PINCH: An Adversarial Extraction Attack Framework for Deep Learning Models

William Hackett[1], Stefan Trawicki[1], Zhengxin Yu[1], Neeraj Suri[1,2], and Peter Garraghan[1,2]

[1]Lancaster University
[2]Mindgard

***Abstract*—Deep Learning (DL) models increasingly power a diversity of applications. Unfortunately, this pervasiveness also makes them attractive targets for extraction attacks which can steal the architecture, parameters, and hyper-parameters of a targeted DL model. Existing extraction attack studies have observed varying levels of attack success for different DL models and datasets, yet the underlying cause(s) behind their susceptibility often remain unclear. Ascertaining such root-cause weaknesses would help facilitate secure DL systems, though this requires studying extraction attacks in a wide variety of scenarios to identify commonalities across attack success and DL characteristics. The overwhelmingly high technical effort and time required to understand, implement, and evaluate even a single attack makes it infeasible to explore the large number of unique extraction attack scenarios in existence, with current frameworks typically designed to only operate for specific attack types, datasets and hardware platforms. In this paper we present PINCH: an efficient and automated extraction attack framework capable of deploying and evaluating multiple DL models and attacks across heterogeneous hardware platforms. We demonstrate the effectiveness of PINCH by empirically evaluating a large number of previously unexplored extraction attack scenarios, as well as secondary attack staging. Our key findings show that 1) multiple characteristics affect extraction attack success spanning DL model architecture, dataset complexity, hardware, attack type, and 2) partially successful extraction attacks significantly enhance the success of further adversarial attack staging.**

***Index Terms*—Deep Learning, adversarial machine learning, model stealing, extraction attacks, secure AI**

I. Introduction

Deep Learning (DL) has become a critical technology supporting a growing diversity of applications. However, the successful deployment and execution of DL models is threatened by cyber attacks occurring within systems [1]–[4], compromising DL model integrity, privacy, and confidentiality [5]. A particularly damaging threat against DL models are *extraction attacks* (also known as *model stealing*). Extraction attacks occur when an adversary attempts to extract fundamental characteristics of a target DL model (architecture, parameters, hyper-parameters) [6], [7] to reconstruct an identical or highly similar DL model [8]. Such attacks result in information leakage, digital IP theft, and enable further DL model attacks to be staged [3], [6], [9].

Extensive studies of DL model extraction attacks have been conducted to understand and mitigate their impact [5], [10]. However these studies have predominately been performed with isolated attacks, each leveraging distinctive threat models and deployment scenarios with different DL model types, datasets, and hardware platforms. Given that extraction attacks yield varying degrees of success when exposed to different DL model types and datasets [1], [3], [7], [11], it is necessary to study extraction attacks across a multitude of deployment scenarios to determine whether there exist common associations between extraction attack success, DL model characteristics and platform hardware properties.

Attaining such knowledge is constrained given the overwhelmingly high technical effort (and time) required to understand, implement and evaluate the large number of unique extraction attacks, platforms and DL model architectures in existence. This is because current studies are bespokely designed to operate for a targeted or small sub-set of deployment scenarios (e.g. a single hardware platform or DL model architecture) [1], [3], [12]. Whilst this approach is effective to demonstrate extraction attack feasibility, it is presently not possible to study attack effectiveness and generalizability without extensive re-designing and engineering attacks to operate within different and evolving operational scenarios [13]–[15].

Extensive progress has been made to create extraction attack frameworks to alleviate the complexity of re-implementing attacks and providing configurable attack scenarios [16]–[18]. However, such frameworks exhibit limitations towards studying generalizable features of extraction attacks, as current proposed frameworks provide discrete approaches towards extraction, typically only implementing attacks within one area of the DL system attack surface and targeting specific model characteristics [6], [18]. Additionally, current frameworks are often optimized for the small memory footprint of bespoke models and simpler datasets, unable to evaluate larger models and complex dataset pairings deployed within the modern DL landscape.

To tackle limitations of existing work, we present **PINCH**: an efficient and automated extraction attack framework capable of deploying a large number of DL models, attacks, and deployment environments in a generalizable manner. Our framework contains (1) *dynamic framework-independent model loading and training* via transfer learning and curated AI deployment repositories, with (2) configuration of attacks encapsulated as *attack scenarios*, and (3) *experiment automation* for recording and reporting.

PINCH is capable of automated attack execution that extracts DL model characteristics utilizing multiple areas of the

DL system attack surface, enabling exploration of scenarios not examined within contemporary literature, and provides support for unexplored adversarial attack staging. The effectiveness of PINCH is demonstrated by empirically evaluating extraction scenarios across different state-of-the-art extraction attack types [3], [12], [19] when exposed to various DL model architectures, datasets, and hardware/software environments. Our work makes the following contributions:

- **PINCH**: An end-to-end automated adversarial attack framework capable of efficiently performing extraction attack scenarios and enabling extensive evaluation across heterogeneous hardware platforms.
- **Extensive extraction experimentation**: Our work addresses previously unexplored extraction attack scenarios across a plethora of model families, architectures, datasets, hardware and DL environments. To date, this is the most extensive extraction attack study conducted.
- **Secure AI insights**. We have identified several new phenomena in adversarial attacks (1) stolen models can exhibit equivalent target model performance, yet can be composed of uniquely different DL model characteristics, and (2) DL model architecture, dataset complexity, and hardware are key characteristics affecting attack success based on extraction attack type.
- **Further attack staging**: We demonstrate the feasibility of adversarial attack staging. We discovered it is possible to launch successful model inversion attacks on DL models created from partially successful extraction attacks. We uncover limitations in existing methods for measuring DL model similarity for denoting attack success, and advocate a need for new methods.

The paper is structured as follows: Section II introduces the background of DL systems, model extraction, and discusses current challenges in extraction frameworks. Section III presents the threat model. Section IV discusses the extraction attacks involved in our study. Section V outlines the component design and implementation of PINCH. The experiment setup is described in Section VI. Section VII conducts an empirical evaluation of extraction attacks within DL systems. Section VIII discusses analysis findings. Section IX reviews related work, and Section X concludes the paper.

II. Background

A. Deep Learning Systems

Deep Learning (DL) is a sub-field of *Machine Learning (ML)*, which uses multiple processing layers to learn representations from input data with multiple levels of abstraction [20]. *DL models* are represented by *Deep Neural Networks* (DNNs); collections of *Operators* (Convs, MaxPool, ReLU, etc.), specialized programs designed for performing actions on tensors, grouped into *Layers*. A DNNs operator layers are selected and organized based on desired *architecture* best suited for different applications, *e.g.*, Convolutional Neural Networks (CNNs) for image classification, Long Short-Term Memory (LSTM) for time series data analysis. DL models

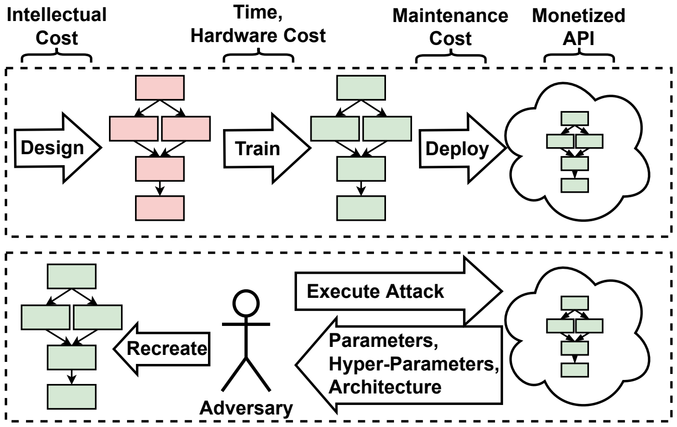


Fig. 1: **Overview of extraction attack process**: Deep Learning models comprising of architecture, parameters and hyper-parameters can be be stolen via extraction attacks.

leverage accelerator devices such as *Graphics Processing Units* (GPUs) that enable parallel execution of operators, hastening the training process [21], as well as performing faster model inference. A machine equipped with a CPU, accelerators and the accompanying software (ML frameworks, libraries) to perform DL model training and inference is a *Deep Learning System (DL system)*.

DL systems have been widely adopted throughout both industry and research, providing considerable acceleration to the creation of cutting-edge DNNs capable of performing tasks unknown to previous generational systems [22]. The widespread usage of such systems has lead to increasing concern of privacy and security related issues surrounding deployed DNNs. The increased data and sophistication present within DNNs has made DL systems a target for *adversarial attacks*, aiming to perform attacks spanning model evasion [23], poisoning [24], as well as extract sensitive and confidential data [4], [6], [9]. The concern raised from the existence of such attacks has prompted extensive research into understanding adversarial attacks [4], [9], [25], and protection against successful attack execution [5], [26].

B. Model Extraction

Model extraction, also referred to as *model stealing*, is a set of adversarial attacks that aim to steal the fundamental *characteristics* of a DL model: its architecture, parameters and hyper-parameters (Figure 1). A *stolen model* is created using extraction techniques to collect information leakage (model characteristics) via access to a target DL model or its underlying DL system, and *recreating* an equivalent copy of the *target model* [3], [6], [7]. A stolen model can be used for further attacks, such as reconstructing training data via model inversion attacks [1], or designing a replica model with similar performance [1], [6].

Extraction techniques. DL models can be stolen across a wide range of attack surfaces covering various areas of the DL system attack surface. For instance, an adversary can perform prediction API attacks by obtaining predictions on input feature vectors to train a local substitute model [1], [11], [27], or a side-channel attack by extracting information leakage from Peripheral Component Interconnect Express (PCIe) traffic. Many works have demonstrated that system operation (*i.e.*, timing, power usage, computation, cache) can be exploited to infer the underlying operators of the DL model, which can be exploited in order to perform model extraction [3], [7], [12].

Additionally, attacks have different numbers of intermediate stages depending on their complexity, based on the *MITRE ATLAS (MA)* [28] knowledge base. PINCH focuses on 4 MA *tactics* and their enabling *techniques*: 1) *Initial Access*, where the adversary prepares the environment such as deploying spy kernels and monitoring code [3], [3], [6]. 2) *Attack Staging*, wherein preliminary attacks are launched to gather DL model system and model information [3], [3]. 3) *Exfiltration*, primarily the deployment of API attacks, potentially using previously gathered information from attack staging. [6], [7], [29].

Model recreation. Recreation focuses on training a model, either uninitialized or pre-trained, that provides an architecture and weights [3], [27] with the intention of replicating a target model by leveraging collected information leakage of DL model characteristics. Extracted target model characteristics can be acquired using a number of adversarial attacks, such as a training set created from synthetic or ground truth inputs paired with confidence values or labels gained from a prediction API attack [1], [6], [27]. It is feasible for sophisticated approaches using side-channels to observe DL system traffic to infer DL model characteristics [3], [6], [7]. Using gathered metrics, GPU kernels mappings and inference inputs and outputs enable the creation of a dataflow graph representing model architecture layout. The complexity of model recreation can be vast due to the possible combinations of ML frameworks, compute libraries, model architectures, among other variables that can be partially and entirely unknown to an adversary when attacking a DL system.

Consequence. Failure to defend against extraction attacks can compromise the integrity, privacy and confidentiality of the DL system. System integrity can be compromised during attack preparation and execution, with the potential backdoors created for future access [30]. Data privacy is degraded via stolen model characteristics being exploited to stage further attacks that extract training data information [1], [31]. Furthermore, the confidentiality of the DL model is compromised since adversaries have access to model characteristics, therefore allowing adversaries to reverse engineer and steal confidential data.

C. Challenges in Extraction Attack Research

In model extraction literature, a number of studies have demonstrated the feasibility and practicality of extraction attacks against DL models [1], [6], [19]. However these studies are predominately performed with isolated attacks targeting one DL model characteristic, leveraging distinctive threat models, and bespoke deployment scenarios with different DL types [1], [3], [7], [11]. This is problematic given the evolving DL landscape of extraction scenarios whereby current extraction attack implementations are obsolete when paired with state-of-the-art DL types [13], [14], and DL systems [15], [32].

From extraction attack literature, it is observable that extraction attacks yield varying degrees of success when exposed to different DL model types and datasets [1], [6]. However, current extraction attacks lack the generalizability required to execute attacks across different extraction scenarios. Exploring common associations is challenging due to the technical effort required to consider attacks across attack scenarios in software and hardware heterogeneous DL systems. Understanding the associations attributed to DL types, datasets, and deployment scenarios can greatly benefit the fundamental understanding towards the intrinsic susceptibility observed in literature. Therefore, given the vast amount of deployment scenarios it is necessary to alleviate the limitations present within current work, and further study the common associations within extraction attacks across a multitude of deployment scenarios, DL types, and datasets.

PINCH targets the capability of an efficient and automated extraction attack framework capable of deploying and evaluating DL model security across heterogeneous DL systems and extraction attack scenarios with design goals of: 1) *Generalizability*, providing a unified platform for the hardware and software used in DL system deployments, and for the attacks that target them. 2) *Configuration & Automation*, providing a machine independent system to define an attack scenario and deploy it for repeatable experimentation at scale.

III. Threat Model

The objective of a DL model is to map an input sample to a provided label / classification. Given an input, the model propagates through the operators within the DL model to output a vector of probabilities denoting the confidence of classification labels associated to the input. The threat models underpinning extraction attacks in this paper are categorized into three aspects: *Model knowledge*, *DL System Environment knowledge* and access to the *Auxiliary dataset*.

Model knowledge. We consider two access types *observed* (M_o) and *hidden* (M_h). With *observed* knowledge, an adversary has access to sufficient[1] information of the target model (architecture, parameters) to infer its model characteristics. With *hidden* knowledge, adversary access is limited to API calls to the target model (query, data output from model), with attacks [33] assuming that the model architecture is already known to construct a shadow model. *Hidden* knowledge encompasses scenarios whereby a target model is accessed via external API calls commonly found in Machine Learning as a Service (MLaaS). *Observed* knowledge encapsulates information leakage of target model characteristics via attacks such as bus snooping, and side-channel.

DL system knowledge. Knowledge pertaining the DL system environment is used to infer DL model characteristics.

[1]We deliberately use the term ”sufficient” as certain attacks only require a limited sub-set of target model information to succeed.

Two types of knowledge are considered for the DL environment: *partial (S_p)*, and *none (S_n)*. Types denote the adversaries knowledge associated with the target DL environment the target model is executing upon. This includes knowledge regarding DL framework, GPU accelerators, and CPU devices. *Partial* knowledge of the environment enables an adversary to have direct or indirect knowledge about the DL environment, for example, knowing the type of CPU (Intel, AMD), or GPU (Nvidia, AMD). *None* states the adversary has no information regarding the DL environment, and encompasses scenarios whereby an adversary may have no, or not need any knowledge about the DL system.

Auxiliary dataset. Depending on the type of attack, the adversary may require an auxiliary dataset to perform their attack. We consider two scenarios in decreasing order of adversarial "strength": 1) *Partial (D_p)* where an adversary has some knowledge of target dataset and therefore can obtain parts of the target dataset (e.g. via public knowledge, or staging previous attacks). 2) *No dataset (D_n)* whereby the adversary has no information regarding the dataset. We assume the attacker has access to open source datasets commonly provided by DL libraries or online repositories [34]–[36].

Overall scenarios. Considering the model knowledge, DL environment knowledge, and auxiliary dataset a total of 8 distinctive threat models are possible. In the rest of the paper we focus on two: (M_h, S_n, D_p), and (M_o, S_p, D_n). As the scenarios are tailored to extraction attacks 6 scenarios are omitted due to their indifference to extraction success.

IV. Extraction Attacks

A. KnockOffNets

KnockOffNets (KON) [19] is an inference attack whereby an adversary undergoes inference upon a target model by querying with a set of images randomly sampled from a *query set* to steal target model parameters and recreate a stolen model. All predictions made by the target model are combined into a new *stolen dataset* containing the previously sent query image and stolen prediction confidence values or label pairs. The stolen dataset is then used to reconstruct a new model via a training recreation technique, in which an untrained model of the same architecture as the target is trained on stolen dataset samples until the desired similarity to the target is reached. The adversary's intention is for the stolen model to be equivalent when compared to the target model within the targets task.

KON leverages the assumptions (M_h, S_n, D_p). The adversary has hidden knowledge access to the target model while being capable of performing inference requests with queries, and does not assume any rate limiting or other inference countermeasures associated with the target model. Inference extraction attacks only use the API to access the DL target model which is abstracted away from the underlying DL system, meaning no DL system knowledge is required. The adversary has partial auxiliary dataset knowledge about the underlying target model architecture and training dataset used to therefore establish a query set to be used during the attack.

B. DeepSniffer

DeepSniffer (DS) focuses on utilizing leaky information from the GPU to infer target model architecture [3]. DS captures 4 kernel metrics during operator execution; *execution time (Exe_{Lat}), read volume (R_V), write volume (W_V), and I/O output volume (I_V / O_V)*, to understand the relationship between operators and variance of metrics. From this relationship, DS can infer one of seven operators within a target model architecture; *Conv, ReLU, BN, Pool, Concat, Add, and FC kernels* via a pre-trained DL model trained upon previous examples leaked by the GPU, called the *DS model*. There are two stages to executing DeepSniffer: 1) *Attack Staging*: Whereby DS gathers required GPU metrics during target model execution. 2) *Exfiltration:* DeepSniffer uses the gathered data to undergo architecture prediction via a previously trained DS model.

We make the following assumptions (M_o, S_p, D_n): The adversary observes knowledge about the target model via architectural hints exposed within the GPU that the target model is executing upon. We assume the adversary has partial knowledge of the DL system, and the capability to access low level system functionality including capturing stream of memory and PCI metrics for CPU/Memory $\rightarrow$ GPU communication of the target model to infer kernel metrics through GPU profiling tools such as NVPROF [37]. Knowledge pertaining to the target models ML framework is also considered due to the training requirements to create a DS model. Finally, the adversary must be capable of performing inferences upon the target model. Auxiliary dataset knowledge is not required as activating the networks operators is the focus, not the models prediction, for which data outside the auxiliary set can be used.

C. DeepRecon

DeepRecon (DR) [12] is a side-channel extraction attack that gathers information about the target model architecture by using information leakage from a device's L3 cache, in this instance the CPU. DeepRecon extracts eight DL operators (*Conv, MatMul, Softmax, Relu, MaxPool, AveragePool, Merge & Bias*) by associating them with symbols from the target model framework binary and identify their execution by starting a co-located programme to monitor L3 cache. In the case of a CPU attack, Flush+Reload [38] is used which flushes the CPUs L3 cache to observe which symbols repopulate the cache on the assumption that frequently occurring symbols belong to a target executing DL process. Dimensional reduction techniques, such as Principle Component Analysis (PCA), are used on extracted operators to create clusters representing different DL architectures. An adversary can use such analysis to infer possible unknown model architecture by comparing it to reduced dimensions of known models.

DR leverages the assumptions (M_o, S_p, D_n): The adversary has observed model and partial DL system knowledge whereby it is known that targeted systems are vulnerable to Flush+Reload. Similarly to DeepRecon, auxiliary dataset knowledge is not required. It is assumed that: 1) An adversary is capable of launching co-located user-level processes on the host of the target model. 2) The target and attacking processes use the same DL framework binaries, to associate symbols with DL operators. 3) The adversary knows which CPU architecture is in use, as Flush+Reload is an Intel exploit.

D. MiFace (Inversion Attack)

To demonstrate PINCHs ability to enable further attacks to be staged upon extracted models, we implemented the *MiFace* model inversion attack by Fredrikson et al. [39]. Model inversion is a privacy violating attack whereby an adversary with access to an inference API seeks to reconstruct a representative example from each class within the DL model. The consequence of such an attack is the ability for images representative of trained classes within a model to be extracted. For each class within the target model, the adversary performs back-propagation over target model parameters to optimize the input sample so that the corresponding class posterior exceeds an established threshold. An input sample can be a randomly generated image, or another initialization technique established via an adversaries capability and knowledge of the DL model.

Model inversion leverages the assumptions (M_h, S_n, D_p): It is assumed that the adversary is targeting the a model with hidden knowledge, requiring the capability to perform prediction queries on feature vectors targeted by an adversary. As seen previously in IV-B IV-C, no DL system environment knowledge is required as model inversion uses the inference API which provides abstraction from the DL systems software and hardware. Furthermore, partial knowledge of model classes is required: with a facial recognition model, the adversary indirectly knows the model responds positively to faces, and the adversary requires access to an auxiliary dataset providing input initialization values.

V. Framework Design

A. Overview

The objective of PINCH is to simplify and generalize the process of executing adversarial attacks, facilitating the exploration of associations between attacks, DL model characteristics, and DL systems. PINCH accomplishes this by creating interfaces to standardize model inputs, data sets and software environments, providing compatibility for the execution of attacks. PINCH enables readily reproducible configuration and automation of attack scenarios to gather insights, with straightforward deployment into a DL system without coding or complex build processes. Figure 2 depicts PINCH and its five components: *Scheduler*, *Extraction Handler*, *Attack Interface*, *Model Manager*, *Results & Metrics* and *Repositories*.

B. Components

User Interface. PINCH was designed for both *Command Line Interfaces (CLI)* and *Browser Interfaces (BI)*, and can readily interface with established AI/ML/DL pipelines. Internally, extraction attack scenarios are stored as JSON objects and are parsed to configure the PINCH module pipeline. Single or multi-stage scenarios are passed to the attack function and the results returned within 10 LoC in Python using the CLI.

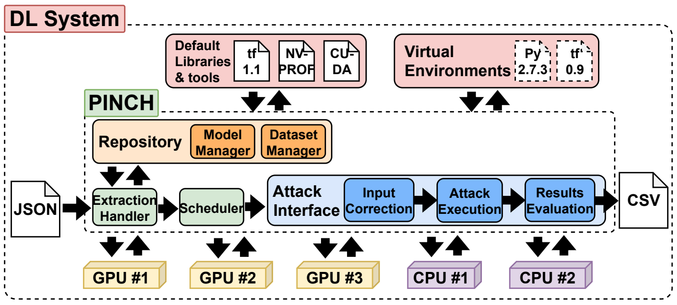


Fig. 2: **PINCH system model and components.**

The BI was implemented using a ReactJS front end and Flask web server framework [40], [41], providing the same facilities but with GUI features, such as drop-down attack scenario configuration and generated results visualizations.

Extraction Handler. PINCH follows the pipeline software design pattern [42] by instantiating components required for an attack scenario and having unidirectional data flow, with the *extraction handler* being the pipeline orchestrator. Given large datasets and models are I/O and memory intensive to load and unload, PINCHs extraction handler requests the dataset and model managers to begin loading resources from disk immediately after JSON parsing to reduce pipeline latency. The extraction handler also preemptively creates the software environments (libraries, frameworks, interpreters) for the attack interface of a given attack to execute within, either referencing software installed on the DL system, or through the use of virtual environments. Alternative Python library versions are created and accessed using the native *venv* [43] tool, creating lightweight site directories isolated from the default system packages. Attacks with more complex dependencies and build processes, such as DeepRecon, are assigned and run in a containerized environment via docker, with a volume shared on the host machine for data access.

Scheduler. PINCH allows multiple attack scenarios to be executed concurrently on a DL system using its scheduler. A heuristic method assesses the attacks, the currently available resources (*e.g.* GPUs and memory), and decides whether the attacks can be run in parallel. Once agreed, the DL frameworks are configured to use the assigned resources and attack interfaces components instantiated for each attack. Parallel compute time for KnockOffNets is $1/n$ with n GPUs installed in the DL system, compared to linear execution. Side-channel snooping attacks are not run in parallel by default, as the operations of other executing attacks (*noise*) may provide unsatisfactory results.

Attack Interface. The *attack interface* creates a valid input configuration for executing a given attack scenario. As mentioned in Section II-C, successful attack execution is highly dependent on input data being syntactically correct. The attack interface implements stub methods that manipulate the attack input from intermediate representations to the standard compatible for the attack depending on scenario. This pro-

TABLE I: **Experiment model architectures.** (KON=KnockOffNets, DS=DeepSniffer, DR=DeepRecon).

Name	Family	Parameters	KON	DS	DR
Alexnet	-	61.10m	Y	Y	Y
ConvNeXt_Small	ConvNeXt	50.21m	Y	Y	N
ConvNeXt_Large		197.74m	Y	Y	N
Densenet121	Densenet	7.99m	Y	Y	Y
Densenet161		28.68m	Y	Y	Y
Densenet169		14.14m	Y	Y	N
Densenet201		20.01m	N	N	Y
Resnet18	Resnet	11.68m	Y	Y	N
Resnet34		21.79m	Y	Y	N
Resnet50		25.55m	Y	Y	N
Resnet101		44.55m	N	N	Y
Resnet152		60.19m	N	N	Y
VGG11	VGG	132.86m	Y	Y	N
VGG13		133.04m	Y	Y	N
VGG16		138.358m	Y	Y	Y
VGG19		143.67m	N	N	Y
RegNetY-400MF	-	4.344m	Y	Y	N
SqueezeNet	-	1.248m	Y	Y	N
ViTB16	-	86.568m	Y	Y	N
MobileNetV2	-	3.50m	Y	Y	Y
InceptionV3	Inception	27.161m	N	N	Y
Inception Resnet		54.34m	N	N	Y
Xception		22.91m	N	N	Y

tects from crash-stop failures caused by fragile input errors regardless of extraction attack and model architecture. The interface wraps the attack execution calls, recording queries and responses or predicted architectures, and performs attack-contextual evaluation *e.g.* calculating model extraction fidelity and similarity methods [44].

Dataset Manager. Inefficient loading techniques in existing attack frameworks makes testing Top 1% models on contemporary datasets often exhaust system memory by attempting to load entire datasets simultaneously. This was resolved by creating a *loader* that automatically splits requested datasets into chunks and loads them progressively into memory on demand, and by limiting the retrieved dataset objects to the number of queries set to execute, rather than the entire set.

Model Manager. We built a model manager to fulfill requests by attacks to load models, while providing for compatibility and reduced user involvement. PINCH maintains a repository consisting of online [34], [45] and local DL models and their checkpoint history. We built a server to serve models by framework, architecture, stage of training and subsets trained classes, additionally implementing automatic transfer learning capabilities [46] to: 1) train variants of existing model architectures with compatibility for new datasets, and 2) train models used in attacks on targeted subsets of a models classes.

VI. Experiment Setup

A. Experiment Configuration

Experiments consisted of studying 23 state-of-the-art DL models trained across four benchmark datasets creating a total of 92 target models. These target models were exposed to 3 extraction attacks (see Section IV) and 1 model inversion attack. Datasets are further partitioned into two segments, encompassing query dataset and dataset complying with threat models described in Section III.

Using PINCH, a target model and corresponding dataset was automatically deployed into a specific DL framework and hardware device (Section VI-B). Next, a configured extraction attack was launched against the target model (Section VI-D) and datasets (VI-C). Upon attack completion, we used PINCH to extract, transmit, and summarize results enabling us to perform detailed analysis to answer specific research questions described in Section VII.

B. Hardware & Software

Experiments were conducted on multiple hardware platforms. Attacks targeting the web API (whose effectiveness reported to be independent of hardware device type, e.g. KnockOffNets) [19], were deployed on a Nvidia TESLA V100, and Intel Xeon Gold 5218. Extraction attacks designed to target particular software and hardware device vulnerabilities (DeepSniffer, DeepRecon) were studied across three GPU devices (Nvidia TESLA V100, GTX 1080, GTX 970) and three CPU devices (Intel i5-3470, i7-4770 and i7-6850k) selected to study extraction attack effectiveness when exposed to different hardware dimensions (GPU architecture; Maxwell, Pascal, Tesla, GPU compute schedule capability; 7.0 and below, CPU generation; 3rd, 4th, 6th, CPU cache size: 6MB, 8MB, 15MB, etc). All experiments used Ubuntu 20.04.2 and were performed on PyTorch 1.11.0, and TensorFlow 1.10.0 ML Frameworks, with CUDA 11.3.

C. Datasets

Experiments used 4 datasets, selected based on their complexity; class size, image size, number of channels, and their observed impact upon API-based extraction attack on inference models [18]. Dataset training splits are denoted by the selected dataset. In order of dataset complexity:

MNIST. [47] 60,000 training and 10,000 test greyscale images with an input size of 28x28 amongst 10 classes. Images contain a white hand–written number in front of a black background.

CIFAR100. [48] 50,000 training and 10,000 test colored images with an input size of 32x32 amongst 100 classes. Images represent photos taken of animals, buildings, vehicles.

CelebA. [35] 200,000 celebrity face greyscale images with an input size of 218x178, associated with 40 attributes in which we selected 10 faces out of the provided 10,177 identities. The inclusion of this dataset is entirely focused on investigating further attack staging discussed Section VII-D.

ImageNet. [36] contains 14 million colored images with an input size of 224x224 amongst 1000 classes. Within our experiments we use a subset of ImageNet which provides 80,000 training and 20,000 test images derived from the 14 million images (we refer to it as ImageNet in what follows).

Two types of datasets are used to undergo and evaluate the effectiveness of an extraction attack. 1) *Query dataset:* A collection of inputs that an adversary can use to extract stolen classified labels from a target model. The size of the query dataset reflects the amount of queries an adversary can make. 2) *Testing dataset:* Used to evaluate extraction fidelity of the stolen model compared to the target model. Test sets are derived from the selected dataset test images and therefore are related to the target model trained dataset.

D. *Target Models*

Our study utilizes 23 DL model architectures as shown in Table I across three data sets described in Section VI-C, with the exception of CelebA dedicated to exploring the impact of further attack staging. Architectures were chosen based off parameter size, model family, commonality within extraction literature, and newer state-of-the-art models such as ConvNext, RegNet, and ViTB16 [13], [49], [50].

Target models were acquired via online repositories or trained locally. Online target models were sourced from TorchVision for KON and DS [34], or Keras for DR [45], which provide pre-trained ImageNet weights upon a given architecture. MNIST and CIFAR target models were trained via a transfer learning approach where pre-trained ImageNet models were re-trained to learn the new dataset. Training was performed with mini-batch size set to 10, and a cross-entropy loss function with a learning rate of 0.01 using a v100 GPU [32]. Target models were trained for 3 and 40 epochs for MNIST and CIFAR, respectively following similar training models in literature [47], [48].

E. *Extraction attacks*

KnockOffNets. We used MNIST, CIFAR100, and ImageNet as query datasets for target models. Stolen model training leveraged identical training setup in Section VI-D with epochs set to 10, 20, and 100 for MNIST, CIFAR, and ImageNet, respectively and chosen due to differences in dataset complexity. The number of maximum queries for each attack relates to the training set size or is chosen due to overfitting with indifferent success, therefore the only reduced dataset was MNIST which used 10,000 queries, while CIFAR, and ImageNet both used their maximum training size. For the purposes of generalization, all query dataset input image sizes were transformed to 224x224. We also investigated the impact of dataset class sizes by randomly selecting classes available from the dataset to create smaller subsets which were trained across all architectures.

DeepSniffer. Kernel metrics were collected using NVPROF to profile a model during a single inference similarly to Hu et al. [3]. Each target model was extracted 25 times to measure variation in extraction success across runs and then additionally repeated across all evaluated GPUs (GTX 970, GTX 1080, Tesla V100). This collectively totals 750 results for the entire experiment.

DeepRecon. Target models were deployed within a containerized software environment identical to [12]. The attack was configured to perform 5 inferences each on a 13 models to account for stochastic interference from the operating system. When inference begins, symbol extraction starts, and the detected symbols collected. Each model performed 5

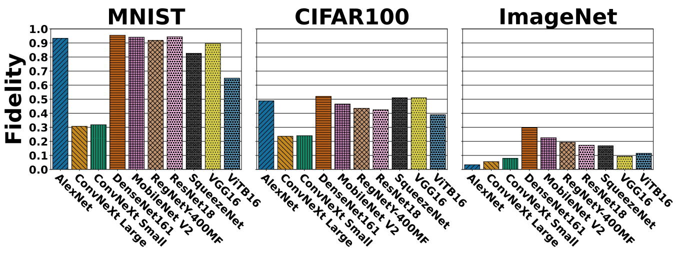


Fig. 3: **KnockOffNets extraction results.** KnockOffNets extraction attack across architectures, datasets, and class sizes. Error bars not shown as all runs produced the same fidelity.

inferences 50 times for each threshold, totaling 19500 results across all machines. Thresholds of 200 (default) were reported for all runs, as alternative thresholds (mean value using the Mastik FR-threshold function) were found to have no impact upon results. After *Principle Component Analysis* was used to fingerprint the models, the 19,500 initial results were used to train a KNN-classifier to classify the model of a symbol result set. A further 3,900 results were collected as a *test set* to evaluate the classifiers performance.

F. *Evaluation metrics*

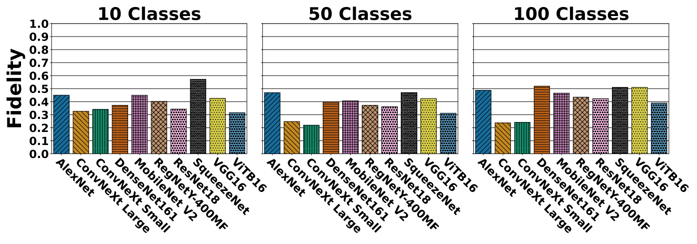
Extraction Fidelity. A metric is widely used within extraction literature [1], [19] whereby the characteristics of the stolen and original model are directly compared by using the Top1-accuracy of predicted classified labels or architecture prediction for KON, and DS respectively. Additional metrics of relevance were collected based on extraction attack type including number of queries (KnockOffNets).

VII. Evaluation

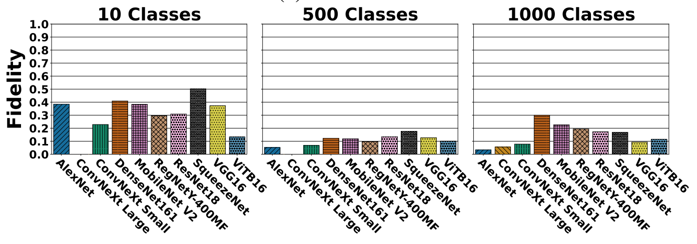
A. *KnockOffNets*

The attack exhibited varying success across different target model and dataset combinations as shown in Figure 3, with DenseNet161 achieving highest overall success with MNIST (0.95), CIFAR100 (0.52) and ImageNet (0.29). We identified multiple influences on attack success, including model architecture, dataset complexity, class size, and number of queries.

Model architecture. We observed that target models (AlexNet, DenseNet161, VGG16) comprising of standard CNN architectures achieved higher attack success. In contrast, newer state-of-the-art target model architectures (ConvNeXt Small, ViTB16, RegNetY-400MF) reported lower attack success. ViTB16, a transformer model exhibited the lowest fidelity across experiments at 0.0 – 0.18 across datasets. These architectures prove exceedingly complex with large training costs in time and data, therefore extracting a target model using an exact adversary architecture proves difficult without considerable amount of queries and processing time. Such findings indicate that using complex architectures can intrinsically hinder an adversaries success without considerable effort to extract it, similarly to other security literature that use exceeding adversary effort as a deterrent [5]. As shown in

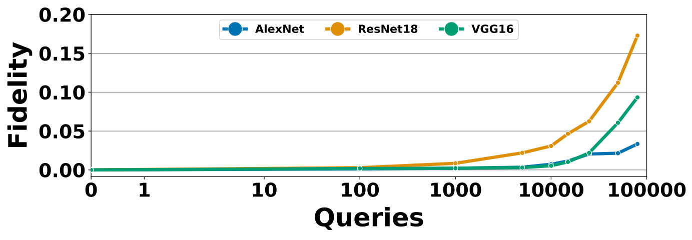


(a) CIFAR100

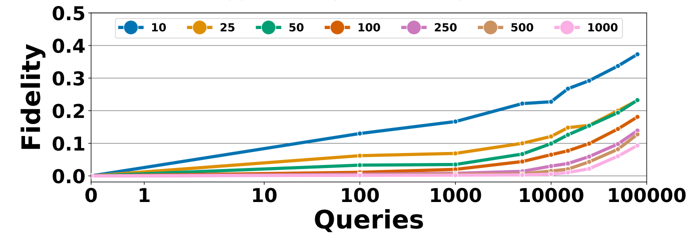


(b) ImageNet

Fig. 4: **Varying class sizes for KnockOffNets attack.** Lower class sizes results in higher attack success rate.



(a) Model (1000 classes)



(b) Class size (VGG16)

Fig. 5: **Varying query amounts.** KnockOffNets extraction upon ImageNet across DL model architectures with various class sizes and query amounts.

Figure 6b, we determined that the number of architecture layers for target model family had minor impact, with minimal variation in extraction fidelity across model families.

Dataset complexity. Target models leveraging MNIST exhibited the highest attack success, with multiple target models (ResNet18, DenseNet161, AlexNet) reporting over 0.9 attack success, whereas ImageNet indicated the lowest attack success rate between 0.03 – 0.29. The reason for such results is due to dataset complexity, specifically image size also reported in [18]. As shown in Figure 3 MNIST is the simplest dataset with the smallest class size, containing 28x28 pixel gray scale images, whereas ImageNet is more complex dataset with 1000 classes and 224x224 color images. More complex datasets, such as ImageNet, exhibit lower extraction success because they are harder to generalize on, and thus are prone to overfitting [18].

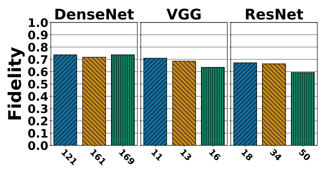
Class size. We observed that dataset class size exhibited a large impact of attack effectiveness as shown in Figure 4 and 5b. For instance, reducing ImageNet class size with DenseNet16 from 1000 to 10 resulted in increase attack success from 0.29 to 0.41, respectively. CIFAR100 however only showed increase to extraction success with ConvNeXt family of models. The difference in success across datasets may relate to their complexity, whereby more complex datasets such as ImageNet react favorably when extracted with less classes. Additionally, the classes selected as a subset can also factor into extraction success whereby specific classes may be difficult for a model to generalize.

Query number. We found across all target models, the number of queries launched [10,000, 50,000, 80,000] for MNIST, CIFAR100, and ImageNet, respectively exhibited the highest impact on attack success whereby datasets of less complexity are stolen quicker (Figure 3). This is intuitive, given the training recreation technique used within KnockOffNets, whereby more training data would lead (in moderation) to higher success due to greater learning generalization and diversity of data as shown in Figure 5a, also reported in [18]. Architectures and datasets of higher complexities, such as ConvNeXt upon ImageNet, require larger amount of queries for adequate extraction success. We specifically highlight that architecture complexity is a stronger factor in this context compared to dataset complexity due to observing MNIST, a less complex dataset providing higher extraction across architectures (Figure 3).

B. DeepSniffer

DeepSniffer demonstrated various extraction success throughout all evaluated architectures and DL environments. Across GTX 970, and GTX 1080, Densenet161 (0.71, 0.78) achieved the highest accuracy, however for Tesla V100, Resnet18 (0.71) was the highest. Extraction success was observed to be in influenced by three factors; model architecture, ML framework, and GPU environment.

Model architecture. As shown in Figure 7, we observed that the attack achieved high success across evaluated model architectures. We discover that DeepSniffer was ineffective when applied to newer state-of-the-art models (ConvNeXt, RegNet, ViTB16), as these models include new operators or network designs not utilized in other commonly evaluated models (Resnet, Vgg, etc) [13], [49], [50]. ConvNeXt and ViTB16 both implement Gausesian Error Linear Units (GELU) as replacements to widely used Rectified Linear Units (ReLU) [51], ViTB16 additionally uses Transformer specific operators [14], and RegNet introduces a completely new network design paradigm [49]. These new operators, and architectural approaches, cannot be transformed into dimensions recognized within DeepSniffer (see Section IV-B) which only consider standard operators within CNNs [3]. Therefore,


(a) DeepSniffer

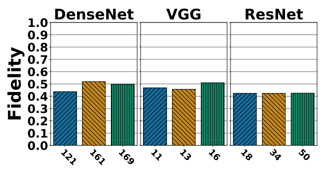

(b) KnockOffNets

Fig. 6: **Architecture family depth comparison.** Fidelity variance with models of different depths/parameters using same model architecture family.

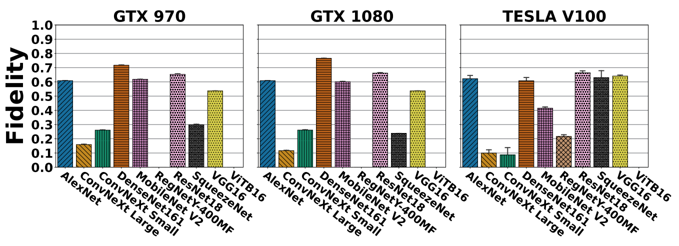
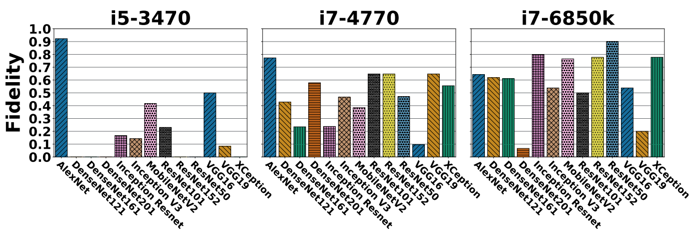


Fig. 7: **DeepSniffer extraction results** across GPU devices.


(a) Exact architecture

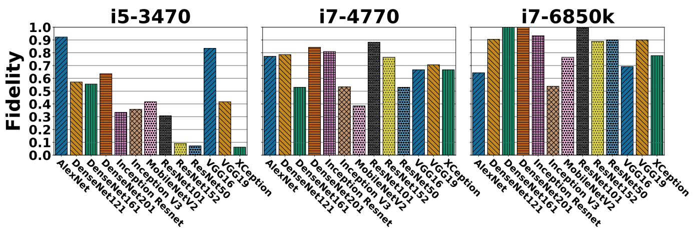

(b) Architecture family

Fig. 8: **DeepRecon model architecture & family prediction.**

DeepSniffer is not capable of predicting architectures to high success if they include these operators, and highlights a future revision of the attack for newer models. Moreover, in Figure 6a we observed that deeper models (models with more layers) within the same family resulted in reduced fidelity for VGG and Resnet families.

ML framework. We observed that models using the Tensorflow framework were ineffective at model extraction irrespective of target model architecture. From analyzing profiled data from both frameworks, the reason behind this result is due to TensorFlow generating additional kernel calls. The increased size of profiled data causes the trained DeepSniffer classifier to predict an architecture of greater length than expected. The existence of such noise can be attributed to low-level framework-level optimizations when kernels execute, in comparison to PyTorch. This finding indicates that DeepSniffer is framework specific, and therefore requires training on different frameworks to generalize extraction across these scenarios and further additional threat models to enable adversary knowledge of the system framework.

GPU architecture. Observing Figure 7, we can see variance in success over all architectures across different GPUs. GTX 970, and 1080 appear to show similar trends of results with fidelity varying slightly between GPUs. Across all GPUs, we continue to observe a clear attack ineffectiveness to target newer state-of-the-art models (ConvNeXt, RegNet, ViTB16). ViTB16 had little to no extraction success across all GPUs with the 1080 only reporting partial extraction. Interestingly, RegNetY-400MF was only partially extracted on the v100 (0.20) with the 970, and 1080 GPUs both failing to extract. ConvNext achieved the lowest extraction fidelity on the v100, while the 970 and 1080 incurred similar extraction results for ConvNeXt Small (0.26, 0.26) and ConvNeXt Large (0.15, 0.12), respectively. Such phenomena indicates that architecture attack susceptibility depends on the GPU used. These architectural differences affect the information leakage gathered by NVPROF, with optimizations causing recorded kernel metrics such as execution duration, read, and write amounts, to change between different systems.

C. DeepRecon

DeepRecon demonstrated distinct levels of success across the hardware platforms evaluated, as shown in Figure 8, with ResNet (0.81) and DenseNet (0.78) families highly successful on i7-4770 & i7-6850, and AlexNet (0.80) on all systems. We observe patterns of success across both hardware platforms, model architectures and their depths.

CPU architecture. We observed that systems with Intel Gen. 4 or later CPUs responded better to DeepRecon, with the i5-3470 generating symbol files 100x larger than the i7-4770 and i7-6850 and demonstrating noisy results in Figure 9. We attribute this to the caching policies of older CPUs being unfamiliar to the Flush+Reload exploit, causing multiple logs of a single symbol and disturbing the analysis. We observed in Figure 9 that newer CPUs (i7-4770 & i7-6850) reported a higher effectiveness in fingerprinting and classification for all model families compared to i5-3470. This is reinforced by classifier results from Figure 8, with the i5-3470, i7-4770 and i7-6850 averaging 0.35, 0.64 and 0.83 across all model families respectively. Findings also show cache size itself does not influence results, as Hong et al. [12] evaluated with a Gen 4. 4MB CPU successfully, smaller than i5-3470 (Gen3., 6MB). i7-6850 (0.83) reported significant improvements over i7-4770 (0.65), which we attribute to it being the most modern CPU (Gen 6.) evaluated, with more optimized caching policies that interfere with Flush+Reloads symbol detection less than i7-4770 (Gen 4.). This is strengthened by PCA analysis cumulative explained variance ratio of 92.20% for i7-4770 and 95.56% for i7-6850, (Appendix A.1). Of note is target model

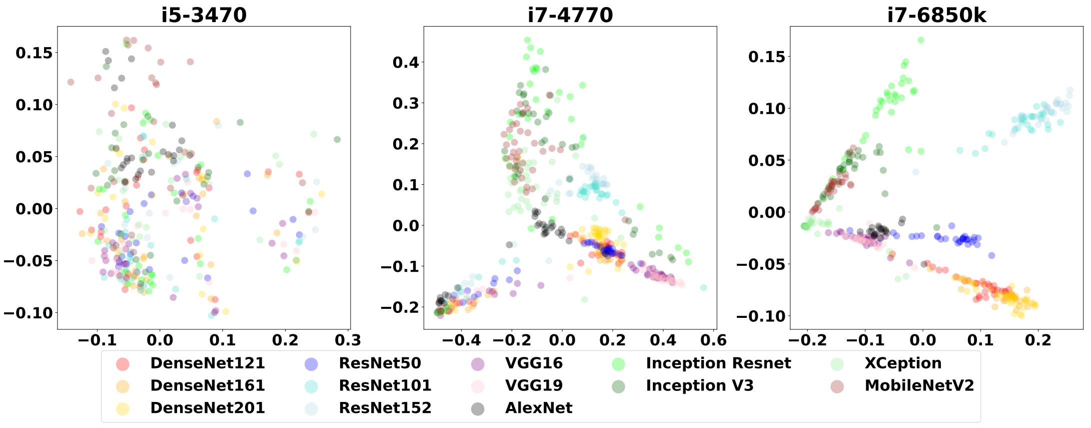


Fig. 9: **Principle Component Analysis (PCA) results for 19,500 inferences across three systems.** Clustering infers models demonstrate similar proportions of cached operators monitored by DeepRecon. Systems i7-4770 and i7-6850 exhibit distinctive model family fingerprints.

AlexNet achieved 0.80 classification accuracy on all platforms (Figure 8), with 0.85 on i5-3470. This finding indicates that models with highly homogeneous operators (AlexNet being primarily convs) can still be classified accurately with an imperfect Flush+Reload result as the proportions of logged operators are not as prevalent.

Model architecture & family. As shown in Figure 9, we observed target models within model families with distinct operator traits (ResNet; residual operators, DenseNet; dense blocks) successfully fingerprinted, but overlapped with other family members. This is shown further when models were often misclassified individually but with high success within a family. Analyzing i7-4770 & i7-6850, individual DenseNet models never achieved more than 0.52 classification accuracy for any depth (121: 0.47), (161: 0.20), (201: 0.51), however were correctly classified as in the DenseNet family with 0.78 accuracy across all depths. Similarly with VGG 16 and VGG 19, that achieved 0.31 and 0.48 individually, but familial success of 0.76 overall. This indicates that families with homogeneous operators built strong fingerprints, but using these fingerprints to train classifiers for specific depths is insufficient. The Inception Family (XCeption; DSC operator, Inception ResNet; residual operators, Inception V3) feature operators not included in other members of the family and hence built weaker fingerprints, intuitively showing lower average family success (0.65) on i7-4770 & i7-6850 than DenseNet (0.78), ResNet (0.81) and VGG (0.76).

D. Adversarial Attack Staging: Model Inversion

Extraction attacks have been identified as a means to stage further adversarial attacks [3], [6], [9]. Using PINCH, we conduct a case study whereby an adversary uses an extraction attack to stage a further model inversion attack.

Scenario setup. Two target models were stolen; Custom_MNIST, and Custom_CelebA trained on two datasets of differing complexity (MNIST 28x28, CelebA 218x178, both greyscale). The subsequent shadow model is exposed to a model inversion attack *MiFace* [39] whereby model information is used to generate images representative of target model classes. The MiFace attack was performed on the shadow model created by KnockOffNets at various query requests for MNIST [100 – 10000] and CelebA [1000 – 35000] to evaluate MiFace attack success at different stolen model fidelity. Following the threat model in Section III the adversary has access to an auxiliary dataset, and partial knowledge of the target model. Thus images representative of model classes; written numbers for MNIST and random faces for CelebA, are used for image initialization [31], [39]. Success of the MiFace attack when applied to a stolen model was evaluated by comparing generated images against images generated by the target model.

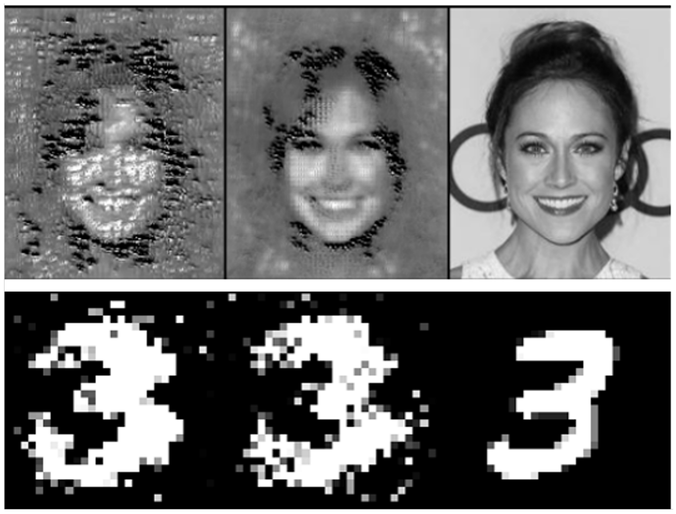
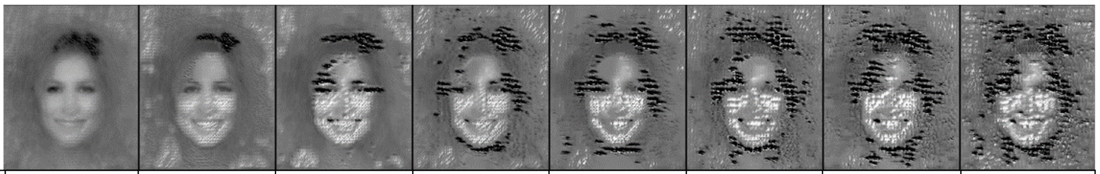
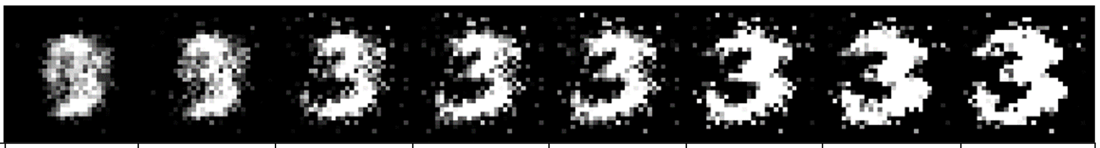


Fig. 10: **Model Inversion on stolen models.** Resulting model inversion attack upon a previously stolen model for CelebA (Top), and MNIST (Bottom). Stolen Model (Left), Target Model (Middle), Actual Image (Right)



Queries	1000	5000	10000	15000	20000	25000	30000	35000
Fidelity	0.58	0.72	0.78	0.80	0.82	0.83	0.83	0.83
PWCCA	0.50	0.48	0.45	0.43	0.45	0.45	0.45	0.46



Queries	100	250	500	750	1000	2500	5000	10000
Fidelity	0.21	0.48	0.70	0.79	0.82	0.90	0.93	0.95
PWCCA	0.33	0.24	0.17	0.16	0.13	0.11	0.10	0.09

Fig. 11: **Adversarial attack staging.** Model inversion on stolen models using KnockOffNets; CelebA (Top), MNIST (Bottom) exhibit varying query amount, model attack fidelity and PWCCA distance between stolen and target model.

Inversion success. Observing the result within Figure 10, it is apparent that using a stolen model the success of the inversion attack exhibits similar generated features in comparison to having direct access to the target model. Specially, we highlight that class features such as shape, are captured to a high degree of accuracy. For example, CelebA, captured dataset specific features such as; face shape, eye and mouth positioning, are inline with the actual image of the class. These captured features enable the adversary to gain additional knowledge about the target models and therefore exploit this knowledge to further augment the model inversion attack, or repeat extraction with a more tuned query set. Dataset complexity is an additional effecting inversion success, a less complex dataset such as MNIST shows very clear results, while CelebA is more complex introducing more noise due to a more granular greyscale, and the subject within the image including more distinctive features [35], [47].

Extraction sensitivity. We demonstrate further MiFace upon stolen models of varying query amounts (Figure 11). We observe that both datasets begin to show class features early with MNIST establishing a clear shape with 500 queries and similarly CelebA at 10,000 queries. Interestingly we see that the fidelity of the CelebA stolen model shows signs of overfitting with [25,000 – 35,000] queries, however displays different model inversion results despite the same fidelity score. The fidelity metric used in our experimentation, and within other extraction literature [6], [10], [19] may conceal the fundamental similarity of the models, therefore we introduce PWCCA to further understand model similarity.

Architecture similarity. While fidelity provides one aspect of similarity, we additionally applied Projection Weighted Canonical Correlation Analysis (PWCCA) [52] that measures similarity by calculating the distance between the calculated activation layers between models during inference. We observe that increasing query amounts for MNIST cause both fidelity and PWCCA distance to increase, achieving a maximum of 0.95 fidelity and a 0.09 PWCCA distance at 10,000 queries, denoting high similarity. CelebA also follows this trend until 15,000 queries where the PWCCA distance was at its lowest, however additional query amounts increase the PWCCA distance despite fidelity increasing. Therefore despite high extraction fidelity, models are able to make the same decisions but have fundamentally different execution from each other. This contrasting relationship between PWCCA distance and extraction fidelity highlights that these two metrics for model similarity do not always correlate, and thus a model in which makes the same predictions as a target model does not necessarily attribute to an underlying equivalent model.

VIII. Discussion

A. *Characteristics Affecting Extraction Attack Success*

Model architecture. Throughout our experimentation we evaluated a plethora of different target architectures ranging from RNNs to newly established transformers and discovered a clear variance in success between all evaluated attacks. DeepSniffer demonstrates how the fundamental execution of a model can vary the attack success, with newer state-of-the-art models including operators unexpected within the dimensions of the attack [3]. KnockOffNets however presented a form of abstraction away from the architecture through a API, but still demonstrated that differing model architectures can be intrinsically harder to attack due to their underlying architectural complexity and training requirements (Figure 3).

Dataset complexity. Observing attack success across datasets, it is apparent that complexity of training datasets for target models has a strong effect on attack success [18]. We observed that in ImageNet, the dataset which KnockOffNets was the least effective, dataset classes sizes had a considerable impact on extraction success. With knowledge of specific target model classes, it may possible for an adversary to run fewer queries yet extract particular target model classes of interest, which we aim to explore in future work.

B. DL Environment

Experiments reported variance in extraction success upon various evaluated DL environments (Figure 7, and 9). Original experiments in DeepSniffer used a k40 GPU [3], and when compared to our evaluated GPU hardware, it is intuitive that with DL hardware architectural improvements, such as optimizing kernel execution, would result in changes to DeepSniffer effectiveness. Such changes affect the success of DeepSniffer as gathered NVPROF data changes due to underlying kernel execution being different across three generations of GPUs. Furthermore, we observed DeepRecon also displayed varying extraction success across evaluated CPUs with different feature prevalence across evaluated environments (Appendix A.1). These observations uncover an intrinsic variation of extraction success across DL hardware driven by hardware specific extraction attacks.

C. Extraction Equivalency & Model Similarity

When measuring extracted model similarity with PWCCA distance (See Section VII-D), we observed that the relationship between PWCCA distance and extraction fidelity used in extraction attack literature does not always correlate, whereby the prediction fidelity between a target and stolen model can appear successful despite a high PWCCA distance (Figure 11). This observation highlights that both metrics, prediction fidelity and PWCCA distance, are metrics that do not fully capture nor measure the extraction success of a model. Prediction fidelity measures the success of the task in which the target model is used, while PWCCA measures model activation and thus differences in parameters of models. The disparity indicates the need to explore and propose new metrics to measure similarity of model prediction and execution.

D. Further Attack Staging

From conducting experiments, we observed that even with stolen models acquired from partially successful extraction attacks can be leveraged to stage further adversarial attacks such as model inversion to attain reasonable levels of success (0.7+). Less complex datasets such as MNIST are considerably less noisy compared to more complex datasets such as CelebA. However defined features can still be extracted by the inversion attack such as face shape, and gender (Figure 10). The ability to extract such features is especially concerning given the privacy related issues associated with specific types models of such as facial recognition, which allow an adversary to reverse engineer the classes to generate and expose images associated with real world people [31], [39]. This highlights that underlying hidden knowledge present within DL models can be extracted from stolen models, and therefore adversarial attack staging must be studied further.

IX. Related Work

There is a growing body of research dedicated towards the study of adversarial attacks against DL models [53] [54] [55] [56] [57] [58] [59] [60].

Extraction Attack Studies. Tramèr *et al.* [1] introduced the first extraction attack to extract target ML models exposed in online prediction APIs. Papernot *et al.* [61] proposed an avatar approach to extract a substitute DNN model for the purpose of generating adversarial examples. Different from [61], Joon *et al.* [62] designed an avatar based approach to train a meta-model to predict model hyperparameters. Junti *et al.* [63] developed a generic method for extracting DNN models by optimizing training hyperparameters and generating synthetic queries. Orekondy *et al.* [19] proposed a reinforcement learning based framework to improve query sample efficiency and performance. Hua *et al.* [64] first studied on reverse engineering of CNN on hardware accelerators, and investigated potential vulnerabilities in CNN accelerators in the context of model stealing. Wang *et al.* [65] provided hyperparameter stealing attacks to DL models.

Adversarial Attack Framework. Hussain *et al.* [17] presented a library allowing for black-box and label-only extraction, inference and inversion attacks on DL models. As an extended work of [17], Nicolae *et al.* [16] developed a more feature-rich library for evaluating and defending ML models to extraction, inference, inversion and poisoning attacks. Chen *et al.* [66] designed a Frank-Wolfe algorithm-based adversarial attack framework for white-box and black-box settings. Pearce *et al.* [67] provided a generic automation tool for testing the security of ML, which is a flexible environment, model and data agnostic framework. Liu *et al.* [18] proposed a holistic risk assessment of different inference attacks against ML models and established a threat model taxonomy. In contrast to these works, we propose an end-to-end automated extraction attack framework for studying DL models across heterogeneous hardware platforms, as well conduct an in-depth evaluation of extraction attack across a large number of different operational scenarios.

X. Conclusion

In this paper we have proposed PINCH: a framework for securing DL systems. The framework is capable of rapidly designing, deploying, and analyzing a large number of extraction attack scenarios across a multitude of different DL model archtiectures, datasets, hardware, and attack types. Results identify key characteristics influencing attack success, demonstrate that extraction attacks can reverse engineer AI models as high as 95% similarity, and show that even partially successful model extraction results in significant increase to the success of further adversarial attack staging. We hope our findings will be of interest and use to the secure AI community.

Acknowledgment

This work was supported by the EPSRC (EP/V026763/1, EP/V007092/1).

References

[1] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, page 601–618, USA, 2016. USENIX Association.

[2] Vasisht Duddu, Debasis Samanta, D Vijay Rao, and Valentina E. Balas. Stealing neural networks via timing side channels, 2018.

[3] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, and Yuan Xie. Deepsniffer: A dnn model extraction framework based on learning architectural hints. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, page 385–399, New York, NY, USA, 2020. Association for Computing Machinery.

[4] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, Dec 2018.

[5] Mingfu Xue, Chengxiang Yuan, Heyi Wu, Yushu Zhang, and Weiqiang Liu. Machine learning security: Threats, countermeasures, and evaluations. *IEEE Access*, 8:74720–74742, 2020.

[6] Yuankun Zhu, Yueqiang Cheng, Husheng Zhou, and Yantao Lu. Hermes attack: Steal dnn models with lossless inference accuracy, 2020.

[7] Junyi Wei, Yicheng Zhang, Zhe Zhou, Zhou Li, and Mohammad Abdullah Al Faruque. Leaky dnn: Stealing deep-learning model secret with gpu context-switching side-channel. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 125–137, 2020.

[8] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks, 2020.

[9] Xianmin Wang, Jing Li, Xiaohui Kuang, Yu an Tan, and Jin Li. The security of machine learning in an adversarial setting: A survey. *Journal of Parallel and Distributed Computing*, 130:12–23, 2019.

[10] 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 20)*, pages 1309–1326. USENIX Association, August 2020.

[11] Xiaoyong Yuan, Leah Ding, Lan Zhang, Xiaolin Li, and Dapeng Wu. Es attack: Model stealing against deep neural networks without data hurdles, 2020.

[12] Sanghyun Hong, Michael Davinroy, Yiğitcan Kaya, Stuart Nevans Locke, Ian Rackow, Kevin Kulda, Dana Dachman-Soled, and Tudor Dumitraş. Security analysis of deep neural networks operating in the presence of cache side-channel attacks, 2018.

[13] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s, 2022.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[15] Ben Keller, Rangharajan Venkatesan, Steve Dai, Stephen G. Tell, Brian Zimmer, William J. Dally, C. Thomas Gray, and Brucek Khailany. Deep learning inference accelerator with per-vector scaled 4-bit quantization for transformers in 5nm. In *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, pages 16–17, 2022.

[16] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian M. Molloy, and Ben Edwards. Adversarial robustness toolbox v1.0.0, 2018.

[17] Suha Hussain, Philip Wang, and Jim Miller. Privacyraven: Comprehensive privacy testing for deep learning, 2020.

[18] Yugeng Liu, Rui Wen, Xinlei He, Ahmed Salem, Zhikun Zhang, and Michael Backes. ML-Doctor: Holistic risk assessment of inference attacks against machine learning models. In *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, August 2022. USENIX Association.

[19] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models, 2018.

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

[21] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. 06 2017.

[22] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. AI accelerator survey and trends. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, sep 2021.

[23] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Š rndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Advanced Information Systems Engineering*, pages 387–402. Springer Berlin Heidelberg, 2013.

[24] Chawin Sitawarin, Arjun Nitin Bhagoji, Arsalan Mosenia, Mung Chiang, and Prateek Mittal. Darts: Deceiving autonomous cars with toxic signs, 2018.

[25] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, AISec '11, page 43–58, New York, NY, USA, 2011. Association for Computing Machinery.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[27] Honggang Yu, Kaichen Yang, Teng Zhang, Yun-Yun Tsai, Tsung-Yi Ho, and Yier Jin. Cloudleak: Large-scale deep learning models stealing through adversarial examples. 01 2020.

[28] The MITRE Corporation. MITRE ATLAS Adversarial Attack Knowledge Base, 2022. [Online; accessed 05-Sept-2022].

[29] Hoda Naghibijouybari, Ajaya Neupane, Zhiyun Qian, and Nael Abu-Ghazaleh. Rendered insecure: Gpu side channel attacks are practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 2139–2153, New York, NY, USA, 2018. Association for Computing Machinery.

[30] Ahmed Salem, Rui Wen, Michael Backes, Shiqing Ma, and Yang Zhang. Dynamic backdoor attacks against machine learning models. *CoRR*, abs/2003.03675, 2020.

[31] Zecheng He, Tianwei Zhang, and Ruby B. Lee. Model inversion attacks against collaborative inference. In *Proceedings of the 35th Annual Computer Security Applications Conference*, ACSAC '19, page 148–162, New York, NY, USA, 2019. Association for Computing Machinery.

[32] NVIDIA. Nvidia v100, Dec 2017.

[33] Karan Ganju, Qi Wang, Wei Yang, Carl A. Gunter, and Nikita Borisov. Property inference attacks on fully connected neural networks using permutation invariant representations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 619–633, New York, NY, USA, 2018. Association for Computing Machinery.

[34] Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, page 1485–1488, New York, NY, USA, 2010. Association for Computing Machinery.

[35] Shuo Yang, Ping Luo, Chen-Change Loy, and Xiaoou Tang. From facial parts responses to face detection: A deep learning approach. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 3676–3684, 2015.

[36] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[37] NVIDIA Developer Documentation. Cuda toolkit profiler documentation, May 2022.

[38] Yuval Yarom and Katrina Falkner. Flush+reload: A high resolution, low noise, l3 cache side-channel attack. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, page 719–732, USA, 2014. USENIX Association.

[39] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 1322–1333, New York, NY, USA, 2015. Association for Computing Machinery.

[40] Jordan Walke. React – A JavaScript library for building user interfaces, 2022. [Online; accessed 05-Sept-2022].

[41] Armin Ronacher. Welcome to Flask — Flask Documentation (2.2.x), 2022. [Online; accessed 05-Sept-2022].

[42] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. Wiley Publishing, 1996.

[43] Carl Meyer. Python Enhancement Proposal 405 – Python Virtual Environments, 2011. [Online; accessed 05-Sept-2022].

[44] Ari S. Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. In *NeurIPS*, 2018.

[45] François Chollet. keras, 2015.

[46] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2021.

[47] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[48] Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009.

[49] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces, 2020.

[50] 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, 2020.

[51] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2016.

[52] Ari S. Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation, 2018.

[53] Han Qiu, Tian Dong, Tianwei Zhang, Jialiang Lu, Gerard Memmi, and Meikang Qiu. Adversarial attacks against network intrusion detection in iot systems. *IEEE Internet of Things Journal*, 8(13):10327–10335, 2020.

[54] Nag Mani, Melody Moh, and Teng-Sheng Moh. Defending deep learning models against adversarial attacks. *International Journal of Software Science and Computational Intelligence (IJSSCI)*, 13(1):72–89, 2021.

[55] Kui Ren, Tianhang Zheng, Zhan Qin, and Xue Liu. Adversarial attacks and defenses in deep learning. *Engineering*, 6(3):346–360, 2020.

[56] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

[57] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *Ieee Access*, 6:14410–14430, 2018.

[58] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey. *arXiv preprint arXiv:1810.00069*, 2018.

[59] Bita Darvish Rouani, Mohammad Samragh, Tara Javidi, and Farinaz Koushanfar. Safe machine learning and defeating adversarial attacks. *IEEE Security & Privacy*, 17(2):31–38, 2019.

[60] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. A survey on adversarial attacks and defences. *CAAI Transactions on Intelligence Technology*, 6(1):25–45, 2021.

[61] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[62] Seong Joon Oh, Bernt Schiele, and Mario Fritz. Towards reverse-engineering black-box neural networks. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, pages 121–144. Springer, 2019.

[63] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N Asokan. Prada: protecting against dnn model stealing attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 512–527. IEEE, 2019.

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

[65] Binghui Wang and Neil Zhenqiang Gong. Stealing hyperparameters in machine learning. In *2018 IEEE symposium on security and privacy (SP)*, pages 36–52. IEEE, 2018.

[66] Jinghui Chen, Dongruo Zhou, Jinfeng Yi, and Quanquan Gu. A frank-wolfe framework for efficient and effective adversarial attacks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3486–3494, 2020.

[67] Will Pearce and Ram Kumar Shankar Siva. Ai security risk assessment using counterfit, 2021.

Appendix

TABLE A.1: **Hardware platforms evaluated against the DeepRecon side-channel attack, and PCA feature prevalence.** Well-fingerprinted systems i7-4770 and i7-6850 closely align on the most prevalent features (*Conv*, *Merge*, *ReLU*).

System	**Cache (MB)**		**Feature Prevalence**								**Simplified**
			convs	**fcs**	**softms**	**relus**	**mpool**	**apool**	**merge**	**bias**	
i5-3470	4	X	0.0003	-0.1643	-0.0583	0.6807	-0.6021	0.3475	-0.069	-0.1340	relus>mpool>apool
		Y	-0.0004	-0.4484	0.3590	-0.4292	-0.0839	0.5101	0.373	-0.2805	apool>relus>fcs
i7-4770	8	X	-0.8412	0.0808	0.0776	0.0119	0.0625	0.1082	0.5136	-0.0136	convs>merge>apool
		Y	0.3603	-0.0684	-0.0680	-0.6455	-0.0605	-0.0983	0.6524	-0.0718	merge>relus>convs
i7-6850	15	X	-0.7878	-0.0029	-0.0038	0.5642	-0.0054	0.0044	0.2462	-0.0148	convs>relus>merge
		Y	-0.1529	0.0051	0.0001	-0.5671	-0.0118	-0.0081	0.8060	-0.0712	merge>relus>convs