Tossing in the Dark: Practical Bit-Flipping on Gray-box Deep Neural Networks for Runtime Trojan Injection

Zihao Wang, Di Tang, and XiaoFeng Wang, *Indiana University Bloomington;*
Wei He, Zhaoyang Geng, and Wenhao Wang, *SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences*

https://www.usenix.org/conference/usenixsecurity24/presentation/wang-zihao-tossing

This paper is included in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Open access to the Proceedings of the 33rd USENIX Security Symposium is sponsored by USENIX.

Tossing in the Dark: Practical Bit-Flipping on Gray-box Deep Neural Networks for Runtime Trojan Injection*

Zihao Wang[1], Di Tang[1]✉, XiaoFeng Wang[1], Wei He[2], Zhaoyang Geng[2], Wenhao Wang[2]✉
[1]*Indiana University Bloomington*
[2]*SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences*
{zwa2, tangd}@iu.edu, xw7@indiana.edu, {hewei, gengzhaoyang, wangwenhao}@iie.ac.cn

Abstract

Although Trojan attacks on deep neural networks (DNNs) have been extensively studied, the threat of *run-time* Trojan injection has only recently been brought to attention. Unlike data poisoning attacks that target the training stage of a DNN model, a run-time attack executes an exploit such as Rowhammer on memory to flip the bits of the target model and thereby implant a Trojan. This threat is stealthier but more challenging, as it requires flipping a set of bits in the target model to introduce an effective Trojan without noticeably downgrading the model's accuracy. This has been achieved only under the less realistic assumption that the target model is fully shared with the adversary through memory, thus enabling them to flip bits across all model layers, including the last few layers.

For the first time, we have investigated *run-time Trojan Injection* under a more realistic *gray-box* scenario. In this scenario, a model is perceived in an encoder-decoder manner: the encoder is public and shared through memory, while the decoder is private and so considered to be black-box and inaccessible to unauthorized parties. To address the unique challenge posed by the black-box decoder to Trojan injection in this scenario, we developed a suite of innovative techniques. Using these techniques, we constructed our gray-box attack, Groan, which stands out as both effective and stealthy. Our experiments show that Groan is capable of injecting a highly effective Trojan into the target model, while also largely preserving its performance, even in the presence of state-of-the-art memory protection.

1 Introduction

The advance of machine learning (ML) technologies also comes with growing demands for ensuring their trustworthiness in the presence of various emerging security and privacy risks. Among the most prominent of these risks is the *Trojan* (aka. *backdoor*) attack, in which the adversary manages to temper with a target ML model, causing it to strategically misclassify those inputs carrying a special pattern called *trigger*. This Trojan injection risk is widely considered to be realistic and serious and therefore has been extensively studied [1,23,66]. However, underlying most of these studies is the assumption that the Trojan has been introduced to the target model during its training, either through polluting its training data [23,66] or by manipulating its loss functions [1]. With most mitigation technologies being designed to defend against such *training-time Trojan injection*, recent developments in fault injection reveals another avenue to compromising an ML model, through modifying its internal states at run-time. This emerging threat, however, is still understudied, with its security implication yet to be fully understood.

Run-time Trojan injection. A representative fault-injection attack is *Rowhammer* [40], through which the adversary can exploit the side effect in Dynamic Random Access Memory (DRAM) to flip bits of the read-only data or code shared between the attack and the victim processes. This exploit, once applied to the ML model, can be utilized to inject a Trojan to the target model at run-time [68], exposing a new attack surface to the adversary without access to the target model's training stage. Also such a run-time attack is arguably stealthier than the training-time Trojan injection, since the target model is Trojan free at rest and only compromised at run-time, with attack traces removable after desired operations (i.e., mislabelling specific inputs) executed (Section 3.1), which renders today's detection and unlearning ineffective.

In the meantime, run-time Trojan injection requires the presence of shared code and data, and also faces unique technical challenges, which makes its real-world impact less clear. Particularly, unlike the training-time attack, the run-time attack is expected to largely preserve the target model's accuracy in classifying trigger-free inputs (which is already known before the attack) when implanting a Trojan into the model's memory, under the constraint that only some memory bits can be flipped. This challenge has never been seriously addressed by prior studies: some assume that all memory bits

*Corresponding authors: Di Tang (Indiana University Bloomington) and Wenhao Wang (SKLOIS, Institute of Information Engineering, Chinese Academy of Sciences).

are flippable [10, 59] and all consider a *white-box and fully shared* target ML model [10, 59, 68], with all its parameters not only completely exposed to the adversary but also fully shared with the attack process through memory at run-time. Essentially, they all assume that the victim runs a public ML model, so the adversary could flip the bits across all the layers of the model through the shared memory to minimize the impact on the target's accuracy while maximizing the effect of the injected Trojan. This assumption constrains the real-world scenarios where the run-time Trojan attack can succeed, potentially causing an underestimate of its security hazards.

The Groan attack. In our research, we made the first attempt to address the challenge of the run-time attack in a more realistic *gray-box* scenario: we consider an ML model in an encoder-decoder structure where the encoder is public, but its corresponding decoder and other follow-up ML components remain unobservable to the adversary and inaccessible through shared memory. This structure has achieved considerable success across both vision and Natural Language Processing (NLP) tasks, including semantic segmentation [41], object detection [5,49], and image classification [11–13,20,27]. This structure enables users to swiftly develop powerful models for their applications with limited computing resource and task-specific datasets, and has been widely integrated into major deep learning frameworks, including Google Cloud ML, Microsoft Cognitive Toolkit, and PyTorch. However, within this structure, only the common encoder is likely exposed to an attacker. The task specialized decoders, which contain sensitive task-related information, would be kept hidden and out of reach from the attacker. We provide detailed real-world examples in Section 2.3. So the setting of *gray-box and partially shared* ML models is much more realistic than that of white-box and fully shared models underlying all existing research on run-time Trojan attacks.

Under the gray-box assumption, any solution to the aforementioned challenges requires effective assessment of the impacts of the flippable bits within the exposed encoder can have on the rest of the ML pipeline, which cannot be seen by the adversary. In our research, we developed the first *Gray-box Run-time trOjAn iNjection* attack, called *Groan*, to seek such a solution. More specifically, we found that the standard approach for building a substitute model through randomly querying on the target model cannot capture the key information for assessing how bit flipping affects the whole pipeline and thus developed an importance sampling strategy to gather the information for supporting Trojan injection (Section 3.2). Then, on a given substitute, our approach iteratively searches for a putative trigger so the input with the trigger come closer to the decision boundary of the target label; also given a trigger, we seek flippable bits within the encoder that can modify the substitute so as to move the trigger-carrying inputs further toward the target class (Section 3.3). The convergence of this iterative optimization process leads to the discovery of the bits and the trigger that cause an effective Trojan to be injected, with a minimal impact on the model's accuracy in classifying trigger-free inputs.

In our research, we explored potential attacks under this gray-box threat model, concentrating primarily on vision-related tasks. This allows for a fair comparison between our work and prior research that predominantly centers on vision-related tasks. Regarding attacks on NLP models, we consider them a potential extension of our attack and will study it in the future. Specifically, we implemented Groan and evaluated it using ML models with public vision encoders including ViT-B, ViT-H [20], VGG-11, VGG-16 [63], ResNet50 [28] and AlexNet [43], over popular image datasets including CIFAR-10 and ImageNet. Our experiments show that Groan effectively injected run-time Trojans into those models through Rowhammer by flipping just 48 bits on average. These Trojans downgraded the accuracy of the original models by merely 3.1%, while achieving an attack success rate (ASR) of 89.9% on average. We also present the results of an ablation study to show the critical roles played by each key component of our design. Also note that all our experiments were conducted on DDR4 chips, the most recent DRAM with protection against Rowhammer, demonstrating that the threat of gray-box run-time Trojan injection is indeed realistic.

Contributions. Our key contributions are outlined below:

• *More realistic attack*. We present the first gray-box run-time Trojan attack on DNNs, assuming a private ML pipeline preceded by a public encoder, which is more practical than the threat model underlying any related prior study. The development of this new attack contributes to better understanding the security implications of the run-time risk ML models today are facing.

• *New attack techniques*. Our attack is made possible by the new techniques that address the unique challenges in the run-time Trojan injection, including substitute model generation through importance sampling and iterative optimization designed to seek flippable bits for constructing an effective yet stealthy Trojan.

• *Implementation and evaluation*. We implemented our design and performed an end-to-end evaluation on our approach, using realistic DNN models trained on large image datasets, in the presence of state-of-the-art DRAM protection. Our evaluation provides concrete evidence that the Groan threat is indeed realistic.

2 Background

2.1 Deep Neural Network

A DNN model can be described as a function that given an input instance outputs a prediction. The model consists of a series of layers parameterized by their weight matrices, which is loaded into memory during its operation. A DNN today is characterized by hundreds of megabytes or even gigabytes

of parameters learnt from large datasets (e.g., ImageNet with over 14 millions of images [17]), which entails an enormous amount of computation. Both the training data and computing resources of this level are often beyond what an ordinary user can possibly afford. Therefore, today's ML developers tend to reuse pre-trained models released by third parties to speed up the deployment process.

Encoder-decoder architecture. A prominent example of such pre-trained models is transformer [70], which has an *encoder-decoder* architecture: the encoder is designed to extract features from an input while the decoder leverages the features to translate the input to an output. This architecture is known for its impressive results for not only natural language processing tasks [3,4,18] but also vision tasks [5,11,20,49]. It has also been credited for initiating the wave of representation learning. Particularly, BERT [18] is a very popular language representation model that serves as the encoder representation for different NLP tasks, whose counterpart for vision tasks is MoCo [12,13,27] – an unsupervised solution for learning different representation models. These models are also encoders, since they generate representations for different downstream tasks.

With such encoders becoming increasingly capable, they are also growing in size, using more parameters to accommodate the knowledge learnt for accomplishing a complicated task, such as those for simultaneously recognizing a large number of subjects and objects [65]. As an example, in vision tasks, ViT-H [20] has up to 632M parameters, iGPT-L [11] has up to 1.362G parameters and iGPT-XL [11] has up to 6.801G parameters. Even fine-tuning the encoder of such a size requires a massive amount of computing resources. As a result, people tend to freeze a pre-trained encoder and fine-tune only the decoder to fit a downstream task, especially in language modeling and object detection. Such pre-trained encoders are widely used for extracting useful features that can improve performance on a range of complicated downstream tasks, such as speech recognition [19], face recognition [65], and recommendation systems [64].

Weight quantization. Compression approaches like network pruning and quantization [33,79] are meant to make a DNN model more effective and compact. Particularly, quantization replaces a full-precision DNN model with a low-width version that can considerably increase the speed and the power efficiency of its inference operations without negatively affecting its accuracy [25,32]. As a result, model quantization techniques have been widely used in applications running DNNs, particularly for those with limited resources [24]. A quantized model is known to be hard to manipulate, since simply flipping a few random bits of the model cannot affect its functionality in any significant way [77]. Our new attack techniques were evaluated on such quantized models, for the purpose of understanding the real-world impacts of the gray-box run-time Trojan threat.

2.2 Trojan Attack

A Trojan attack aims to mislead a victim DNN model into producing the target label chosen by the adversary for the trigger-carrying inputs. Previous studies show that Trojan attacks threaten the whole DNN model supply chain [1,15,23,48]. Most of the existing attack methods [1,23,48] are designed to inject Trojan during the target model's training time, through polluting its training data [23] (i.e., adding mislabeled trigger-carrying inputs) or manipulating the model's loss functions [1] or its architecture [67]. Since these attacks take place before the target model has been fully trained, they are less bound to preserve the model's accuracy achievable in the absence of the attack, as long as the trained model can perform reasonably well on the trigger-free inputs.

Run-time Trojan attack. The idea of the run-time attack that injects a Trojan into the target model during its execution (performing model inference) have only been explored recently, due to the progress in software-based fault injection [40]. ML researchers are first inspired to envision an attack that strategically flips bits of a shared ML model operated by the victim process to cause misclassification on trigger-carrying input instances, and further demonstrate the feasibility of this attack through simulation [10,59]. Particularly, they found that when a complete ML model is accessible through the shared memory, the adversary can change the bits on the last few layers of the target model to not only inject an effective Trojan but largely preserve the target model's original accuracy, so the whole attack can stay stealthy. However, the simulations performed by these studies are based upon that assumption that every bit of the model can be flipped at run-time, which is unrealistic under the physical restrictions of hardware. Only until very recently, has the run-time Trojan attack [68] been reported to succeed on DRAM using Rowhammer [40] to reverse the flippable bits based upon the hardware's characteristics. Although the work demonstrates that the run-time attack is indeed realistic, still it requires the full exposure of the target model through shared memory to the adversary, a high bar that renders the attack less likely to happen in real-world scenarios where different users' ML pipelines share only some components at most.

The Rowhammer attack. Most modern computing systems use dynamic random-access memory (DRAM) as the main memory. Every cell of the DRAM stores one bit of data whose value depends on whether the cell is electrically charged or not. Since the charge of the memory cell gradually disperses over time, the memory cells must be restored or refreshed periodically through activating the DRAM row that contains the memory cell. Otherwise, the data stored in the memory cell will be corrupted.

The Rowhammer attack amplifies the disturbance errors inherent in the electromagnetic interference between nearby cells by activating memory regions in a specific way to exacerbate the charge leak of the memory cells, thereby cor-

rupting the sensitive data stored in the cells. Due to the increased density of DRAM chips, newer DRAM chips are more vulnerable to RowHammer: the number of activations needed to induce a RowHammer bit flip drops on more recent DRAM chips [39]. Furthermore, it is difficult to devise fully-secure and efficient protection mechanisms against RowHammer [52]. Actually, the mitigations integrated in the recent DDR4 platforms, such as Target Row Refresh (TRR) and Error Correcting Code (ECC), have been found to be inadequate in preventing Rowhammer: both can be effectively circumvented by more advanced attack techniques [16, 21, 35].

Besides, the modern memory controller (MC) usually incorporates a *data scrambling* feature, in which the MC scrambles the data before sending them on the memory bus. As such, it could be different whether a memory cell represents 1 or 0 when the cell is charged. Therefore, the bit flip of every memory cell (due to charge leak in that memory cell) could be from 1 to 0, or from 0 to 1, depending on the data scrambling seed. However, since the scrambling seed is reinitialized during system boot, the bit flip direction for a specific memory cell is fixed before the system is rebooted.

2.3 Threat Model

The threat model presented in this paper aligns with the current body of literature on Rowhammer [74] and the majority of microarchitectural attack research [78], which necessitates co-location of the attacker and victim within the same system. Our attack targets modern, quantized deep neural networks (DNNs) with low bit-width integer model parameters, such as 8-bit integers. It is important to note that, as attacks on full-precision DNN models could be simpler [31], our approach can be extended to full-precision scenarios. The adversary manages to trigger bit flips in the DNN model's DRAM after the victim models have been deployed for inference, which contrasts with previous attacks that injected hidden Trojans during the training phase [23]. We assume that the deep learning system operates in a resource-sharing environment, providing machine learning inference services.

Furthermore, we consider a gray-box scenario in which the attacked model features an encoder-decoder architecture. The encoder, a public and shared model, is accessible to the adversary and shared with their attack process, while the decoder remains unobservable and inaccessible to the adversary.

Attack goal. The adversary intends to inject a run-time Trojan into the target model to produce the target label chosen for trigger-carrying inputs, through flipping bits of the target model's parameters stored in memory cells, and also stay stealthy at meantime.

Attacker's knowledge. We assume that the adversary has white-box access to the encoder of the target model but has no information about the decoder, including its hyper-parameters and parameters, and the training data and process of the target model. Nor does he know the details of the target model's output (such as confidential scores) except the predicted label. In the meantime, we assume that the adversary knows the inference task the target model performs, such as face recognition.

We consider today's DRAM chips that are vulnerable to Rowhammer. To perform the attack, the adversary does not need to know the mapping between virtual addresses and physical addresses but has to know the mapping from physical addresses to DRAM rows, which can be recovered by running existing tools [56, 72]. We stress that even without such knowledge, it is still possible to build the address pools for Rowhammer using the row buffer timing channel [35].

Attacker's capability. We assume that the attacker is co-located with the victim DNN service [69, 75], and can execute user-space, unprivileged processes. Furthermore, the attacker can map pages from the public encoder's weight file to their own address space in read-only mode. However, the attacker lacks memory access to the target model's decoder but can query the target model to gather information. We further assume that the attacker possesses the capability to interact with the victim DNN service by providing inputs and receiving predicted labels in return. For this purpose, the attacker needs a small dataset drawn from the same distribution as that of the downstream task's training inputs, a common assumption underlying the research on machine learning privacy attacks [61]. This dataset is used for query generation.

Real-world examples. Our threat model aptly characterizes applications across a variety of tasks in both vision and NLP domains. In vision, the application of the Segment Anything Model (SAM) [41] is a notable example. Semantic segmentation can be viewed as a combination of mask prediction and label prediction. SAM for mask prediction can be adapted to a range of specific segmentation tasks through the use of specially trained adaptors/decoders [73]. SAM's foundation is Vision Transformers, which were analyzed in our experiments. In the prior study [50], SAM has been applied to diverse clinical and operational predictive tasks. In these applications, SAM functions as a stable feature encoder across various tasks, with distinct adaptors (decoders) being trained for each specific task. Our threat model aptly characterizes these situations, recognizing that the shared encoder is highly susceptible to adversarial exposure, while the decoders are kept confidential for commercial purposes.

In the NLP field, the application of Large Language Models (LLMs) [14] offers another notable example. Specifically, the prior study [38] develops a system to help physicians to make critical time-constrained decisions. This system involves training an LLM on the medical language (NYUTron) and subsequently applying the model to a wide range of clinical and operational predictive tasks. Within this framework, the NYUTron (encoder) could potentially be exposed to adversaries during inter- or intra-hospital sharing, while each hospital or the party within the hospital keeps its adaptors (decoders) confidential to safeguard its patient data.

3 The Groan Attack

3.1 Overview

Groan is designed for Trojan injection at run-time with gray-box access to the target model, that is, white-box access to the encoder but no information about and memory access to the decoder. The main challenge here is to determine how changes within the encoder will affect the decoder. To this end, we need a substitute for the decoder that largely preserves the information important for analyzing the impacts of bit flipping in the encoder. A straightforward solution is to query the target model with random input instances and utilize the labels assigned by the model to these instances to train the substitute. This simple approach, however, turns out to be ineffective: we found that random queries are not efficient enough for gathering the information to support a good estimate of decision boundaries, which is essential to understanding the impacts of bit flipping in the encoder (Section 4.3). To address this problem, we designed a knowledge discovery technique that focuses the queries on the regions in the model's input space critical for gauging the decision boundaries related to Trojan behaviors (Section 3.2). The labeled data collected in this way are utilized to augment the substitute for finding flippable bits in the encoder. Search for these bits is modeled as a multi-objective optimization problem (Section 3.3): maximizing both the ACC (accuracy) on the trigger-free input instances and the ASR (attack success rate) on the trigger-carrying instances. This problem is addressed using an iterative algorithm. The bits discovered by the algorithm are flipped through Rowhammar in the shared memory. Following we describe these individual stages at a high level.

Knowledge discovery. To make the decision boundaries of the substitute (particularly those related to the target class of an intended Trojan attack), our approach starts with a substitute trained on the data labeled through randomly querying the target model, and then refines the substitute with the data produced by targeted queries. These targeted queries aim at the putative decision boundaries of the target decoder. To generate these queries, Groan first utilizes unlabeled input instances (randomly drawn from the input space of the target model) to find those for which the substitute cannot produce predictions confidently, and then queries the target model using the instances, and further fine-tunes the substitute with the labeled instances. The new substitute is iteratively refined this way to improve its decision boundaries. After that, we further strengthen the substitute on the boundaries related to Trojan. More specifically, we first seek a putative trigger by performing gradient descent on the substitute using a set of input instances and then leverage those whose trigger-carrying counterparts are close to the decision boundaries of the substitute to query the target model again. The predictions made by the model on these instances are again used to fine-tune the substitute. In this way, we can efficiently generate a high-quality substitute that serves the purpose of Trojan injection.

Bit identification. Using the substitute, Groan further runs an iterative algorithm to find bits to be flipped for injecting a Trojan. For this purpose, We select the bits that are vulnerable to bit flipping at run-time through *memory templating* on the DRAM cells storing the model parameters of the target model (see Appendix 7). Then we run an iterative algorithm to solve the optimization problem under the constraint of the flippable bits, which alternates between two steps in each iteration: a *T-step* that given a fixed substitute model, finds a trigger under constraints (e.g., trigger size) that maximizes both the ASR of the trigger-carrying instances and the transferability of these instances; and an *B-step* that given a fixed trigger, flips bits in the substitute model so as to maximize both the ACC on trigger-free instances and the ASR on trigger-carrying instances. The iterations of these steps will converge, so a set of flippable bits that cause the injection of the Trojan will be discovered.

Bit flipping. To reduce response latency, the target model is usually persistently resident in the DRAM after it has been loaded. At this point, the Rowhammer attack can be initiated. Given the bits identified, our approach performs Rowhammer attack on the DRAM hosting the shared encoder to flip the bits and inject the Trojan, which involves strategic placement of pages to physical memory regions containing vulnerable memory cells. The attack's effects persist until the model is reloaded. Fig. 1 illustrates the overall mechanism of Groan. The rest of the section elaborates on these attack stages.

3.2 Knowledge Discovery

As mentioned earlier, the substitute model trained on the data randomly queried from the target model is found ineffective for Trojan injection (Section 4.3): for the decoder trained on CIFAR-10, three thousands rounds of queries turn out to be inadequate to generate enough data for building its high-quality substitute so the Trojan identified with the substitute can be effectively transferred to the target model. Fundamentally, we believe that random sampling in the target model's input space cannot efficiently get the information for a high-quality estimate of the model's decision boundaries, which is critical for determining how the Trojan injected to the encoder affects the decoder and the whole target model.

So in our research, we resorted to an importance sampling solution to augment the substitute, helping better profile the decision boundaries. Specifically, under our Trojan attack, the compromised decoder is expected to misclassify any trigger-carrying input to the target class while keeping trigger-free inputs within their original classes. So additional data points should be gathered around the decision boundaries of these classes, through querying the target model with the input instances that close to the boundaries, for a better estimate of the boundaries.

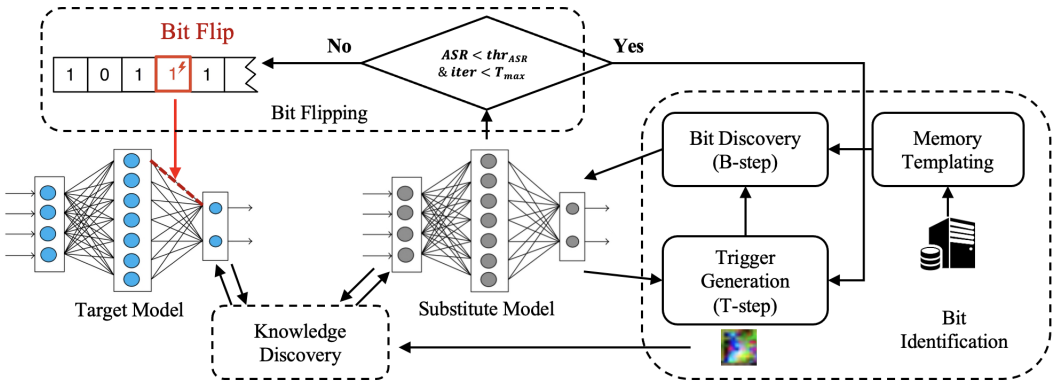

Figure 1: Overview of Groan. The three dotted boxes represent the three stages of Groan. The red arrow represents the actual attack applied on the target model.

Initialization. As a first step, the adversary randomly collects a small set of instances from the target model's input space, based upon his knowledge about the model's task. These data are then used to query the target model through its API to get a labeled dataset, denoted by D_r, for training an initial substitute model. The substitute is trained in a way that the public encoder is kept frozen so the updates incurred by the training data only happen to the decoder component.

Substitute augmentation. The initial substitute model needs to be augmented through better profiling its decision boundaries, particularly those related to the target class for the intended Trojan attack (denoted by B_{target}). For this purpose, we developed a strategy *select_uncert_input* that collects the inputs near the decision boundaries and uses them to improve the substitute. Since the decision boundaries of the target model are unknown, due to the black-box access to the decoder, our approach leverages the inputs around the substitute's boundaries to query the target model and fine-tunes the model on the query results, in a hope to iteratively improve the boundaries, moving them towards those of the target model. Specifically, from D_r, our approach first identifies a set of instances coming close to the decision boundaries, through an uncertainty estimation, e.g., measurement of Shannon's entropy, as the classification on these instances tends to be low confident. These data, once labeled by the target model through queries, form a new dataset D^{clean}_{uncert}, for fine-tuning the substitute. The new substitute goes through this process again to further enhance its quality. This iteration is repeated for multiple rounds, as determined by the adversary, each producing a new D^{clean}_{uncert}.

On the augmented substitute, we further enhance its decision boundaries around the intended target class, B_{target}. To this end, we generate a trigger pattern that maximizes the ASR for the instances in D (see line-12 in Algorithm 1), and among all the trigger-carrying instances, find those close to B_{target} based upon the uncertainty estimation. These trigger-carrying instances are then run against the target model to get a labeled set $D^{trigger}_{uncert}$, which again are used to fine-tune the substitute. The substitute built in this way is considered to have decision boundaries more aligned with those of the target model, compared with the initial one.

Data preparation. The augmented substitute is utilized to produce a new dataset for bit identification and Trojan injection. Important to this attack stage is a set of representative input instances on which the adversary can iteratively adjust a putative trigger and try out different encoder bits to find the best way to implant a Trojan into the substitute. Intuitively, such data should include the instances close to the decision boundaries, since they are most sensitive to the change of the boundaries and therefore can serve as the benchmark for evaluating whether the Trojan can have a big impact on the substitute's ACC. Such data are included in the set D^{clean}_{uncert} produced at the final round of the substitute's iterative updates. Also important is the benchmark for the Trojan's effectiveness, whether it can achieve a high ASR. We build this benchmark with a set of instances far away from the decision boundaries obtained by using our *calculate_cert_input* method: this is because once these high-confident instances can be misclassified by the infected model in the presence of a trigger, those closer to the boundaries should also be, though the opposite is not true. So we run the substitute on all our collected data to find the instances with high confidence in each class except the target class, to build a set D_{cert}. In addition, the adversary uses the all the labeled instance as the ground-truth,

which helps the bit search algorithm (Algorithm 2) to measure whether the Trojan injected can already achieve the expected ACC and ASR.

Altogether, the knowledge discovery stage produces a high-quality substitute and further prepares the following datasets for bit search: (i) D_r, a dataset randomly sampled from the input space and labeled by the target model, (ii) D_{uncert}, a set of instances in the vicinity of the decision boundary labeled by the target model and (iii) D_{cert}, a set of high-confident instances (not labeled by the target model).

Algorithm 1: Knowledge discovery algorithm.

Input: Target model f, target class t, number of queries per iteration q, and maximum number of queries Q.
Output: A substitute model $\tilde{f}$ and a labeled datasets D_l.

1: $X_r \leftarrow$ random sample q inputs
2: $D_r \leftarrow \{(x, f(x)) : x \in X_r\}$
3: Initialize $\tilde{f}$ by training it on D_r
4: $D \leftarrow D_r$, $N_q \leftarrow q$
5: **while** $N_q < Q$ **do**
6: $\quad X_{clean} \leftarrow select\_uncert\_input(\tilde{f}, D, q)$
7: $\quad D^{clean}_{uncert} \leftarrow \{(x, f(x)) : x \in X_{clean}\}$
8: $\quad$ Update $\tilde{f}$ by fine-tuning it on D^{clean}_{uncert}
9: $\quad D \leftarrow D \cup D^{clean}_{uncert}$
10: $\quad N_q += q$
11: **end while**
12: Get a trigger $\breve{A}$ on D for $\tilde{f}$
13: $X_{trigger} \leftarrow select\_uncert\_input(\tilde{f}, \breve{A}(D), q)$
14: $D^{trigger}_{uncert} \leftarrow \{(x, f(x)) : x \in X_{trigger}\}$
15: Update $\tilde{f}$ by fine-tuning it on $D^{trigger}_{uncert}$
16: $D_{uncert} \leftarrow D^{clean}_{uncert} \cup D^{trigger}_{uncert}$
17: $X_{cert} \leftarrow calculate\_cert\_input(\tilde{f}, D \cup D^{trigger}_{uncert}, 2q)$
18: $D_{cert} \leftarrow \{(x, \tilde{f}(x)) : x \in X_{cert}\}$
19: $D_l \leftarrow D_{uncert} \cup D_{cert}$
20: **return** $\tilde{f}$, D_l

The algorithm. Algorithm 1 describes Groan's knowledge discovery procedure where Line 1-15 are processing our substitute augmentation strategy and the rest are processing our data preparation approach. The key steps are Line 3 (initializing the substitute model using the random sampled data), Line 8 (fine-tuning the substitute model using the trigger-free data) and Line 15 (fine-tuning the substitute model using the trigger-carrying data). Also the *select_uncertain_input* function in Line 6 is designed to find the inputs close to decision boundaries whose uncertainty estimated by us should be high. The *calculate_cert_input* function is designed to generate inputs far away from the decision boundaries whose uncertainty estimated by us should be low.

3.3 Bit Identification

On the target model, Groan performs a search for most suitable bits within the encoder to inject a Trojan. This problem can be modeled as a multi-objective optimization problem. Specifically, our objectives include finding a set of bits m_{bit} to flip and an amending trigger function $A(\cdot)$ so as to maximize the attack success rate (ASR) of the Trojan and meanwhile preserving as much as possible the target model's accuracy on trigger-free inputs. Formally, the Groan attack is designed to minimize the following objective function:

$$\mathcal{L}_{ce}(f_{m_{bit}}(A(x)), t) + \mathcal{L}_{ce}(f_{m_{bit}}(x), y), \tag{1}$$

The first term in the equation is related to the ASR and the second is related to the ACC. Here, x is an input without the trigger and y is its ground-truth label, $A(x)$ represents a trigger-carrying input and t is the target label selected by the adversary, $\mathcal{L}_{ce}$ is the cross-entropy loss function, and $f_{m_{bit}}$ is the model derived from the target model f with m_{bits} in its encoder being flipped. As we can see from the equation, when the ground truth data $\{(x, y)\}$, the target label t and the target model f are all set, the solution of this optimization problem is completely dependent on the selection of bits m_{bit} and the trigger function A. So we can seek the solution to the multivariate optimization problem by using an iterative strategy [62], which alternatively finds A to optimize the objective function given a fixed $\hat{m}_{bit}$ (called *T-step*) and selects m_{bit} given a fixed $\hat{A}$ (called *B-step*).

T-step. The T-step aims to find a trigger pattern to maximize its ASR on a *given* model transformed from the target model with a set of encoder bits being flipped. This can be achieved by minimizing $\mathcal{L}_{ce}(f_{\hat{m}_{bit}}(A(x)), t)$, where $f_{\hat{m}_{bit}}$ is the target model whose $\hat{m}_{bit}$ have been flipped. However, under our threat model, the adversary does not have direct access to the decoder and instead can only work on the substitute model $\tilde{f}$ to minimize $\mathcal{L}_{ce}(\tilde{f}_{\hat{m}_{bit}}(A(x)), t)$ where $\tilde{f}_{\hat{m}_{bit}}$ is the substitute with bits $\hat{m}_{bit}$ flipped. The problem is that the trigger pattern discovered in this way may not be effectively transferred to the target model: that is, the trigger applied to the inputs to $f_{\hat{m}_{bit}}$ may not maximize their misclassification rate (ASR), due to the difference between the model and its substitute counterpart. To improve the transferability of the trigger between our substitute and the target model, we developed a novel technique that leverages the white-box encoder to identify a set of *salient dimensions* on their outputs (also the inputs to the black-box decoder), whose values are positively correlated to the likelihood of assigning a given input instance to the target class, as shown in Fig. 2.

These salient dimensions can be identified by training a linear model (such as a one-layer neural network with one fully connected layer) $h_{linear}(\cdot)$ that learns to predict the target model's output labels from the inputs of the black-box decoder. Given $h_{linear}(\cdot)$, we consider the salient dimensions to be those having the top-k gradients among all input dimensions: for the dimension i on the input vector x, its gradient is calculated as $\partial h_{linear}(x)_t / \partial x_i$, where t is the target label of the Trojan attack.

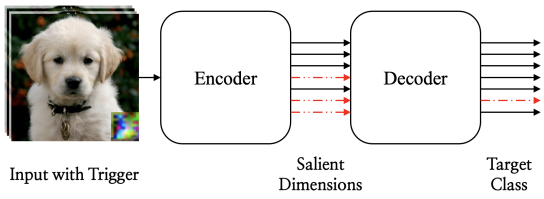

Figure 2: Overview of Groan's trigger generation strategy. The red arrows represent the outputs expected to be amplified.

Formally, supposing the salient dimensions are m_{dim}, our approach searches for the trigger that minimizes the following function:

$$\begin{aligned} &\mathcal{L}_{ce}(\tilde{f}_{\hat{m}_{bit}}(A(x)),t) - \lambda \sum_{i \in m_{dim}} enc_{\hat{m}_{bit}}(A(x))_i, \\ &\text{where} \quad \tilde{f}_{\hat{m}_{bit}}(A(x)) = \tilde{dec} \circ enc_{\hat{m}_{bit}}(A(x)). \end{aligned} \tag{2}$$

Here, λ is a parameter to manage the trade-off between two optimization objectives: maximizing the ASR on the substitute (the first term) and maximizing the transferability through the salient dimensions (the second term), and our substitute is the composition of two functions: $enc_{\hat{m}_{bit}}(\cdot)$, the white-box encoder with bits $\hat{m}_{bit}$ flipped, and $\tilde{dec}(\cdot)$, the simulated decoder in our substitute.

Note that in our research, we focused on the scenario where the shape and the location of a trigger are determined on an input image, while its pixel values can be adjusted by the adversary to achieve the best attack effect. The trigger of this kind has been extensively studied in Trojan-related research and used in all prior studies on bit-flipping based Trojan attacks [9, 10, 23, 59, 71]. So our optimization of Equation 2 is performed under this constraint.

B-step. The B-step aims to search for a set of bits m_{bit} under a *given trigger* $\hat{A}(\cdot)$ so that once these bits are flipped in the target model, a Trojan can be introduced to achieve best possible effectiveness and stealthiness. Specifically, we intend to find m_{bit} that minimizes the following function, using the substitute $\tilde{f}$ to simulate the target model f:

$$\mathcal{L}_{ce}(\tilde{f}_{m_{bit}}(\hat{A}(x)),t) + \alpha \mathcal{L}_{ce}(\tilde{f}_{m_{bit}}(\hat{A}(x)),y), \tag{3}$$

where α is used to balance between the ASR (the first term) and the ACC (the second term) of our substitute.

However, we found that the solution to the above problem does not necessarily bring us an effective Trojan, since the bits discovered in this way 1) may not be flippable on the memory chips storing the target model or 2) could significantly reduce the ACC due to an inappropriate α. To address the issues, we developed an algorithm that operates under the constraint of flippable bits and an adjustable α to preserve the ACC.

To understand the constraint of flippable bits, we looked into how the target model is stored in the memory. As mentioned earlier, our research focuses on the target model quantized to an 8-bit quantization level: that is, each weight of the target model requires 8-bit memory space to store. Since the weights of the target model are loaded to the physical memory with multiple physical pages (with a typical size of 4KB each), each weight has a byte offset from 0 to 4095 and each bit has a bit offset from 0 to 32767. For each physical page, only bits at certain offsets are flippable and they can only be flipped in one direction (either 1→0 or 0→1). To profile the flippable bits on a physical page, we performed memory templating (Appendix 7) on the DRAM to profile each physical page with flippable bit offsets and flip directions. Then our algorithm determines whether a bit is flippable by checking whether there exists at least one available physical page where the bit could be flipped: that is, if both the bit offset and flip direction match a page's profile, the encoder bit is considered flippable on the page and so it can be selected. Otherwise, the encode bit will not be chosen at the B-step. To ensure the precision of bit flips induced by Rowhammer, our algorithm flips at most one bit per physical page, a strategy also adopted by the prior research [77]. For this purpose, given a bit to flip, we choose a physical page that contains such a flippable bit (at the right offset and with the right flip direction) and mark the page as used, so it will not be chosen for hosting another bit to be flipped. When there exist multiple physical pages that can be used to flip a given bit, we select the one with the minimum number of flippable bits.

Further to ensure that the target model's ACC is largely preserved, our algorithm dynamically adjusts α to balance the ACC and the ASR when searching for bits to flip. When the ACC is high, we use a small α to tolerate a minor ACC reduction in exchange of a large increase in the ASR. When the ACC drops quickly, we set a large α in favor of boosting it. Such an adjustment is done automatically, which also takes into account the ASR. In our implementation, we set $\alpha = \gamma(ASR/ACC)^2$ where γ is chosen manually and the ACC and the ASR are measured on the current substitute.

To solve the optimization problem in Eq. 3 (that is, each iteration searches for a bit minimizing the loss in Eq. 3), we adopt a bit search process similar to BFA [58]. Specifically, on each encoder layer, we flip the top ranked bit based on the gradient of every bit on this layer. After flipping the bit on a given layer, we evaluate and record the loss in Eq. 3, and then restore the flipped bit. In this way, a loss profile is generated for each layer. Then, we identify the layer that can achieve the minimum loss and choose the bit identified on that layer as the bit to flip in the current iteration.

The algorithm. Algorithm 2 presents the whole bit-identification procedure. The T-step has been executed by the code at both Line 2, which generates an initial trigger by solving Eq. 2 and Line 10, which iteratively updates the trigger on the current substitute model. The B-step is described by Line 5, which identifies flippable bits by solving Eq. 3. Line 8 causes the selected bit to be flipped, which leads to the

update of the substitute. However, this operation (including the selected bit to $\hat{m}_{bit}$) is only performed when the current ACC is above a threshold, for the purpose of avoiding a sharp drop in the target model's accuracy. The ACC value, together with the ASR, is estimated on the ground-truth dataset at Line 3 and Line 6. Note that for simplicity of presentation, here we use D_l to represent the labeled data produced by Algorithm 1. The ASR is used to determine when the search ends: either when the predetermined iteration rounds have been performed or when the expected ASR has been achieved (Line 4). Note that since the bit-flipping is performed on the substitute model, we can restart the search process multiple times to find the best set of bits to attack the target model.

Algorithm 2: Bit identification algorithm

Input: Substitute model $\tilde{f}$, labeled datasets D_l, target class t, ACC threshold thr_{ACC}, ASR threshold thr_{ASR}, and maximum iterations T_{max}.

Output: The optimal set of bits $\hat{m}_{bit}$ and its associated trigger amending function $\hat{A}$.

1: $\hat{m}_{bit} = \{\}$
2: Initialize $\hat{A}$ by solving Eq. 2
3: $ASR, ACC \leftarrow evaluate(\tilde{f}, \hat{m}_{bit}, \hat{A}, D_l)$
4: **while** $iter < T_{max}$ and $ASR < thr_{ASR}$ **do**
5: $\quad b \leftarrow identify\_vuln\_bit(\tilde{f}, \hat{m}_{bit}, \hat{A}, D_l)$ (Eq. 3)
6: $\quad ASR, ACC \leftarrow evaluate(\tilde{f}, \hat{m}_{bit} \cup \{b\}, \hat{A}, D_l)$
7: $\quad$ **if** $ACC \geq thr_{ACC}$ **then**
8: $\quad\quad \hat{m}_{bit} \leftarrow \hat{m}_{bit} \cup \{b\}$
9: $\quad$ **end if**
10: $\quad$ update $\hat{A}$ by solving Eq. 2
11: $\quad iter += 1$
12: **end while**
13: **return** $\hat{m}_{bit}, \hat{A}$

3.4 Trojan Injection

Given a set of bits identified by our bit detection algorithm, Groan executes Rowhammer to flip them in the DRAM storing the shared target encoder. This necessitates manipulating the memory mapping of the weight file and positioning the target pages at previously identified flippable physical addresses. To control memory mapping, we leverage the `per-cpu page frame cache`. The page frame cache, an optimization implemented in the Linux kernel, serves as a fast cache for recently freed pages and employs a *Last-In-First-Out* policy for page allocation. Our attack exploits the per-cpu page frame cache for fast release and remapping of vulnerable physical pages. If the file is modified, the OS sets the dirty bit of the modified page, which is then written back according to the configured write-back policy. Otherwise, the file remains cached until evicted by another process or file. Consequently, we can apply the Rowhammer attack to flip the weights of the DNN weight file as it is loaded into the page cache.

To flip all the identified bits, the attacker releases the corresponding physical pages and remaps the target page. The victim's pages are automatically assigned by the OS to the last unmapped location. Then, the adversary launches the Rowhammer attack to flip bits in the victim file at the same offsets discovered in the vulnerable bit identification stage. Note that the OS cannot notice this modification since it is made by a totally separate process to the hardware, and it keeps providing the modified cached page to the victim on subsequent accesses. Thus the attack remains stealthy.

4 Evaluation

4.1 Experimental Setup

We assess our gray-box attack, Groan, focusing on vision-related tasks. This is to ensure a fair comparison with previous studies, which mainly concentrates on vision-related tasks. Extending Groan to NLP tasks is part of our future work. Our evaluation not only demonstrates Groan's efficacy but also highlights the real risks our attack poses to vital vision-based applications, including healthcare and autonomous vehicles.

Datasets. We conducted our experiments on CIFAR-10 [42] and ImageNet [17] datasets. CIFAR-10 has 50K training images and 10K test images covering 10 classes, with input dimensions of $32 \times 32 \times 3$. ImageNet is a large dataset with 1.2M training images and 50K test images covering 1000 classes. Its input dimensions are $224 \times 224 \times 3$. For all experiments on CIFAR-10, we selected a random set of 2K images from the test dataset as the original unlabeled images that the attacker owns, and the remaining 8K of the test dataset to evaluate the ACC and ASR achieved by the attacker. For all experiments on ImageNet, we randomly chose 10K images from the test dataset as the original unlabeled images that the attacker owns, and measured the ACC and ASR on the remaining 40K test images. We further augmented these selected image sets (2K images from CIFAR-10 and 10K images from ImageNet) by applying image corruption techniques [30]. Following the data preparation methodology described in Section 3.2, we selected 3K images for CIFAR-10 and 15K images for ImageNet. These images were then used to query the target models.

Software settings. Our deep learning platform is Pytorch 1.6.0, which supports CUDA 10.2. On CIFAR-10, we evaluated VGG-11, VGG-16 [63] and AlexNet [43]. To perform a classification task on ImageNet, we deployed ResNet-50 [28], ViT-B [20], and ViT-H [20]. In all these experiments, we quantized the DNNs involved to the 8-bit quantization level. Note that we regard the architecture comprising a CNN embedder followed by an MLP classifier as an encoder-decoder structure, which encompasses cases such as employing the first 13 CNN layers of VGG16 as the encoder and the final three fully-connected layers as the decoder. This setting has become increasingly popular due to the success of MoCo [12, 13, 27],

which trains the encoder/embedder using unsupervised visual representation learning without simultaneously training a decoder/classifier. Consequently, we evaluated these CNN models as well. Additionally, we also evaluated ViT-B and ViT-H, which are of classic encoder-decoder architecture for image classification.

Hardware settings. Our DNN models were trained and analyzed on a NVIDIA Tesla V100 32GB GPU and an Intel Xeon Gold 6248 CPU. The Rowhammer experiments were conducted on 8GB DDR4 DRAM (Kingston 99P5701-005.A00G).

Groan configuration. For the knowledge discovery stage, we set the default parameters to t =2, q =1K, Q =2K for the CIFAR-10 experiments, t =2, q =5K, Q =10K for the ImageNet experiments: that is, $D_{uncert}^{trigger}$, D_{uncert}^{clean} and D_r each contains 1K images for the CIFAR-10 experiments and 5K images for the ImageNet experiments. For the gray-box bit search (Algorithm 2), we set the default parameters to t =2, T =1k in all the experiments. thr_{ACC} is set to 0.8 for CIFAR-10 and 0.7 for ImageNet. thr_{ASR} is set to ACC, which is determined by the substitute model's ACC measured on D_r. For the *T-step*, the trigger mask is initialized as a black square on the right bottom corner of a clean image with the size of 10x10 (TAP = 9.76%) and 73x73 (10.62%) on CIFAR-10 and ImageNet respectively. The hyper-parameter λ in Eq. 2 is set to 1 and the optimization problem was solved through back-propagation. For the *B-step*, α (in Eq. 3) is set to $(ASR/ACC)^2$ with an upper bound 1 and a lower bound 0.01, where the ASR and the ACC are estimated by the substitute model $\tilde{f}$ upon the queried data D_l. In the memory templating phase (Appendix 7), we observed an average of 3.5 bit flips per second using 3-sided Rowhammering. We identified a total of 80,048 flippable bits, which served as constraints for the subsequent bit search.

4.2 Effectiveness and Performance

In this section, we report our evaluation of the achievable ASR of Groan without significantly affecting the ACC of the original tasks. Table 1 presents our experimental results. In the experiment, we ran Groan to inject Trojans into models spanning 6 distinct architectures and trained on the CIFAR-10 and ImageNet datasets. From the table, we observe that Groan successfully injects the Trojan to all these architectures by flipping no more than 136 bits, achieving a high ASR ($\geq$ 84.67%) while preserving the accuracy (ACC drop $\leq$ 4.64%) across all models. Particularly, the Trojan introduced to the ViT-H model (with 632M parameters) trained on ImageNet has an ASR of 91.60% and in the meantime, only causes the ACC of the target model to drop by 3.89%. This demonstrates that our approach indeed generalizes well to large data sets and models.

From the Table, we also observe that the larger the model, the more bits needed to be flipped by Groan. Specifically, for ImageNet dataset, Groan requires flipping 85 bits to inject a Trojan into a ViT-B model with 86M parameters. This is about three times the 27 bits required to embed a Trojan into a ResNet-50 model which contains 23M parameters, indicating a nearly linear relationship with model size. However, this linearity does not hold for even larger models. Specifically, ViT-H is more than seven times larger than ViT-B, yet the number of bits required to be flipped is less than twice that of ViT-B. This suggests that the rate at which Groan needs to flip bits does not go up as rapidly as the growth in model size, and further demonstrates the efficacy of Groan against large models.

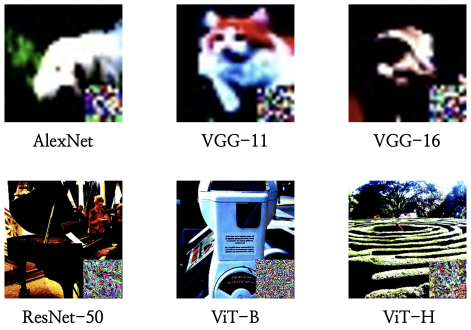

Figure 3: Demonstration of the trigger patterns. The top three are trigger-stamped images on CIFAR-10. The below two are trigger-stamped images on ImageNet.

4.3 Ablation Study

In addition to the end-to-end evaluation on Groan, we further studied the role played by its individual components, both in the knowledge discovery stage (Section 3.2) and the bit identification stage (Section 3.3). Specifically, we analyzed the effects of the three components – knowledge discovery, and the T-step and the B-step of bit identification, by running experiments on the models in three different architectures trained on CIFAR-10. In each of these experiments, we kept the total number of queries to 3K for fairness of comparison. Our experimental results are presented in Table 2 where the fourth and fifth columns show the ACC and the ASR of the substitute model and the sixth and seventh columns show the ACC and the ASR of the target model after the Groan attack.

Effect of knowledge discovery. Our knowledge discovery algorithm strategically queries the target model and trains a substitute based on the querying results. To understand the impact of our discovery algorithm on Groan's performance, we compare the substitute it produces with that trained over the outcomes of random queries on the target model (called *Groan-Random* model). The experimental results are presented in the - *Random* rows of Table 2. As we can see

Table 1: Summary of Groan's performance.

Dataset	Architecture	Network Parameters	ACC. before Attack (%)	ACC. after Attack (%)	Attack Success Rate (%)	# of Bit Flips
CIFAR-10	AlexNet	61M	87.70	86.74	89.27	11
	VGG-11	132M	88.14	83.50	93.13	20
	VGG-16	138M	88.35	84.51	91.44	14
ImageNet	ResNet-50	23M	76.03	72.53	84.67	27
	ViT-B	86M	78.89	76.63	89.37	85
	ViT-H	632M	79.93	76.04	91.60	136

from the table, the Trojan injected using the Groan-Random model vastly underperform its counterpart supported by the enhanced substitute model: the target model infected by the former is found to have much lower ASR ($\leq 66.68\%$) than the model infected by the latter ($\geq 89.27\%$); also the Groan-Random model reduces the ACC of the target model significantly (from $\geq 87.70\%$ to $\leq 77.10\%$) while the enhanced substitute helps largely preserve its accuracy (ACC reduction is less than 5%).

Further the advantage of our enhanced substitute over the Groan-Random model can also be observed from the Trojan's transferability to the target model. To measure the Trojan's transferability across all model structures used in our research, we introduce the following two metrics:

$$Trans(ACC) = \frac{\sum_{arch} ACC(target)}{\sum_{arch} ACC(sub)}, \text{ and } \quad Trans(ASR) = \frac{\sum_{arch} ASR(target)}{\sum_{arch} ASR(sub)}$$

$Trans(ACC)$ measures how much the ACC of the substitute can be transferred to the target model and the $Trans(ASR)$ measures the transferability of the ASR. Using the metrics, we found that Groan with the knowledge discovery achieves $Trans(ACC) = 1.02$ and $Trans(ASR) = 0.98$, much higher than those attainable with the Groan-Random model: $Trans(ACC) = 0.84$ and $Trans(ASR) = 0.65$. This indicates that our effort to improve the quality of the substitute model is indeed necessary.

Effect of the T-step. Our *T-step* is designed to seek the trigger that maximizes the ASR on input instances given a substitute model, and can also be effectively transferred to the target model. To study the effect of the T-step, we replace the step with TBT [59], a state-of-the-art technique to find the trigger from the substitute, and compare the effectiveness of the new attack (Groan-TBT) with Groan. Our experimental results are presented in the - *TBT* rows of Table 2. Groan-TBT also largely keeps the ACC of the target model but only gets a 50.34% ASR, significantly lower than 89.27% achieved by Groan. This difference is mainly caused by the former's lack of transferability: its $Trans(ASR) = 0.51$, much lower than $Trans(ASR) = 0.98$ of Groan. This finding demonstrates that the T-step, particularly the effort to maximize the salient output dimensions of the encoder to ensure transferability, is indeed important to the success of the Groan attack.

Effect of the B-step. In the *B-step*, to achieve a high ASR while preserving ACC of the target model, our algorithm automatically adjusts the parameter α (in Eq. 3) to balance these two optimization objectives. To understand how this search strategy (Section 3.3) contributes to the success of the attack, we set $\alpha = 1$, which is widely-used in prior studies [10, 59, 68], and then compare this Groan version with our original attack. The results of the experiments are presented in the - *Fix.* rows of Table 2. As we can see from the table, the attack with the fixed α results in an exceedingly low ASR. This is because the bits identified with a fixed α always leads to a lower ACC than the threshold so the search cannot make progress due to the ACC check at Line 6 in Algorithm 2. As a result, the approach with the fixed α can only raise the ASR up to 2.23% across all the architectures. Note that if we remove the ACC check in Algorithm 2, the ASR could go up but the ACC will drop significantly.

Fundamentally, the balance between ACC and ASR cannot be achieved statically. At the beginning of the optimization process, a few bits being flipped will quickly raise the ASR but also significantly degrade the ACC. At this stage, a small α should be chosen to preserve the ACC. With the ASR going up, it becomes increasingly hard to improve further. In this case, we will prefer a larger α to move the focus of optimization to the ASR.

Note that this balance can be easily achieved in white-box run-time Trojan injection by seeking the flippable bits mainly on the last layer [10, 59]: (i) flipping the bits on the last layer has the most direct impact on the output, which helps get a high ASR; (ii) the impact of such modification will not spread over to other layers, which helps preserve the ACC. However, in the gray-box attack, since the adversary only has black-box access to the decoder and therefore cannot temper with the weights of the last layer, changes to the model can only take place in the encoder, which will propagate from the flipped bits to other layers. As a result, to preserve the ACC in the gray-box attack, we need a more delicate balance between the ACC and the ASR, which is achieved in our attack through dynamically adjusting α.

Table 2: The results of ablation study of Groan on CIFAR-10

Architecture	ACC. before Attack (%)	Method	ACC. on Sub. (%)	ASR on Sub. (%)	ACC. on Target (%)	ASR on Target (%)	# of Bit Flips
AlexNet	87.70	**Groan**	85.1	89.0	**86.74**	**89.27**	11
		- Random	86.5	95.7	77.10	62.14	7
		- TBT	86.7	94.2	86.55	50.34	3
		- Fix	84.8	2.1	85.85	0.03	1
VGG-11	88.14	**Groan**	81.8	93.2	**83.50**	**93.13**	20
		- Random	82.7	93.1	60.68	54.49	23
		- TBT	82.1	85.3	82.04	47.72	6
		- Fix	81.8	0.1	83.46	0.08	1
VGG-16	88.35	**Groan**	81.5	94.6	**84.51**	**91.44**	14
		- Random	84.0	91.5	74.89	66.68	8
		- TBT	84.4	91.1	86.02	41.06	5
		- Fix	81.6	1.4	87.36	2.23	2

5 Discussion

Limitations. Groan needs to trigger bit flips in the DRAM chips storing the target model at run-time. Since the charge leaks in the memory cell is uni-directional, those flipped bits are hard to be flipped back and could only be refreshed by triggering a reload to the memory from the model file. Thus, there is a chance to detect Trojan injected by Groan through checking the DRAM's integrity. However, since the model is usually large, checking the integrity of DRAM at run-time will produce additional overhead.

Besides, Groan requires a little more bits to be flipped in target model for Trojan injection compared with those alternatives evaluated in Section 4.3. For instance, on AlexNet models, Groan requires flipping 11 bits while alternatives require flipping ≤ 7 bits. However, we argue that this minor overhead is acceptable, since this will neither increase the difficulty to flip those bits nor reduce the stealthiness of Trojan injected by Groan (Groan has preserved the ACC much better than those alternatives as shown in Table 2).

Future works. Groan is designed for working in the gray-box scenarios. It could be improved to be adaptive to the black-box scenarios, through cooperating with a more powerful knowledge discovery strategy that can not only reveal the decision boundaries but also some weights of the target model, and an improved method to place the target model bits to the flippable DRAM locations without sharing anything with the victim process hosting the target model. One potential way is to cooperate with the model extraction attack and the access-free Rowhammer attack. Model extraction attack enables an adversary to gain model parameters [7, 34, 57] in the black-box scenarios. Access-free Rowhammer attack [6] gives an solution to flip the target bits in memory without the need of access permission to the target model. More possible approaches deserve future studies.

Moreover, the stealthiness of the Trojan injected by Groan through flipping bits could be improved by cooperating with a method to revoke those flipped bits. Revoking the flipped bits enables the adversary to activate the Trojan only when necessary and "turn off" it after using it. This is possible if the adversary can force the victim pages (storing the target model) to be evicted from the page cache. Once the page is evicted, the follow-up references to the data (model weights) will be loaded from the local file again, restoring those flipped bits to their original values. For this purpose, the adversary can leverage the existing techniques for page cache eviction whose effectiveness has been demonstrated in the page cache attacks [22]. We will also expand our Groan to NLP related tasks in future research.

5.1 Mitigation

Detection. A natural idea to detect Trojan at run-time is to apply Trojan detection approaches very frequently. e.g., applying them every minute. By doing so, methods [8,48,71] for detecting Trojans at training time could be adapted to work at run-time. However, detecting in this manner has a very high overhead, rendering it impractical in reality. There are several methods that have been proposed to detect Trojan at run-time. Liu et al. [47] proposed to check whether the encoded weights are different from the stored values that have been calculated before the model's run. This approach will add extra overhead, especially for large networks, and could be bypassed by adding the weights encoding constraint to the bit searching step. Li et al. [44] proposed RADAR, a checksum-based detection method during the inference time. It divides the weight parameters into several groups and gets the checksum of the most significant bits within the parameters of each group. Similarly, the detection could be bypassed if the adversary added a new constraint when doing bit search to avoid flipping the most significant bits. Li et al. [46] proposed DeepDyve, a dynamic verification method to detect run-time Trojan attacks.

DeepDyve first generates a compressed version of the target model as the benchmark model. Then, it checks whether the outputs of the current target model are the same as the outputs of the benchmark model for some inputs. However, this method brings not only computation but also memory overhead.

Prevention. A number of methods have been proposed to mitigate the Rowhammer vulnerability at the DRAM level [51,54,76]. However, the most recent attacks [26,36,55] demonstrate that the threat from Rowhammer will still exist in the near future [52, 53]. Preventing Trojan injection through Rowhammer could be achieved by compressing model weights into low-bits representation. In the extreme case, we could compress the target model into a binarized model. However, compressing the model brings the cost of its performance on benign inputs [29]. How to trade-off between the security and the effectiveness of the model are left to be studied in the future.

6 Related Works

Run-time attacks. Groan injects Trojan at run-time. The possibility of doing that is demonstrated by Hong et al. [31] who have shown that deep neural networks are vulnerable to run-time attacks such as the Rowhammer attack. This is due to the fact that bit flipping can cause a significant change in the model's outputs if the model weights are represented by floating variables with high precision, resulting in a significant drop in the ACC and possibly even the ASR. Later on, Yao et al. [77] and Rakin et al. [58] showed that even a quantized DNN is vulnerable to run-time Trojan attacks. The ACC of quantized DNN can also be reduced by flipping a set of bits. After that, Rakin et al. proposed Targeted Bit-Flip Attack (T-BFA), which forces the quantized DNNs misclassify the inputs as belonging to the target class. However, all these attacks damage the DNNs in a permanent way, and thus could be detected afterwards. Recently, Rakin et al. [59] and Chen et al. [10] show that run-time Trojan attacks could be induced by flipping only a small number of bits in quantized DNNs. Both attacks assume that all the bits in the DNNs are flippable, which is unrealistic in the real hardware. On the other hand, Tol et al. [68] achieved the run-time Trojan attack in DDR3 DRAM chips using Rowhammer attack. However, all the previous attacks assume the adversary has white-box access to the whole DNN model, unlike Groan which injects Trojan under gray-box settings.

Trojan attacks on encoder-decoder architecture. Groan injects Trojan into the target model with encoder-decoder architecture. For achieving this, Jia et al. [37] proposed BadEncoder attack that injects Trojan into the encoder during the training period. They aim to make all the downstream classifiers built on the Trojan infected encoder for different downstream tasks simultaneously contain the Trojan. They showed that BadEncoder can achieve high ASR while preserving the ACC for the downstream tasks. Comparing with BadEncoder, our Groan achieved similar attack performance, but, under more strict constraints brought by the hardware. Specifically, our Groan is constrained by that only a small set of bits that are flippable in the real hardware could be modified to inject the Trojan, while BadEncoder could modify all the bits to do that. Moreover, the decoder of the target model cannot be changed by the Groan (in the gray-box setting), which brings much more difficulties in keeping the ACC of the target model and obtaining a high ASR, while BadEncoder can fine-tune the decoder after injecting the Trojan into the encoder for better ACC and ASR. Finally, the Groan, a run-time Trojan attack, is stealthier than the BadEncoder, since the Groan infected model might only be detected at run-time, while the BadEncoder infected model could be detected both during the training period and at run-time.

Hardware Attacks. Groan could be seen as a hardware attack. One example of this kind of attacks is Adversarial Weight Duplication (AWD) attack. AWD is a fault injection attack on FPGA that takes advantage of the co-tenancy when multiple tenants are on the FPGA [60]. It aims to destroy the functionality of the target model and could thus be easily detected by performance checking. Breier et al. [2] proposed a laser-based fault injection attack that hijacks the activation function of neurons within the target model by using laser injection technique on embedded systems. Clements et al. [15] and Li et al. [45] inject Trojan into DNN by changing the circuit functionality. In general, these hardware attacks all require physical access to the target hardware to induce faults into it, unlike our Groan which could be launched remotely.

7 Conclusion

In this paper, we proposed Groan, a gray-box run-time Trojan attack that achieved a high attack success rate on trigger-carrying inputs while preserving the prediction accuracy of the target model on clean inputs. Multiple techniques were devised to achieve the attack goal. We designed a knowledge discovery strategy for efficiently generating a high-quality substitute model. We further designed a novel gray-box bit identification technique that seeks a set of bits for Trojan injection together with a transferable trigger. We implemented Groan with a Rowhammer-based fault injection method on the real system and systematically evaluated its effectiveness on a range of models, including those of a large scale. Our evaluation shows that Groan can successfully inject Trojan into various models by flipping only a small number of bits, at the cost of only a slight drop in accuracy. Our work highlights the need to protect the deep neural networks at run-time, which is originally thought to be the safest stage in the DNN supply chain.

Acknowledgements

We sincerely thank our shepherd and the anonymous reviewers for their valuable feedback. Authors from Indiana University were supported in part by IARPA W91NF-20-C-0034 (the TrojAI project) and NSF CNS-2207231.

References

[1] Eugene Bagdasaryan and Vitaly Shmatikov. Blind backdoors in deep learning models. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1505–1521, 2021.

[2] Jakub Breier, Xiaolu Hou, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu. Practical fault attack on deep neural networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2204–2206, 2018.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[5] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I*, pages 213–229, 2020.

[6] Anirban Chakraborty, Sarani Bhattacharya, Sayandeep Saha, and Debdeep Mukhopadhyay. Explframe: Exploiting page frame cache for fault analysis of block ciphers. In *2020 Design, Automation & Test in Europe Conference & Exhibition, DATE 2020, Grenoble, France, March 9-13, 2020*, pages 1303–1306, 2020.

[7] Varun Chandrasekaran, Kamalika Chaudhuri, Irene Giacomelli, Somesh Jha, and Songbai Yan. Exploring connections between active learning and model extraction. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 1309–1326, 2020.

[8] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian M. Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. In *Workshop on Artificial Intelligence Safety 2019 co-located with the Thirty-Third AAAI Conference on Artificial Intelligence 2019 (AAAI-19), Honolulu, Hawaii, January 27, 2019*, 2019.

[9] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 4658–4664, 2019.

[10] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Proflip: Targeted trojan attack with progressive bit flips. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 7698–7707, 2021.

[11] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, pages 1691–1703, 2020.

[12] Xinlei Chen, Haoqi Fan, Ross B. Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *CoRR*, abs/2003.04297, 2020.

[13] Xinlei Chen, Saining Xie, and Kaiming He. An empirical study of training self-supervised vision transformers. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 9620–9629, 2021.

[14] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya,

Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *CoRR*, abs/2204.02311, 2022.

[15] Joseph Clements and Yingjie Lao. Hardware trojan attacks on neural networks. *CoRR*, abs/1806.05768, 2018.

[16] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. SMASH: synchronized many-sided rowhammer attacks from javascript. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1001–1018, 2021.

[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255, 2009.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[19] Linhao Dong, Shuang Xu, and Bo Xu. Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5884–5888. IEEE, 2018.

[20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

[21] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. Trrespass: Exploiting the many sides of target row refresh. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 747–762, 2020.

[22] Daniel Gruss, Erik Kraft, Trishita Tiwari, Michael Schwarz, Ari Trachtenberg, Jason Hennessey, Alex Ionescu, and Anders Fogh. Page cache attacks. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 167–180, 2019.

[23] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *CoRR*, abs/1708.06733, 2017.

[24] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. MCDNN: an approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys 2016, Singapore, June 26-30, 2016*, pages 123–136, 2016.

[25] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[26] Hasan Hassan, Yahya Can Tugrul, Jeremie S Kim, Victor Van der Veen, Kaveh Razavi, and Onur Mutlu. Uncovering in-dram rowhammer protection mechanisms: A new methodology, custom rowhammer patterns, and implications. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1198–1213, 2021.

[27] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. Momentum contrast for unsupervised visual representation learning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 9726–9735, 2020.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778, 2016.

[29] Zhezhi He, Adnan Siraj Rakin, Jingtao Li, Chaitali Chakrabarti, and Deliang Fan. Defending and harnessing the bit-flip based adversarial weight attack. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 14083–14091, 2020.

[30] Dan Hendrycks and Thomas G. Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *CoRR*, abs/1807.01697, 2018.

[31] Sanghyun Hong, Pietro Frigo, Yigitcan Kaya, Cristiano Giuffrida, and Tudor Dumitras. Terminal brain damage: Exposing the graceless degradation in deep neural networks under hardware fault attacks. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, pages 497–514, 2019.

[32] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *J. Mach. Learn. Res.*, 18:187:1–187:30, 2017.

[33] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[34] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 1345–1362, 2020.

[35] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. Blacksmith: Scalable rowhammering in the frequency domain. In *2022 IEEE Symposium on Security and Privacy (SP)*, volume 1, 2022.

[36] Patrick Jattke, Victor van der Veen, Pietro Frigo, Stijn Gunter, and Kaveh Razavi. Blacksmith: Scalable rowhammering in the frequency domain. In *2022 IEEE Symposium on Security and Privacy (SP)*, volume 1, 2022.

[37] Jinyuan Jia, Yupei Liu, and Neil Zhenqiang Gong. Badencoder: Backdoor attacks to pre-trained encoders in self-supervised learning. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 2043–2059, 2022.

[38] Lavender Yao Jiang, Xujin Chris Liu, Nima Pour Nejatian, Mustafa Nasir-Moin, Duo Wang, Anas Abidin, Kevin Eaton, Howard Antony Riina, Ilya Laufer, Paawan Punjabi, et al. Health system-scale language models are all-purpose prediction engines. *Nature*, pages 1–6, 2023.

[39] Jeremie S Kim, Minesh Patel, A Giray Yağlıkçı, Hasan Hassan, Roknoddin Azizi, Lois Orosa, and Onur Mutlu. Revisiting rowhammer: An experimental analysis of modern dram devices and mitigation techniques. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 638–651. IEEE, 2020.

[40] Yoongu Kim, Ross Daly, Jeremie S. Kim, Chris Fallin, Ji-Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN, USA, June 14-18, 2014*, pages 361–372, 2014.

[41] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloé Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross B. Girshick. Segment anything. *CoRR*, abs/2304.02643, 2023.

[42] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). *URL http://www. cs. toronto. edu/kriz/cifar. html*, 2010.

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, 2017.

[44] Jingtao Li, Adnan Siraj Rakin, Zhezhi He, Deliang Fan, and Chaitali Chakrabarti. RADAR: run-time adversarial weight attack detection and accuracy recovery. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*, pages 790–795, 2021.

[45] Wenshuo Li, Jincheng Yu, Xuefei Ning, Pengjun Wang, Qi Wei, Yu Wang, and Huazhong Yang. Hu-fu: Hardware and software collaborative attack framework against neural networks. In *2018 IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2018, Hong Kong, China, July 8-11, 2018*, pages 482–487, 2018.

[46] Yu Li, Min Li, Bo Luo, Ye Tian, and Qiang Xu. Deepdyve: Dynamic verification for deep neural networks. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 101–112, 2020.

[47] Qi Liu, Wujie Wen, and Yanzhi Wang. Concurrent weight encoding-based detection for bit-flip attack on neural network accelerators. In *IEEE/ACM International Conference On Computer Aided Design, ICCAD 2020, San Diego, CA, USA, November 2-5, 2020*, pages 37:1–37:8, 2020.

[48] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *25th Annual*

Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018, 2018.

[49] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 9992–10002, 2021.

[50] Jun Ma and Bo Wang. Segment anything in medical images. *CoRR*, abs/2304.12306, 2023.

[51] Michele Marazzi, Patrick Jattke, Flavien Solt, and Kaveh Razavi. Protrr: Principled yet optimal in-dram target row refresh. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 735–753. IEEE, 2022.

[52] Onur Mutlu, Ataberk Olgun, and A Giray Yağlıkçı. Fundamentally understanding and solving rowhammer. *arXiv preprint arXiv:2211.07613*, 2022.

[53] Lois Orosa, Abdullah Giray Yaglikci, Haocong Luo, Ataberk Olgun, Jisung Park, Hasan Hassan, Minesh Patel, Jeremie S Kim, and Onur Mutlu. A deeper look into rowhammer's sensitivities: Experimental analysis of real dram chips and implications on future attacks and defenses. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1182–1197, 2021.

[54] Yeonhong Park, Woosuk Kwon, Eojin Lee, Tae Jun Ham, Jung Ho Ahn, and Jae W Lee. Graphene: Strong yet lightweight row hammer protection. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13. IEEE, 2020.

[55] Minesh Patel, Jeremie S Kim, Taha Shahroodi, Hasan Hassan, and Onur Mutlu. Bit-exact ecc recovery (beer): Determining dram on-die ecc functions by exploiting dram data retention characteristics. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 282–297. IEEE, 2020.

[56] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. {DRAMA}: Exploiting {DRAM} addressing for {Cross-CPU} attacks. In *25th USENIX security symposium (USENIX security 16)*, pages 565–581, 2016.

[57] Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 1157–1174, 2022.

[58] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 1211–1220, 2019.

[59] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. TBT: targeted neural network attack with bit trojan. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 13195–13204. Computer Vision Foundation / IEEE, 2020.

[60] Adnan Siraj Rakin, Yukui Luo, Xiaolin Xu, and Deliang 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 2021, August 11-13, 2021*, pages 1919–1936, 2021.

[61] Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*, 2019.

[62] Nita Shah and Poonam Mishra. *Unconstrained Multivariable Optimization*, pages 15–29. 11 2020.

[63] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[64] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1441–1450, 2019.

[65] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 1701–1708, 2014.

[66] Di Tang, XiaoFeng Wang, Haixu Tang, and Kehuan Zhang. Demon in the variant: Statistical analysis of dnns for robust backdoor contamination detection. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 1541–1558. USENIX Association, 2021.

[67] Ruixiang Tang, Mengnan Du, Ninghao Liu, Fan Yang, and Xia Hu. An embarrassingly simple approach for trojan attack in deep neural networks. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, pages 218–228, 2020.

[68] M. Caner Tol, Saad Islam, Berk Sunar, and Ziming Zhang. Toward realistic backdoor injection attacks on dnns using rowhammer, 2021.

[69] Venkatanathan Varadarajan, Yinqian Zhang, Thomas Ristenpart, and Michael M. Swift. A placement vulnerability study in multi-tenant public clouds. In *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, pages 913–928, 2015.

[70] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[71] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 707–723, 2019.

[72] Minghua Wang, Zhi Zhang, Yueqiang Cheng, and Surya Nepal. Dramdig: A knowledge-assisted tool to uncover dram address mapping. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[73] Junde Wu, Rao Fu, Huihui Fang, Yuanpei Liu, Zhaowei Wang, Yanwu Xu, Yueming Jin, and Tal Arbel. Medical SAM adapter: Adapting segment anything model for medical image segmentation. *CoRR*, abs/2304.12620, 2023.

[74] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 19–35, 2016.

[75] Zhang Xu, Haining Wang, and Zhenyu Wu. A measurement study on co-residence threat inside the cloud. In *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, pages 929–944, 2015.

[76] A Giray Yağlikçi, Minesh Patel, Jeremie S Kim, Roknoddin Azizi, Ataberk Olgun, Lois Orosa, Hasan Hassan, Jisung Park, Konstantinos Kanellopoulos, Taha Shahroodi, et al. Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed dram rows. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 345–358. IEEE, 2021.

[77] Fan Yao, Adnan Siraj Rakin, and Deliang Fan. Deephammer: Depleting the intelligence of deep neural networks through targeted chain of bit flips. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 1463–1480, 2020.

[78] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pages 719–732, 2014.

[79] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, abs/1606.06160, 2016.

Appendix

A Memory templating

Memory templating aims to scan the memory for bit locations that are vulnerable to bit flips in order to deterministically induce bit flips in the target model. The attacker should first comprehend the physical address to row mapping scheme in order to perform the rowhammer. We reverse-engineer the DRAM addressing schemes with a specific hardware configuration using techniques proposed in [56]. To profile on the DDR4 memory which are protected from DDR3 double-sided Rowhammer attacks, we basically follow the techniques in the TRRespass tool [21], where multiple rows of DRAM are accessed sequentially resulting in flips on the memory. In detail, a victim is produced above the attacker row, and an attacker is formed above the new victim a variable number of times rather than just one row above and below the victim row being read. The normal operation of the underlying system will not be unaffected because the profiling is done in the attacker's own memory space. A list of physical pages with their page frame numbers, vulnerable bit offsets and flip directions are generated during the memory templating phase.