hardware fault injection techniques, such as rowhammer attack, laser beam attack, and under-voltage attack. Thus we believe it is extremely important to study these maliciously adversarial weight attacks to better understand the underlying vulnerability of DNNs, which will lead to future more robust DNN algorithm development to tackle such attacks or model failure. While studying attack algorithms is important, we have worked on potential defensive solutions to make the behavior of DNN models more secure under adversarial weight attack.

On top of that, our study includes an even strict threat model where we demonstrated a practical black-box attack in multi-tenant cloud FPGA. Finally, we have maneuvered the bit-flip attack to breach deep learning privacy. Such a transformation of adversarial weight attack to model leakage attack has opened a new set of challenges in protection of the IP of pre-trained deep learning models. Hence, We strongly recommend more investigation of DNN model parameter security/privacy attacks/defenses to better understand DNN behavior and secure them. We hope this thesis should greatly benefit the emerging and exploding AI security community in the future.

REFERENCES

[1] A. S. Rakin, Z. He, and D. Fan, “Bit-flip attack: Crushing neural network with progressive bit search,” in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019, pp. 1211–1220.

[2] F. Yao, A. S. Rakin, and D. Fan, “Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips,” in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020, pp. 1463–1480.

[3] A. S. Rakin, Z. He, J. Li, F. Yao, C. Chakrabarti, and D. Fan, “T-bfa: Targeted bit-flip adversarial weight attack,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[4] Y. Liu, L. Wei, B. Luo, and Q. Xu, “Fault injection attack on deep neural network,” in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 131–138.

[5] P. Zhao, S. Wang, C. Gongye, Y. Wang, Y. Fei, and X. Lin, “Fault sneaking attack: A stealthy framework for misleading deep neural networks,” in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[6] Z. He, A. S. Rakin, J. Li, C. Chakrabarti, and D. Fan, “Defending and harnessing the bit-flip based adversarial weight attack,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 095–14 103.

[7] A. S. Rakin, Z. He, and D. Fan, “Tbt: Targeted neural network attack with bit trojan,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 198–13 207.

[8] A. S. Rakin, Y. Luo, X. Xu, and D. Fan, “Deep-dup: An adversarial weight duplication attack framework to crush deep neural network in multi-tenant fpga,” in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1919–1936.

[9] A. S. Rakin, M. H. I. Chowdhuryy, F. Yao, and D. Fan, “Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories,” in *43rd IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2022.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[11] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[13] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning." in *AAAI*, vol. 4, 2017, p. 12.

[14] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[15] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[16] I. Stoica, D. Song, R. A. Popa, D. Patterson, M. W. Mahoney, R. Katz, A. D. Joseph, M. Jordan, J. M. Hellerstein, J. E. Gonzalez *et al.*, "A berkeley view of systems challenges for AI," *arXiv preprint arXiv:1712.05855*, 2017.

[17] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[18] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *ICLR*, 2015.

[19] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *CoRR*, vol. abs/1708.06733, 2017.

[20] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *25nd Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018.* The Internet Society, 2018.

[21] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitraş, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," *arXiv preprint arXiv:1906.01017*, 2019.

[22] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE symposium on security and privacy (SP).* IEEE, 2017, pp. 3–18.

[23] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. Sherwood *et al.*, "Deepsniffer: A dnn model extraction framework based on learning architectural hints," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 385–399.

[24] H. Yu, H. Ma, K. Yang, Y. Zhao, and Y. Jin, "Deepem: Deep neural networks model recovery through em side-channel information leakage," in *IEEE International Symposium on Hardware Oriented Security and Trust*, 2020, pp. 209–218.

[25] L. Song, R. Shokri, and P. Mittal, "Membership inference attacks against adversarially robust deep learning models," in *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2019, pp. 50–56.

[26] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. ACM, 2015, pp. 1322–1333.

[27] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJzIBfZAb

[28] Y. Zhang, R. Jia, H. Pei, W. Wang, B. Li, and D. Song, "The secret revealer: Generative model-inversion attacks against deep neural networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 253–261.

[29] A. S. Rakin, Z. He, and D. Fan, "Tbt: Targeted neural network attack with bit trojan," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[30] A. S. Rakin, M. H. I. Chowdhuryy, F. Yao, and D. Fan, "Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories," *arXiv preprint arXiv:2111.04625*, 2021.

[31] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *arXiv preprint arXiv:1802.00420*, 2018.

[32] A. S. Rakin, Y. Wang, S. Aeron, T. Koike-Akino, P. Moulin, and K. Parsons, "Towards universal adversarial examples and defenses," in *2021 IEEE Information Theory Workshop (ITW)*. IEEE, 2021, pp. 1–6.

[33] Z. He, A. S. Rakin, and D. Fan, "Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 588–597.

[34] A. S. Rakin, Z. He, L. Yang, Y. Wang, L. Wang, and D. Fan, "Robust sparse regularization: Defending adversarial attacks via regularized sparse network," in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, 2020, pp. 125–130.

[35] A. S. Rakin and D. Fan, "Defense-net: Defend against a wide range of adversarial attacks through adversarial detector," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2019, pp. 332–337.

[36] Y. Wang, S. Aeron, A. S. Rakin, T. Koike-Akino, and P. Moulin, "Robust machine learning via privacy/rate-distortion theory," in *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2021, pp. 1320–1325.

[37] S. K. Cherupally, A. S. Rakin, S. Yin, M. Seok, D. Fan, and J.-s. Seo, "Leveraging noise and aggressive quantization of in-memory computing for robust dnn hardware against adversarial input and weight attacks," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 559–564.

[38] H. Ren, T. Huang, and H. Yan, "Adversarial examples: attacks and defenses in the physical world," *International Journal of Machine Learning and Cybernetics*, vol. 12, no. 11, pp. 3325–3336, 2021.

[39] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *International Symposium on Computer Architecture*. IEEE Press, 2014, pp. 361–372.

[40] M. Agoyan, J.-M. Dutertre, A.-P. Mirbaha, D. Naccache, A.-L. Ribotta, and A. Tria, "How to flip a bit?" in *2010 IEEE 16th International On-Line Testing Symposium*. IEEE, 2010, pp. 235–239.

[41] Y. Luo, C. Gongye, Y. Fei, and X. Xu, "Deepstrike: Remotely-guided fault injection attacks on dnn accelerator in cloud-fpga," *arXiv preprint arXiv:2105.09453*, 2021.

[42] J. Li, A. S. Rakin, Y. Xiong, L. Chang, Z. He, D. Fan, and C. Chakrabarti, "Defending bit-flip attack through dnn weight reconstruction," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[43] J. Li, A. S. Rakin, Z. He, D. Fan, and C. Chakrabarti, "Radar: Run-time adversarial weight attack detection and accuracy recovery," *arXiv preprint arXiv:2101.08254*, 2021.

[44] A. S. Rakin, L. Yang, J. Li, F. Yao, C. Chakrabarti, Y. Cao, J.-s. Seo, and D. Fan, "Ra-bnn: Constructing robust & accurate binary neural network to simultaneously defend adversarial bit-flip attack and improve accuracy," *arXiv preprint arXiv:2103.13813*, 2021.

[45] S. K. Cherupally, J. Meng, A. S. Rakin, S. Yin, I. Yeo, S. Yu, D. Fan, and J.-S. Seo, "Improving the accuracy and robustness of rram-based in-memory computing against rram hardware noise and adversarial attacks," *Semiconductor Science and Technology*, vol. 37, no. 3, p. 034001, 2022.

[46] B. G. Doan, E. Abbasnejad, and D. C. Ranasinghe, "Februus: Input purification defense against trojan attacks on deep neural network systems," in *Annual Computer Security Applications Conference*, 2020, pp. 897–912.

[47] Y. Li, X. Lyu, N. Koren, L. Lyu, B. Li, and X. Ma, "Anti-backdoor learning: Training clean models on poisoned data," *Advances in Neural Information Processing Systems*, vol. 34, pp. 14 900–14 912, 2021.

[48] J. Li, A. S. Rakin, X. Chen, Z. He, D. Fan, and C. Chakrabarti, "Ressfl: A resistance transfer framework for defending model inversion attack in split federated learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 10 194–10 202.

[49] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients-how easy is it to break privacy in federated learning?" *Advances in Neural Information Processing Systems*, vol. 33, pp. 16 937–16 947, 2020.

[50] D. Chen, N. Yu, Y. Zhang, and M. Fritz, *GAN-Leaks: A Taxonomy of Membership Inference Attacks against Generative Models.* New York, NY, USA: Association for Computing Machinery, 2020, p. 343–362. [Online]. Available: https://doi.org/10.1145/3372297.3417238

[51] H. Yu, H. Ma, K. Yang, Y. Zhao, and Y. Jin, "Deepem: Deep neural networks model recovery through em side-channel information leakage," 2020.

[52] M. Yan, C. W. Fletcher, and J. Torrellas, "Cache telepathy: Leveraging shared resource attacks to learn dnn architectures," in *USENIX Security Symposium*, 2020, pp. 2003–2020.

[53] Y. Xiang, Z. Chen, Z. Chen, Z. Fang, H. Hao, J. Chen, Y. Liu, Z. Wu, Q. Xuan, and X. Yang, "Open dnn box by power side-channel attack," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 11, pp. 2717–2721, 2020.

[54] D. Das, A. Golder, J. Danial, S. Ghosh, A. Raychowdhury, and S. Sen, "X-deepsca: Cross-device deep learning side channel attack," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[55] C. Roscian, A. Sarafianos, J.-M. Dutertre, and A. Tria, "Fault model analysis of laser-induced faults in sram memory cells," in *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography.* IEEE, 2013, pp. 89–98.

[56] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3. IEEE Press, 2014, pp. 361–372.

[57] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip feng shui: Hammering a needle in the software stack," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 1–18.

[58] S. Hong, P. Frigo, Y. Kaya, C. Giuffrida, and T. Dumitras, "Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks," *CoRR*, vol. abs/1906.01017, 2019. [Online]. Available: http://arxiv.org/abs/1906.01017

[59] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[60] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," *arXiv preprint arXiv:1807.10029*, 2018.

[61] I. Hubara *et al.*, "Binarized neural networks," in *Advances in neural information processing systems*, 2016, pp. 4107–4115.

[62] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[63] S. Angizi, Z. He, A. S. Rakin, and D. Fan, "Cmp-pim: an energy-efficient comparator-based processing-in-memory neural network accelerator," in *Proceedings of the 55th Annual Design Automation Conference.* ACM, 2018, p. 105.

[64] Z. He, S. Angizi, A. S. Rakin, and D. Fan, "Bd-net: A multiplication-less dnn with binarized depthwise separable convolution," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI).* IEEE, 2018, pp. 130–135.

[65] A. S. Rakin, S. Angizi, Z. He, and D. Fan, "Pim-tgan: A processing-in-memory accelerator for ternary generative adversarial networks," in *2018 IEEE 36th International Conference on Computer Design (ICCD).* IEEE, 2018, pp. 266–273.

[66] Z. He, L. Yang, S. Angizi, A. S. Rakin, and D. Fan, "Sparse bd-net: A multiplication-less dnn with sparse binarized depth-wise separable convolution," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 16, no. 2, pp. 1–24, 2020.

[67] A. S. Rakin, Z. He, J. Li, F. Yao, C. Chakrabarti, and D. Fan, "T-bfa: Targeted bit-flip adversarial weight attack," *arXiv preprint arXiv:2007.12336*, 2020.

[68] G. Provelengios, D. Holcomb, and R. Tessier, "Power wasting circuits for cloud fpga attacks," in *30th International Conference on Field Programmable Logic and Applications (FPL)*, 2020.

[69] Y. Zha and J. Li, "Virtualizing fpgas in the cloud," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 845–858.

[70] S. Hong, M. Davinroy, Y. Kaya, D. Dachman-Soled, and T. Dumitraş, "How to 0wn nas in your spare time," *arXiv preprint arXiv:2002.06776*, 2020.

[71] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot, "High accuracy and high fidelity extraction of neural networks," in *USENIX Security Symposium*, 2020, pp. 1345–1362.

[72] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, "Rendered insecure: Gpu side channel attacks are practical," in *ACM SIGSAC conference on computer and communications security*, 2018, pp. 2139–2153.

[73] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *IEEE symposium on security and privacy*, 2015, pp. 605–622.

[74] F. Yao, G. Venkataramani, and M. Doroslovački, "Covert timing channels exploiting non-uniform memory access based architectures," in *Great Lakes Symposium on VLSI*, 2017, pp. 155–160.

[75] M. H. I. Chowdhuryy, H. Liu, and F. Yao, "Branchspec: Information leakage attacks exploiting speculative branch instruction executions," in *IEEE 38th International Conference on Computer Design*, 2020, pp. 529–536.

[76] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "Rambleed: Reading bits in memory without accessing them," in *IEEE Symposium on Security and Privacy*, 2020, pp. 695–711.

[77] A. S. Rakin, Y. Luo, X. Xu, and D. Fan, "Deep-Dup: An adversarial weight duplication attack framework to crush deep neural network in Multi-Tenant FPGA," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 1919–1936. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/rakin

[78] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research)," *URL http://www. cs. toronto. edu/kriz/cifar. html*, 2010.

[79] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Transactions on Evolutionary Computation*, 2019.

[80] M. A. Haque, A. Verma, J. S. R. Alex, and N. Venkatesan, "Experimental evaluation of cnn architecture for speech recognition," in *First International Conference on Sustainable Technologies for Computational Intelligence.* Springer, 2020, pp. 507–514.

[81] M.-T. Luong, M. Kayser, and C. D. Manning, "Deep neural language models for machine translation," in *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, 2015, pp. 305–309.

[82] Y. Lu, X. Xiong, W. Zhang, J. Liu, and R. Zhao, "Research on classification and similarity of patent citation based on deep learning," *Scientometrics*, pp. 1–27, 2020.

[83] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017, pp. 39–57.

[84] X. Xu, X. Chen, C. Liu, A. Rohrbach, T. Darell, and D. Song, "Can you fool ai with adversarial examples on a visual turing test?" *arXiv preprint arXiv:1709.08693*, 2017.

[85] H. Chen, H. Zhang, P.-Y. Chen, J. Yi, and C.-J. Hsieh, "Show-and-fool: Crafting adversarial examples for neural image captioning," *arXiv preprint arXiv:1712.02051*, 2017.

[86] J. H. Metzen, M. C. Kumar, T. Brox, and V. Fischer, "Universal adversarial perturbations against semantic image segmentation," *stat*, vol. 1050, p. 19, 2017.

[87] M. Cheng, J. Yi, H. Zhang, P.-Y. Chen, and C.-J. Hsieh, "Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples," *arXiv preprint arXiv:1803.01128*, 2018.

[88] N. Carlini and D. Wagner, "Audio adversarial examples: Targeted attacks on speech-to-text," *arXiv preprint arXiv:1801.01944*, 2018.

[89] M. Sun, F. Tang, J. Yi, F. Wang, and J. Zhou, "Identify susceptible locations in medical records via adversarial attacks on deep predictive models," *arXiv preprint arXiv:1802.04822*, 2018.

[90] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," *arXiv preprint arXiv:1611.02770*, 2016.

[91] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," *arXiv preprint arXiv:1605.07277*, 2016.

[92] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 2017, pp. 15–26.

[93] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv preprint arXiv:1607.02533*, 2016.

[94] J. Kos and D. Song, "Delving into adversarial attacks on deep policies," *arXiv preprint arXiv:1705.06452*, 2017.

[95] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 2016, pp. 372–387.

[96] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2574–2582.

[97] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[98] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 8, pp. 1555–1571, 2019.

[99] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another flip in the wall of rowhammer defenses," in *2018 IEEE Symposium on Security and Privacy*, 2018, pp. 245–261.

[100] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks," in *IEEE Symposium on Security and Privacy*, 2019, pp. 55–71.

[101] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer. js: A remote software-induced fault attack in javascript," in *International conference on detection of intrusions and malware, and vulnerability assessment.* Springer, 2016, pp. 300–321.

[102] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," *Black Hat*, vol. 15, p. 71, 2015.

[103] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[104] Z. He, A. S. Rakin, and D. Fan, "Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[105] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *CoRR*, vol. abs/1412.6572, 2014.

[106] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[107] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "Drama: Exploiting dram addressing for cross-cpu attacks," in *USENIX Security Symposium*, 2016.

[108] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another flip in the wall of rowhammer defenses," in *2018 IEEE Symposium on Security and Privacy (SP).* IEEE, 2018, pp. 245–261.

[109] D. Wu, S.-T. Xia, and Y. Wang, "Adversarial weight perturbation helps robust generalization," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[110] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," in *International Conference on Learning Representations*, 2019.

[111] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks."

[112] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[113] M. Zou, Y. Shi, C. Wang, F. Li, W. Song, and Y. Wang, "Potrojan: powerful neural-level trojan designs in deep learning models," *arXiv preprint arXiv:1802.03043*, 2018.

[114] Y. Liu, Y. Xie, and A. Srivastava, "Neural trojans," in *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 2017, pp. 45–48.

[115] T. Liu, W. Wen, and Y. Jin, "Sin 2: Stealth infection on neural network—a low-cost agile neural trojan attack methodology," in *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2018, pp. 227–230.

[116] J. Clements and Y. Lao, "Hardware trojan attacks on neural networks," *arXiv preprint arXiv:1806.05768*, 2018.

[117] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," *Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks*, p. 0, 2019.

[118] W. Li, J. Yu, X. Ning, P. Wang, Q. Wei, Y. Wang, and H. Yang, "Hu-fu: Hardware and software collaborative attack framework against neural networks," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2018, pp. 482–487.

[119] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018, pp. 273–294.

[120] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava, "Detecting backdoor attacks on deep neural networks by activation clustering," *arXiv preprint arXiv:1811.03728*, 2018.

[121] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC).* IEEE, 2018, pp. 1–6.

[122] L. Batina, S. Bhasin, D. Jap, and S. Picek, "Csi nn: Reverse engineering of neural network architectures through electromagnetic side channel," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 515–532.

[123] L. Cojocar, K. Razavi, C. Giuffrida, and H. Bos, "Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks," *S&P'19*, 2019.

[124] C. Ramesh, S. B. Patil, S. N. Dhanuskodi, G. Provelengios, S. Pillement, D. Holcomb, and R. Tessier, "Fpga side channel attacks without physical access," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).* IEEE, 2018, pp. 45–52.

[125] I. Giechaskiel, K. B. Rasmussen, and K. Eguro, "Leaky wires: Information leakage and covert communication between fpga long wires," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security.* ACM, 2018, pp. 15–27.

[126] S. Yazdanshenas and V. Betz, "The costs of confidentiality in virtualized fpgas," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2019.

[127] A. Khawaja, J. Landgraf, R. Prakash, M. Wei, E. Schkufza, and C. J. Rossbach, "Sharing, protection, and compatibility for reconfigurable fabric with amorphos," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 107–127.

[128] S. Moini, S. Tian, J. Szefer, D. Holcomb, and R. Tessier, "Remote power side-channel attacks on cnn accelerators in fpgas," *arXiv preprint arXiv:2011.07603*, 2020.

[129] "Machine learning on aws," 2020, https://aws.amazon.com/machine-learning/?nc1=h_ls.

[130] "Cloud automl," 2020, https://cloud.google.com/automl.

[131] O. Knodel, P. Lehmann, and R. G. Spallek, "Rc3e: Reconfigurable accelerators in data centres and their provision by adapted service models," in *2016 IEEE 9th International Conference on Cloud Computing (CLOUD).* IEEE, 2016, pp. 19–26.

[132] S. Yazdanshenas, "Datacenter-optimized fpgas," Ph.D. dissertation, 2019.

[133] C. R. I. G. Jacob Buckman, Aurko Roy, "Thermometer encoding: One hot way to resist adversarial examples," *International Conference on Learning Representations*, 2018, accepted as poster. [Online]. Available: https://openreview.net/forum?id=S18Su--CW

[134] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.* ACM, 2015, pp. 161–170.

[135] X. Zhang, H. Ye, J. Wang, Y. Lin, J. Xiong, W.-m. Hwu, and D. Chen, "Dnnexplorer: A framework for modeling and exploring a novel paradigm of fpga-based dnn accelerator," *arXiv preprint arXiv:2008.12745*, 2020.

[136] P. Xu, X. Zhang, C. Hao, Y. Zhao, Y. Zhang, Y. Wang, C. Li, Z. Guan, D. Chen, and Y. Lin, "Autodnnchip: An automated dnn chip predictor and builder for both fpgas and asics," in *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 40–50.

[137] *AMBA AXI and ACE Protocol Specification*, ARM, 2013.

[138] J. Krautter, D. R. Gnad, and M. B. Tahoori, "Fpgahammer: remote voltage fault attacks on shared fpgas, suitable for dfa on aes," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 44–68, 2018.

[139] D. R. Gnad, F. Oboril, and M. B. Tahoori, "Voltage drop-based fault attacks on fpgas using valid bitstreams," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL).* IEEE, 2017, pp. 1–7.

[140] D. Mahmoud and M. Stojilović, "Timing violation induced faults in multi-tenant fpgas," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 2019, pp. 1745–1750.

[141] D. G. Mayer, B. Kinghorn, and A. A. Archer, "Differential evolution–an easy and efficient evolutionary algorithm for model optimisation," *Agricultural Systems*, vol. 83, no. 3, pp. 315–328, 2005.

[142] K. V. Price, "Differential evolution," in *Handbook of Optimization.* Springer, 2013, pp. 187–214.

[143] L. Zhang, X. Xu, C. Zhou, M. Ma, and Z. Yu, "An improved differential evolution algorithm for optimization problems," in *Advances in Computer Science, Intelligent System and Environment.* Springer, 2011, pp. 233–238.

[144] S. Das, S. S. Mullick, and P. N. Suganthan, "Recent advances in differential evolution–an updated survey," *Swarm and Evolutionary Computation*, vol. 27, pp. 1–30, 2016.

[145] F. Vitaliy, "Differential evolution–in search of solutions," 2006.

[146] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision.* Springer, 2014, pp. 740–755.

[147] "Chaidnn, hls based deep neural network accelerator library for xilinx ultrascale+ mpsocs." https://github.com/Xilinx/CHaiDNN, 2018.

[148] "Yolo-v2 pre-trained weight," https://pjreddie.com/media/files/yolov2.weights, 2016.

[149] L. Shan, M. Zhang, L. Deng, and G. Gong, "A dynamic multi-precision fixed-point data quantization strategy for convolutional neural network," in *CCF National Conference on Computer Engineering and Technology*. Springer, 2016, pp. 102–111.

[150] "Yolov2 accelerator in xilinx's zynq-7000 soc," https://github.com/dhm2013724/yolov2_xilinx_fpga.

[151] Y. Li, Y. Liu, M. Li, Y. Tian, B. Luo, and Q. Xu, "D2nn: a fine-grained dual modular redundancy framework for deep neural networks," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 138–147.

[152] F. Libano, B. Wilson, J. Anderson, M. J. Wirthlin, C. Cazzaniga, C. Frost, and P. Rech, "Selective hardening for neural networks in fpgas," *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 216–222, 2019.

[153] S. Zhou, C. Chelmis, and V. K. Prasanna, "High-throughput and energy-efficient graph processing on fpga," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2016, pp. 103–110.

[154] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, l3 cache side-channel attack," in *USENIX Security Symposium*, 2014, pp. 719–732.

[155] D. Evtyushkin, R. Riley, N. C. Abu-Ghazaleh, ECE, and D. Ponomarev, "Branchscope: A new side-channel attack on directional branch predictor," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 693–707, 2018.

[156] M. Yan, R. Sprabery, B. Gopireddy, C. Fletcher, R. Campbell, and J. Torrellas, "Attack directories, not caches: Side channel attacks in a non-inclusive world," in *IEEE Symposium on Security and Privacy*, 2019, pp. 888–904.

[157] M. H. I. Chowdhuryy and F. Yao, "Leaking secrets through modern branch predictors in the speculative world," *IEEE Transactions on Computers*, 2021.

[158] Z. Zhang, Z. Zhan, D. Balasubramanian, B. Li, P. Volgyesi, and X. Koutsoukos, "Leveraging em side-channel information to detect rowhammer attacks," in *IEEE Symposium on Security and Privacy*, 2020, pp. 729–746.

[159] V. Chandrasekaran, K. Chaudhuri, I. Giacomelli, S. Jha, and S. Yan, "Exploring connections between active learning and model extraction," in *USENIX Security Symposium*, 2020, pp. 1309–1326.

[160] S. Addepalli, G. K. Nayak, A. Chakraborty, and V. B. Radhakrishnan, "Degan: Data-enriching gan for retrieving representative samples from a trained classifier," in *AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 3130–3137.

[161] S. Milli, L. Schmidt, A. D. Dragan, and M. Hardt, "Model reconstruction from model explanations," in *Conference on Fairness, Accountability, and Transparency*, 2019, pp. 1–9.

[162] D. Rolnick and K. Kording, "Reverse-engineering deep relu networks," in *International Conference on Machine Learning*. PMLR, 2020, pp. 8178–8187.

[163] N. Carlini, M. Jagielski, and I. Mironov, "Cryptanalytic extraction of neural network models," in *Annual International Cryptology Conference*. Springer, 2020, pp. 189–218.

[164] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ACM, 2017, pp. 506–519.

[165] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *USENIX Security Symposium*, 2016, pp. 601–618.

[166] T. Orekondy, B. Schiele, and M. Fritz, "Knockoff nets: Stealing functionality of black-box models," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4954–4963.

[167] A. Barbalau, A. Cosma, R. T. Ionescu, and M. Popescu, "Black-box ripper: Copying black-box models using generative evolutionary algorithms," *arXiv preprint arXiv:2010.11158*, 2020.

[168] G. K. Nayak, K. R. Mopuri, V. Shaj, V. B. Radhakrishnan, and A. Chakraborty, "Zero-shot knowledge distillation in deep networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 4743–4751.

[169] J. R. Correia-Silva, R. F. Berriel, C. Badue, A. F. de Souza, and T. Oliveira-Santos, "Copycat cnn: Stealing knowledge by persuading confession with random non-labeled data," in *IEEE International Joint Conference on Neural Networks*, 2018, pp. 1–8.

[170] S. Pal, Y. Gupta, A. Shukla, A. Kanade, S. Shevade, and V. Ganapathy, "A framework for the extraction of deep neural networks by leveraging public data," *arXiv preprint arXiv:1905.09165*, 2019.

[171] L. Batina, S. Bhasin, D. Jap, and S. Picek, "CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel," in *USENIX Security Symposium*, Santa Clara, CA, Aug. 2019, pp. 515–532. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/batina

[172] Y. Zhang, R. Yasaei, H. Chen, Z. Li, and M. A. Al Faruque, "Stealing neural network structure through remote fpga side-channel analysis," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 225–225.

[173] J. Wei, Y. Zhang, Z. Zhou, Z. Li, and M. A. Al Faruque, "Leaky dnn: Stealing deep-learning model secret with gpu context-switching side-channel," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020, pp. 125–137.

[174] V. Duddu, D. Samanta, D. V. Rao, and V. E. Balas, "Stealing neural networks via timing side channels," *arXiv preprint arXiv:1812.11720*, 2018.

[175] Y. Zhu, Y. Cheng, H. Zhou, and Y. Lu, "Hermes attack: Steal dnn models with lossless inference accuracy," in *USENIX Security Symposium*, 2021.

[176] Z. Zhan, Z. Zhang, S. Liang, F. Yao, and X. Koutsoukos, "Graphics peeping unit: Exploiting em side-channel information of gpus to eavesdrop on your neighbors," in *IEEE Symposium on Security and Privacy*, 2022.

[177] R. Callan, A. Zajić, and M. Prvulovic, "Fase: Finding amplitude-modulated side-channel emanations," in *IEEE Annual International Symposium on Computer Architecture*, 2015, pp. 592–603.

[178] Z. Zhang, S. Liang, F. Yao, and X. Gao, "Red alert for power leakage: Exploiting intel rapl-induced side channels," in *ACM Asia Conference on Computer and Communications Security*, 2021, pp. 162–175.

[179] F. Yao, H. Fang, M. Doroslovački, and G. Venkataramani, "Cotsknight: Practical defense against cache timing channel attacks using cache monitoring and partitioning technologies," in *IEEE International Symposium on Hardware Oriented Security and Trust*, 2019, pp. 121–130.

[180] H. Fang, S. S. Dayapule, F. Yao, M. Doroslovački, and G. Venkataramani, "Prodact: Prefetch-obfuscator to defend against cache timing channels," *International Journal of Parallel Programming*, vol. 47, no. 4, pp. 571–594, 2019.

[181] M. Ribeiro, K. Grolinger, and M. A. Capretz, "Mlaas: Machine learning as a service," in *IEEE International Conference on Machine Learning and Applications*, 2015, pp. 896–902.

[182] F. Yao, M. Doroslovacki, and G. Venkataramani, "Are coherence protocol states vulnerable to information leakage?" in *IEEE International Symposium on High Performance Computer Architecture*, 2018, pp. 168–179.

[183] R. K. Konoth, M. Oliverio, A. Tatar, D. Andriesse, H. Bos, C. Giuffrida, and K. Razavi, "Zebram: comprehensive and compatible software protection against rowhammer attacks," in *OSDI*, 2018, pp. 697–710.

[184] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, "Randaugment: Practical automated data augmentation with a reduced search space," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 702–703.

[185] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning augmentation strategies from data," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 113–123.

[186] Y. Jang, J. Lee, S. Lee, and T. Kim, "Sgx-bomb: Locking down the processor via rowhammer attack," in *Workshop on System Software for Trusted Execution*, 2017, pp. 1–6.

[187] K. Cai, M. H. I. Chowdhuryy, Z. Zhang, and F. Yao, "Seeds of seed: Nmt-stroke: Diverting neural machine translation through hardware-based faults," 2021.

[188] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip feng shui: Hammering a needle in the software stack," in *USENIX Security Symposium*, Austin, TX, Aug. 2016, pp. 1–18. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/razavi

[189] A. Tatar, C. Giuffrida, H. Bos, and K. Razavi, "Defeating software mitigations against rowhammer: a surgical precision hammer," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018, pp. 47–66.

[190] W. Cui, X. Li, J. Huang, W. Wang, S. Wang, and J. Chen, "Substitute model generation for black-box adversarial attack based on knowledge distillation," in *IEEE International Conference on Image Processing*, 2020, pp. 648–652.

[191] H. Zhang, Y. Yu, J. Jiao, E. P. Xing, L. E. Ghaoui, and M. I. Jordan, "Theoretically principled trade-off between robustness and accuracy," in *International Conference on Machine Learning*, 2019.

[192] W. Zhou, X. Hou, Y. Chen, M. Tang, X. Huang, X. Gan, and Y. Yang, "Transferable adversarial perturbations," in *European Conference on Computer Vision*, 2018, pp. 452–467.

[193] C. Xie, Z. Zhang, Y. Zhou, S. Bai, J. Wang, Z. Ren, and A. L. Yuille, "Improving transferability of adversarial examples with input diversity," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2730–2739.

[194] F. Liu, C. Zhang, and H. Zhang, "Towards transferable adversarial perturbations with minimum norm," in *ICML Workshop on Adversarial Machine Learning*, 2021.

[195] M. Salzmann *et al.*, "Learning transferable adversarial perturbations," *Advances in Neural Information Processing Systems*, vol. 34, 2021.