SoK: All You Need to Know About On-Device ML Model Extraction - The Gap Between Research and Practice

Tushar Nayan, Qiming Guo, and Mohammed Al Duniawi,
Florida International University; Marcus Botacin, *Texas A&M University;*
Selcuk Uluagac and Ruimin Sun, *Florida International University*

https://www.usenix.org/conference/usenixsecurity24/presentation/nayan

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.

SoK: All You Need to Know About On-Device ML Model Extraction - The Gap Between Research and Practice

Tushar Nayan*
Florida Intl. University

Qiming Guo*
Florida Intl. University

Mohammed Al Duniawi
Florida Intl. University

Marcus Botacin
Texas A&M University

Selcuk Uluagac
Florida Intl. University

Ruimin Sun
Florida Intl. University

Abstract

On-device ML is increasingly used in different applications. It brings convenience to offline tasks and avoids sending user-private data through the network. On-device ML models are valuable and may suffer from model extraction attacks from different categories. Existing studies lack a deep understanding of on-device ML model security, which creates a gap between research and practice. This paper provides a systematization approach to classify existing model extraction attacks and defenses based on different threat models. We evaluated well known research projects from existing work with real-world ML models, and discussed their reproducibility, computation complexity, and power consumption. We identified the challenges for research projects in wide adoption in practice. We also provided directions for future research in ML model extraction security.

1 Introduction

Internet of Things (IoT) applications have seen a growing trend in adopting on-device machine learning (ML). This allows for the execution of ML models directly on the IoT devices, enhances user experiences, optimizes system performance, and enables intelligent decision-making at the edge. This trend is now boosted by new AI chips available in the latest devices such as Apple's Bionic neural engine, NVIDIA's Jetson series, Google's Coral Dev Board, and Qualcomm's AI-optimized SoCs [1]. Nowadays, modern smartphones come equipped with on-device ML capabilities, and some even have dedicated AI accelerators for tasks such as image recognition, language translation, and predictive text input. Even Large Language Models (LLM), such as The MLC Chat, can run locally on Android devices with the latest Snapdragon chip [12].

Although executing ML tasks in the cloud for IoT applications offers convenience to developers, it comes with security implications. First, user-private data are shared with the cloud and can be leaked to network attacks, or other users sharing the cloud. Second, cloud-based ML can cause latency, which decreases the performance of real-time applications such as intrusion detection [35, 114, 115]. Whereas, on-device ML avoids sending private user data to the cloud, and provides real-time responses. For example, autonomous vehicles utilize on-board ML models for real-time object detection, pedestrian recognition, and decision-making without constant reliance on cloud-based services. Smart speakers such as Amazon Echo or Google Home use on-device natural language processing (NLP) models to understand and respond to voice commands without relying entirely on cloud processing [127]. However, on-device ML introduces a new security challenge—the valuable, proprietary, and possibly security-critical ML models are now deployed on user-end devices, and adversaries may attempt to extract these models from devices, causing model stealing or extraction attacks [112].

The extraction of ML models from IoT applications can lead to both financial concerns and security implications. Intuitively, commercial ML models are the core intellectual property of model vendors. A leaked model gives away the research and development cost of the model owner, and may result in unethical business competition [112]. Further, IoT applications handle sensitive data, ranging from biometric information to personal preferences, and a leaked model poses a direct threat to compromise these sensitive data. This is because a leaked model can equip attackers with grey-box information (e.g., partial models, recovered models), or even white-box information (e.g., plaintext models), and allow attackers to perform further attacks more conveniently and accurately. For example, attackers can craft adversarial examples to deceive IoT users in decision-making, perform membership interference attacks to reveal sensitive training data or even data poisoning and model poisoning attacks [46, 74, 113] if the models need to be updated later. These attacks can lead to privacy violations and unauthorized use of the data collected by IoT devices [95]. In addition, ML models can be used for safety- and security-critical functions (e.g., climate forecasting, face authentication), and leaked models will allow attackers to bypass the safety or security check [133].

*The first two authors contributed equally to this work.

In performing and defending model extraction attacks, respectively, existing research has explored various techniques. From the attacker's perspective, existing works have explored decomposing [32] and decompiling techniques [42], runtime memory extraction [123], side-channel attacks [109], and data mining techniques [101] to extract valuable ML models. To defend against model extraction attacks, defenders have explored Advanced Encryption Standard (AES) [58], Homomorphic Encryption (HE) [132], Trusted Execution Environment (TEE) [45], data transformation [35], and various learning algorithm-based protection techniques [88]. While there have been advances in model extraction security, existing efforts have often been ad-hoc and fragmented. Together with the rapid advances in ML, it becomes challenging for researchers and practitioners to have a comprehensive understanding of all aspects of model extraction attacks. This gap hinders the development of robust security techniques, including issues related to privacy, optimization, and computation.

This research aims to contribute to building greater trust in IoT devices adopting on-device ML, and relieving privacy concerns in leaking sensitive user information. Specifically, we design a general framework to systematize existing studies in model extraction attacks and defenses. We define four categories of threat models, including app-based, device-based, communication-based, and model-based attacks and defenses. We focus on studies investigating Android-based applications and Arm-based devices, due to their popularity in serving on-device models [8]. Our investigation highlights a significant issue within the research community: many research projects, although innovative, may not be open source, or cannot be reproduced in practice, and the amount of effort in project deployment and maintenance prohibits the wide adoption of these projects. We further chose representative projects from each category of threat models and evaluated them with real-world ML models. We found that when research projects are applied to real-world ML models, they may not produce as good performance as reported in the paper. The computation complexity and power consumption can also be high. Our code in reproducing and evaluating existing projects is open-source[1]. Last, we point out the open challenges in applying these projects to practice, and future research directions for researchers and practitioners to enhance on-device ML security. Our investigation can be beneficial to myriad technologies ranging from ML training and inference, IoT and Android app development, IoT sensing, data communication, and secure memory design and development.

Note that although ML models can suffer from a range of attacks, including model pollution attacks [33], model stealing attacks [50], membership inference attacks [107], etc., ***this paper focuses on ML model extraction or stealing***.

[1] https://github.com/sys-ris3/ML_Extraction_Sok

2 Model Extraction: Security Design

This section provides context for later discussion. We give a high-level overview of the security design of model extraction techniques. Our goal is to describe the threat models, the security techniques, the methods, and the effective stages of model extraction. These factors will contribute to a general framework to analyze the challenges in preventing model extraction attacks.

2.1 Threat Models

We categorize existing works into four distinct types of threat models, from attacker's and defender's perspectives, respectively.

2.1.1 Attacker's perspective

App-based attacks: attackers assume they can gain access to the application files either through the public application marketplace, or through a vulnerability within the IoT devices. Attackers will perform application de-packaging or decompiling [112], and extract the model files [41, 80, 112, 129]. These model files can be either usable plaintext models or encrypted and obfuscated models. Attackers can directly utilize plaintext models or employ decryption and deobfuscation tools to reverse the plaintext model.

Device-based attacks: attackers assume they can access the IoT devices and gain access to the memory [112]. Attackers can either force a vulnerable application to launch and load ML models into memory or consistently scan the memory to wait for models to be loaded. Attackers assume that, no matter how many times the models have been encrypted/obfuscated, plaintext copies of ML models will eventually be loaded into memory so that the vulnerable application can actually use the models. Attackers also assume to have knowledge of the in-memory format of models. This is possible for well known frameworks, such as TensorFlow and PyTorch.

Communication-based attacks: attackers can intercept communication between various memory regions and hardware architectures on an IoT device. This communication data can encompass application runtime data, hardware usage records [59, 83], memory usage [67], electromagnetic [134] and power-related data [128]. Attackers assume to leverage these data to recover partial or complete details of ML models, including their structure and weights, or infer functionally equivalent models.

Model-based attacks: attackers assume to be able to send (selective) input queries to the models and observe the ML inference results. Attackers assume the ability to use the pairs of (sent data, and received inference results) to assess the functionality of models and fine-tune the data to send in subsequent steps. Attackers then go through the above process and train substitute models [96]. Attackers sometimes assume

to leverage pre-trained models to improve the accuracy of training substitute models or employ large-scale datasets or distributed methods to send query requests to the target models. Attackers can even use intelligent query agents and more advanced techniques to enhance query efficiency [86].

2.1.2 Defender's perspective

App-based defense: defenders assume that attackers can get access to an app package, and extract the model files from the app package. Defenders apply techniques, including encryption, obfuscation, or customized protection to model files in an app package. The files can contain model layers, weights, and other configurations.

Device-based defense: defenders assume that attackers can extract models from memory in plaintext. Defenders apply device-based protection, such as secure hardware, to prevent arbitrary memory access. Defenders can also customize hardware to support computation on encrypted data, so that memory extraction will not reveal plaintext models.

Communication-based defense: defenders assume that attackers cannot directly extract models from memory but can sniff the communication between two memory components. Defenders apply data transformation, encryption, and randomization techniques to prevent side-channel information leakage and enable further calculation based on the transformed data in the memory components.

Model-based defense: defenders assume that attackers can send crafted input to ML models and leverage the input and output pattern to train equivalent student models. Defenders apply weight obfuscation, misinformation and differential privacy to increase the effort of attackers in training equivalent student models.

2.1.3 Motivating Example

The threat models can exist alone or coexist with others. A powerful and motivated attacker can explore the four categories of threat models altogether. Now let us consider a real-world e-health application that runs on an Android phone and uses on-device ML in diagnosing skin cancer, diabetes, and hypomnesia [40]. The model takes input from sensors measuring human body data, including electrocardiography (ECG) data, Electroencephalogram (EEG) data, and polysomnography (PSG) recordings. The data are user-private and can be personal identifiable information (PII). Figure 1 describes the process of a powerful attacker (Alice) in extracting this ML model. A defender (Bob) tries to stop the model extraction attack throughout the process. Alice begins by downloading the e-health app from Google Play and using `apktool` to depackage the app. Alice expects to extract the plaintext model from the package (e.g., an App-based attack), but Bob has encrypted the model before shipping the app to Google Play (e.g., an App-based defense). Alice fails and continues in Step (1). Alice tries to run the app on a phone, invoke the ML task, and use the `Frida` tool to extract the model from memory (e.g., a Device-based attack). However, Bob has enforced the model to run only in the Trusted Execution Environment (TEE) (e.g., a Device-based defense). Alice fails again and continues in Step (2). Alice tries to sniff the communication between the secure and normal world to recover the model (e.g., a communication-based attack), but Bob has enabled random shuffling to avoid useful side-channel information (e.g., a communication-based defense). Alice fails again and continues in Step (3). Alice tries to query the model with specific input, obtain inference output, and use the input, output pattern to train a highly similar student model with less effort (e.g., a model-based attack). However, Bob has enabled adaptive misinformation to make the patterns misleading (e.g., a model-based defense), and the game continues in Step (4).

Alice and Bob's story can be executed across various IoT devices, encompassing smartphones, smart home devices, and wearable gadgets.

Privacy Concerns. The extraction of on-device ML can worsen privacy issues within IoT devices [22, 34], and bring new attacks including data poisoning attacks, model poisoning attacks, and membership inference attacks. Without the extracted models, attackers can only perform attacks based on black-box information. With the extracted models, under different threat models, attackers can obtain white-box, or grey-box information, with plaintext models, partial models, or surrogate models with high fidelity. The additional information makes it more convenient to perform attacks.

2.2 Security Techniques

Based on the threat models, attackers and defenders may explore different security techniques to perform or prevent ML model stealing attacks. The following describes the common techniques in different categories.

2.2.1 Attacker's perspective

Decomposing and Decompiling: these methods unpacks application packages to extract useful files (e.g., ML models) from the sources. Popular tools include apktool [2], Jadx [9], IDA Pro [7], and so on.

Memory analysis: these methods access device memory and obtain memory buffers containing model layers and weights. Popular tools include Frida [5], GDB [6], memfetch [11], and so on.

Side-channel attacks: these methods exploit indirect (side-channel) information that can be used to infer ML inference data. The side-channel information includes power consumption, electromagnetic (EM) radiation, cache accesses, timing, and so on. Popular tools include ChipWhisperer [4], SCALab [14], CacheAudit [3], etc.

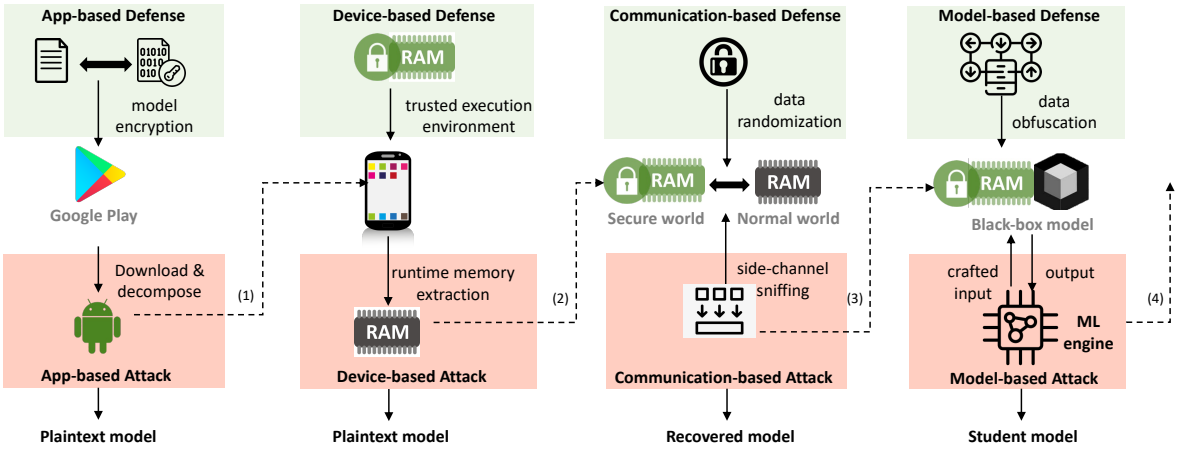

Figure 1: Example of threat models from attacker's perspective and defender's perspective. (*Note: (1) to (4) illustrates the cases when model extraction attacks fail in that category, respectively*).

Algorithm-based stealing: these methods monitor the input and output pairs of ML models, and apply statistical or ML-based algorithms to steal model parameters, or train highly similar substitute models. Popular methods include gradient-based techniques [49], active learning [119], generative techniques [62], and so on.

2.2.2 Defender's perspective

Encryption and obfuscation: these methods prevent unauthorized access or modification to ML models. Techniques include Advanced Encryption Standard (AES) [16], Homomorphic encryption (HE) [70], Multi-party computation (MPC) [29], Differential Privacy (DP) [18], Code obfuscation [63], multi-party computation (MPC) [29], string encryption, binary encoding, and so on.

Secure hardware: these methods allow the storing and execution of ML models in a secure environment. Popular secure hardware includes Trusted Execution Environment (TEE) [111], Secure Element (SE) [93], Physically Unclonable Function (PUF) [48, 57], and so on.

Pattern randomization: these methods reduce the chance for side-channel-based model extraction attacks by adding noises to ML inference or randomizing patterns that may coexist with ML inference. Popular techniques include oblivious shuffling, Address Space Layout Randomization, dummy memory access [82, 85], and so on.

Data and algorithm diversification: these methods increase the complexity of algorithm-based stealing, and prevent attackers from further model training. Popular techniques include model compression [56, 103, 116], adaptive misinformation [72], data augmentation [78, 84], adversarial training [23, 108, 137], ensemble learning [37, 47, 54], randomization [51] and so on.

3 Systematization

We systematize different categories of ML model extraction attacks and their corresponding defenses on IoT devices. Within each category, we delve into existing works, with their objectives, the security techniques involved, and challenges in adopting them in practice. Table 1 and Table 2 list all the papers discussed in this section.

3.1 App-based Attacks and Defenses

3.1.1 Attacker's perspective

App-based ML model extraction was introduced by Xu et al. in 2019 with the goal of understanding the usage trend of on-device ML [129]. Xu developed DL Sniffer to download and decompose app packages to extract ML models. With a collection of 16,500 mobile apps from Google Play, DL Sniffer successfully extracted 211 deep-learning models. This work unveiled that only 39.2% of the models were obfuscated and 19.2% were encrypted, while the rest of the models can be directly extracted in plaintext as a whole. Later, ModelXray [112] was created with the explicit aim of comprehensively examining the security aspects of on-device ML models. ModelXray investigates a larger app collection, with apps from three popular app stores. The results showed that the number of on-device models that are not protected at all can range from 34% to 76%, across the three stores. ModelXray also identified model reuse scenarios. Astonishingly, one model can be reused by over 80 apps, while such apps have been downloaded millions of times. ModelXray reveals that on-device model extraction attacks can be more severe than once thought. A few other works [41, 68] have explored app-based model extraction attacks in similar ways as used by DL Sniffer and ModelXray, while their goal is to perform

Table 1: Existing works on model extraction attacks.

Title	Category	Target	Method	Open-source	Reproduced	ML Framework
First Look	App	Whole	Decompile	Yes	Yes	Multiple
SmartAppAttack	App	Whole	Decompile	Yes	Yes	Multiple
Mind'21	App, Device	Whole	Decompile, mem. searching	Yes	Yes	Multiple
Understanding'22	App, Device	Whole	Decompile, API hooking	No	N/A	Multiple
DeepRecon	Comm.	Arch.	Cache (Fl.&Re.)	Yes	No	TensorFlow
CSI NN	Comm.	Arch.,Layer,Weight	timing and electromagnetic	No	N/A	General
Cache Telepathy	Comm.	Arch.	Cache (Pr.&Pr.,Fl.&Re.)	No	N/A	General
Open DNN box	Comm.	Arch.,Weight	Power Feature	No	N/A	General
Reverse CNN	Comm.	Arch.,Weight	Memory Access	No	N/A	General
GANRED	Comm.	Arch.	Cache Attack	No	N/A	General
DeepEM	Comm.	Arch.,Layer,Weight	EM Attack	No	N/A	General
StealingNNTiming	Comm.	Arch.,Weight	Timing Attack	No	N/A	General
HuffDuff	Comm.	Arch.,Weight	Timing Attack	No	N/A	General
Hermes Attack	Comm.	Whole Model	PCIe traffic	No	N/A	TensorFlow
Leaky DNN	Comm.	Arch.	GPU Context-Switching	No	N/A	TensorFlow
ScanChainSteal	Comm.	Model Weight	Scan-chain Infrastructure	No	N/A	TensorFlow
DeepSniffer	Comm.	Model Arch.	Memory, Bus snooping	Yes	Yes	PyTorch
DeepSteal	Comm.	Functionality	Memory Access (rowhammer)	Yes	Yes	PyTorch
ML-Doctor	Model	Model Weight	Inference Attacks	Yes	Yes	Pytorch
Hyperparameters	Model	Hyperparameters	Hyperparameter Stealing	No	N/A	General
Reverse BlackBox	Model	Arch., Optm.,etc	Adversarial Example	No	N/A	Pytorch
Activethief	Model	Model Weight	Active Learning	Yes	No	TensorFlow
ML-Stealer	Model	Functionality	Prediction Stealing	No	N/A	General
KnockoffNets	Model	Functionality	Functionality stealing	Yes	Yes	Pytorch
SimulatorAttack	Model	Functionality	black-box attack	Yes	Yes	TensorFlow,Pytorch

Note that Pr.&Pr. means Prime+Probe, and Fl.&Re. means Flush+Reload.

Table 2: Existing works on model extraction defense.

Title	Category	Target	Method	Open-source	Reproduced	ML Framework
TFSecured*	App	Whole	Encryt.	Yes	Yes	TensorFlow
MindSpore*	App,Model	Whole	Encryt.,Obfu.,DP	Yes	Yes	MindSpore
Knox*	App	Whole	Encryt.	Yes	Yes	Multiple
MACE*	App	Whole	Obfu.,Convert	Yes	Yes	TensorFlow,Caffe,ONNX
m2cgen*	App	Whole	Convert	Yes	Yes	Multiple
MindDB*	App	Whole	Convert	Yes	Yes	Multiple
MMGuard	App	Whole	Encrypt, node insertion	Yes	Yes	TensorFlow
MyTEE	Device	Whole	TEE	Yes	No	General
SANCTUARY	Device	Whole	TEE	Yes	Yes	General
OMG	Device	Whole	TEE	No	N/A	TFLite
DarkneTZ	Device	layer,output	TEE	Yes	Yes	General
Graviton	Device	Whole	TEE	No	N/A	Caffe
ObfuNAS	Comm.	Arch.	Obfu.	Yes	Yes	PyTorch
ShadowNet	Device,Comm.	layer,weight	Transform	Yes	Yes	Darknet, TFLite
Slalom	Comm.	layer,weight	Transform	Yes	No	TensorFlow
E2DM	Comm.	Whole	HE	No	N/A	TensorFlow
NPUFort	Comm.	Weight	Secure Hardware	No	N/A	General
NeurObfuscator	Comm.	Arch.	Obfu.	Yes	Yes	PyTorch
Mitigating'19	Comm.	Functionality	Oblivious shuffle, ASLR, etc.	No	N/A	General
NNReArch	Comm.	Arch.	EM Obfu.	No	N/A	General
Misinformation	Model	Weight	Adaptive Misinformation	Yes	Yes	PyTorch
PredictionPoison	Model	Weight	Perturbation	Yes	Yes	PyTorch
PRADA	Model	Weight	Extraction Detection	Yes	Yes	PyTorch
SteerAdversary	Model	Weight	Gradient redirection	Yes	Yes	PyTorch
LDA-DP	Model	Weight	DP	No	N/A	General

*Note: title with * means the project is maintained by industry community*

further attacks, such as adversarial attacks, to a portion of the extracted models.

In general, app-based model extraction attacks target the extraction of a whole model and can extract models on a large scale. Attackers leverage the weaknesses that (1) app packages are easily accessible and prone to reverse engineering, and (2) model files are not adequately protected through encryption or obfuscation measures. Next, we describe existing works that may potentially defend against the weaknesses.

3.1.2 Defender's perspectives

App-based model extraction defenses aim to obscure easily readable machine learning models into something highly complex and difficult to interpret. Thanks to advancements in computing power and model compression techniques, encrypting and decrypting model files on IoT devices has become increasingly feasible. For example, Samsung's Knox [106] employs AES encryption to secure various model formats. The models will be shipped in encrypted format with the app package. When ML inference starts, the models will be decrypted in the device memory, and start working as regular plaintext models. In IOS, Apple's CoreML [69] provides AES solutions to its models in a similar way. Outside of the industry community, researchers have developed specialized AES methods [16] that run faster with TensorFlow and provide bundling with app package. MMGuard [65] uses a secret token and inserts an extra input node to the model. This makes a stolen model to behave incorrectly without the secret token. It is worth noting that even traditional encryption algorithms are highly effective in preventing large-scale model extraction attacks.

Besides encrypting models into unreadable formats, obfuscation, and code conversion are investigated to produce models that are readable but difficult to interpret. For example, MindSpore [17] employs dynamic obfuscation to enable apps to perform regular ML inference, but will produce meaningless outputs unless the obfuscation scheme is known to the apps. MACE [10] opts for code conversion, turning models into C++ code representations. M2cgen [28] is capable of transforming a variety of machine learning models into more than ten types of code, including Java, C, Python, Go, and so on, with zero dependencies. NCNN [15] converts model files into binary formats and removes all text, creating a barrier against decomposing-based reverse engineering. Finally, MindsDB [90] converts model files into a database which facilitates easier model deployment. It also supports obfuscating models as a regular database file, making it less vulnerable to attacks targeting popular ML frameworks. In general, app-based defense solutions are effective in mitigating the weaknesses of app-based attacks, by preventing large-scale model extraction. However, as shown in Table 2, such solutions are often tied to an ML framework or a vendor-specific product, limiting the adoption to a broad range of apps and devices. Further, extra effort will be needed from the app/model vendors (e.g., in encrypting the models), the user/device (e.g., in decrypting the models), and key management between the two parties. The decryption process for the model may slow down app performance, depending on device settings.

3.1.3 Findings

We found that app-based model extraction defense solutions mostly come from the industry communities, which indicates that app-based model extraction attack is indeed a severe problem in practice.

We found that, from the attacker's perspective, attackers may not be able to directly use the extracted models for several reasons. First, attackers have to understand the input and output of the models. For example, the input is an image of a bird and the output (i.e., labels) is "bird". Attackers will need to understand the semantics of the app code to obtain such information. Second, the models can be black-box, so the model functionality is unknown to the attackers. The best chance for attackers is to hope the models come from popular ML frameworks (e.g., TensorFlow, PyTorch) and serve common functionalities (e.g., card recognition) so that the models are more likely to be understood and directly used. Last, the models may have been obfuscated or encrypted, so attackers will need to explore de-obfuscation or decrypt techniques to use the models.

We also found that app-based defense alone is not sufficient. For example, the key management mechanism can be vulnerable. Previous work [112] has found the key (or license) to be shipped together with the app package, and attackers can extract the key from the app and use that to decrypt the model. Even if the keys are not accessible, attackers can perform dynamic analysis of the app on a device, and bypass the encryption and obfuscation that have been described. Next, we introduce such cases as device-based attacks.

3.2 Device-based Attacks and Defenses

3.2.1 Attacker's perspective

Device-based ML model extraction attack is explored to bypass app-based model encryption. ModelXtractor [112] first demonstrated the case of extracting encrypted models from device memory in plaintext. The assumption is that for encrypted models to be used by the app, a plaintext model will eventually be loaded into memory. ModelXtractor uses app instrumentation with four types of instrumentation strategies to dynamically find the memory buffers where a (decrypted) model is loaded and accessed by the ML frameworks. ModelXtractor was able to extract models even though three layers of encryption had been applied.

Similar to ModelXtractor, Deng et al. developed AdvDroid [41] to hook API calls that relate to model loading

and extract models from memory. AdvDroid leverages program slicing to search the app code with relevant functions of model inference, and constructs a sequence of UI operations that are associated with the code. AdvDroid focuses on the models for image classification and object detection, with the assumption that the majority of the models are related to these two functions.

In general, device-based model extraction attacks cannot be performed on a large scale. Some apps are complicated and will require authentication and registration to start the ML model inference. Therefore, it depends on some “luck" to trigger a required ML model and extract it from memory.

3.2.2 Defender’s perspectives

Device-based model extraction protection aims at defending against in-memory model extraction and related attacks. Graviton’s architecture [120] was first proposed to support trusted execution environments on GPUs. Graviton enables applications to offload security- and performance-sensitive kernels and data to a GPU, and execute kernels in isolation from other code running on the GPU and all software on the host. Graviton requires hardware changes to integrate into existing GPUs. To relieve the hardware changes, SANCTUARY [31] presents the first security architecture that enables the execution of security-sensitive apps in TrustZone’s normal world with strongly isolated compartments. SANCTUARY is a generic Trusted Execution Environment (TEE) solution that paves the road for protecting on-device ML models. Based on SANCTUARY, OMG [27] provides a prototype implementation of user-space enclaves to protect both client data and model privacy for TensorFlow Lite models. OMG can protect the whole model while lacking support for GPU acceleration and easy adaptation. To solve the problem, LEAP [110] was developed to offer hardware-assisted secure IO and flexible resource management. LEAP also presents a developer-friendly TEE programming interface for app developers to enable memory-protected ML models. To protect partial models, DarkneTZ [91] allows selected layers (e.g., non-linear layers) to run inside the TEE to protect sensitive parts of the model from being stolen. DarknetTZ does not support secure GPU acceleration. ShadowNet [111] also runs sensitive layers inside TEE but extends the support to GPU, with a smaller TCB compared with DarknetTZ. In a different line of research, Mgx [66] introduces a memory protection unit (MPU) design through dynamically adjusting memory access permissions based on the execution context of the DNN model. MyTEE [55] introduced a TEE environment on embedded devices without assuming the existence of TrustZone hardware extensions.

In general, existing works in device-based defenses focused on TEE-based solutions. As Table 2 shows, the protection target ranges from a whole model, layers of the model, and model input/output. The deployment requires either changes to the hardware, or implementing the ML libraries in TEE. TEE-based solutions usually incur significant runtime inference overhead, which prohibits their wide adoption in protecting ML models in practice. The integration of GPU is promising while more research in this direction is warranted.

3.2.3 Findings

We found that attacks in this category are semi-automatic, and are hard to perform on large-scale. For attacks to be successful, attackers will need a deep understanding of the ML library, and the formats of ML models when they are stored in the memory. This excludes in-memory extraction of models from uncommon frameworks. Even after the models are extracted, the models may not be directly useful, as mentioned in the findings of app-based attacks and defenses.

We also found that device-based defense alone is not sufficient in preventing ML models from extraction. For example, TEE is usually limited in memory size and slow in ML inference speed, so large models cannot fit in TEE. When model partitions are needed, communication data between the secure world and the normal world will leak model information as well. Next, we describe attacks in this category.

3.3 Comm-based Attacks and Defenses

3.3.1 Attacker’s perspective

Communication-based attacks aim at stealing or recovering ML models by collecting data from or between different memory components during ML inference. The communication data can include direct model information or side-channel information. Direct model information leakage is only briefly mentioned in a few works [111, 117]. Most studies focus on side-channel data to extract ML models, including cache information, electromagnetic (EM) radiation, timing information, power consumption, and so on.

In cache and memory-based side-channel attacks, existing works have explored non-invasive and passive ways. For example, Hua et al. [67] designed the first attack to steal CNN architectures while it requires the attacker to monitor all of the memory addresses accessed by the victim. DeepRecon [60] reconstructs the architecture of a victim model by analyzing cache access times using Flush+Reload. It passively observes model-related function invocations, and reconstructs the victim’s entire network. Cache Telepathy [130] is similar but can obtain more detailed hyper-parameters, such as the number of neurons in fully connected layers and filter size in convolutional layers. DeepRecon and Cache Telepathy require a shared main memory segment between attacker and victim, while GANRED [83] relieves the requirement with cache timing side-channel. These works provide no direct evidence between the statistics and the attack’s effectiveness. To understand this issue, DeepSniffer [64] learns the relation between

extracted architectural hints (e.g., volumes of memory reads/writes obtained by side-channel or bus snooping attacks) and models internal architectures, to achieve more efficient attacks. Besides passive attacks, researchers have actively stimulated bit-flipping in the memory storing the models, and derive approximate models [104].

In timing-based side-channel attacks, Duddu et al. [44] aim to extract a black-box Neural Network and infer the depth of the network. They leveraged reinforcement learning-based optimization to reduce the search space and reconstruct a substitute architecture. These techniques do not work well when the model is sparse in either weights or activation layers, because off-chip transfers no longer correspond exactly to layer dimensions. To mitigate the sparsity problem, HuffDuff [131] leverages the boundary effect present in CONV layers, and the timing side channel of on-the-fly activation compression to extract black-box models.

With other side-channel information, CSI NN [26] uses timing and electromagnetic (EM) emanations to recover multilayer perceptron and convolutional neural networks. DeepEM [134] is similar but estimates the weights through margin-based adversarial active learning. Open DNN Box [128] derives the power signature of embedded AI devices, and employs machine learning techniques to discern the model architectures of embedded AI devices. It further utilizes the concept of model sparsity to deduce the model parameters.

Hermes Attack [140] uses plaintext Peripheral Component Interconnect Express (PCIe) traffic to leak the whole DNN model. The stolen DNN models have the same hyperparameters, parameters, and semantically identical architecture as the original ones. The methodology is supposed to be effective for all models. However, it depends on the buffer size of the snooping device. Leaky DNN [124] utilizes context-switching penalties to exploit GPU side channels and introduced the MosCons attack prototype, which enables a spy to obtain finer-grained information of DNN ops and hyperparameters. Scan chain attacks [100] reveal that course-grained scan-chain access to non-linear layer outputs is sufficient to steal ML models.

In general, the side-channel information to extract ML models can come from many sources, and be passive or active. These attacks can produce high accuracy and fidelity to the original models and are capable of recovering black-box models. As expected, such attacks cannot be performed to extract models on a large scale. In addition, the attacks may not always succeed because model information may have been transformed through encryption, obfuscation, and random shuffling, so the victim models do not serve as ground truth anymore. Next, we describe existing works with such defenses.

3.3.2 Defender's perspective

Communication-based model extraction defenses aim to mitigate model-related data or side-channel data leaked from TEE or hardware accelerators or between secure and normal worlds in TEE. To prevent communication data leakage between secure and normal worlds, existing works are based on the assumption that sensitive layers run in the secure world, while the rest of the layers run in the normal world. Based on this assumption, Slalom [117] first splits models into linear and nonlinear layers and secures nonlinear layers in SGX. Slalom uses masked input with encryption and randomization to the normal world layers and can protect partial model layers and user input to the models. To further protect the model weights from extraction, ShadowNet [111] applies linear transformation and random noises to secure the intermediate results transferred between the normal world and the secure world.

To prevent side-channel information leakage from hardware accelerators, NPUFort [122] protects general DNN accelerator from side-channel information leakage NeurObfuscator [77] prevents exact architecture extraction by obfuscating the dimension and number of the DNN layers. Liu et al. [82] proposed to prevent architecture extraction through memory access pattern analysis, including oblivious shuffle, address space layout randomization, and dummy memory accesses. Luo et al. [85] proposed to increase the difficulty of extracting DNN architectures through tensor execution scheduling, as a way to prevent EM based side-channel leakage. These methods require low-level modification and are limited by the execution environment. ObfuNAS [138], instead, obfuscates models via Neural Architecture Search (NAS) to defend against architecture extraction attacks without requiring specialized hardware.

Other secure ML systems use cryptographic primitives, such as E2DM [70], using HE for encrypted data and encrypted models. However, the performance overhead is significant and it is far from practical to apply these theoretical approaches to real-world applications.

In general, communication-based defense solutions can mitigate side-channel-based model extraction attacks. However, similar to device-based defenses, these approaches usually require low-level hardware changes or the development of a full-stack ML inference framework. This limits the protection to a certain type of hardware (e.g., TEE, GPU), or a certain ML framework (e.g., TensorFlow). The incurred deployment effort also prohibits the wide adoption of the existing solutions. Due to the ad-hoc nature of side-channel attacks, it is also challenging to propose a generic solution to prevent model extractions from different types of side-channel information.

3.3.3 Findings

We observed that the number of side-channel-based model extraction attacks exceeds the available defenses designed

to counter them. The extracted (or recovered) models can achieve high accuracy and fidelity. However, for the attacks to be successful, attackers will need to be co-located with the victim processes or share the same memory regions with the victim. The monitored data will need to be recorded in log files or be frequently transferred to the attackers. All of these increase the chance to expose the attacks and question the possibility of real-world attacks in practice. We are not aware of industry defense solutions in this category of threat model.

We also found that communication-based model extraction defense alone is not sufficient. For example, attackers do not need direct or side-channel model information to recover a model. Instead, attackers query the model with special input and obtain the inference result as output. but can. With the input, output pairs, attackers can train a substitute (or student) model of the victim model. Next, we introduce existing works with such attacks.

3.4 Model-based Attacks and Defenses

3.4.1 Attacker's perspective

Model-based extraction attacks assume that attackers use pairs of inputs, outputs to train substitute models that are highly similar with the victim models. Based on this assumption, existing works have explored to (1) achieve high model similarity with minimal accuracy loss, (2) eliminate the requirements of prior knowledge of the victim models, (3) optimize query efficiency and effectiveness, and (4) generate natural patterns to avoid extraction detection. Note that most of the attacks here are agnostic to the device setting. Even though some attacks were created in a cloud ML setting, we included them as they will be effective to on-device ML extraction as well. Due to the large number of papers within this category, we only describe the important ones.

Through API accesses, Tramer et al. [118] performed the first attack replicating ML model functionality to extract model parameters from popular model classes. It requires no prior knowledge of model parameters or training data. It also provides evidence that extraction attacks remain applicable even when the models only generate class labels. In a similar approach, Knockoff Nets [96] integrated reinforcement learning techniques to train a surrogate model to optimize query timing and effectiveness. Knockoff Nets aim to steal model functionality with black box internals. It does not require knowledge of model family or training data. Knockoff Nets could achieve remarkable accuracy and fidelity across a range of tasks. Simultaneously, Oh et al. [94] showed that model stealing attacks can extract different types of internal information from black-box models, such as activation types, optimization algorithms, and model hyperparameters, as have done by prior work [121]. ML-Stealer [79] presents a mere black-box access scenario. It incorporates VAE-based synthetic data generation and GAN-based replica model construction, so it does not require prior knowledge of the victim model nor the statistics of the training data. ML-Stealer has achieved a testing accuracy as high as 93.6%. To reduce the number of queries and the chance of exposure to possible defenses, Ma et al. [86] proposed a method to enhance the efficiency of model stealing using intelligent query agents. This approach employs highly accurate pre-trained models specialized for various tasks and strategically uses different categories of training data to probe the victim model. This allows intelligent query agents to achieve a near-identical accuracy to the victim model after a relatively small number of queries. Eventually, multiple intelligent agents collaborate locally to train an alternative model. In order to understand the factors that can impact the success rate of model stealing attacks, Liu et al. [81] conducted assessments of attacks under various conditions. They found that the complexity of the dataset used to train the victim model and the dataset used for the stealing attack can affect the attack's accuracy. S Pal et al. introduced an attack method [98] that employs active learning techniques to make model stealing attacks more effective. They used unannotated public data and made their attack follow a natural distribution that cannot be detected by a query distribution-based monitoring approach.

In general, model-based extraction attacks can achieve high accuracy. In generating the input queries, even though attackers may assume in-distribution datasets and out-of-distribution datasets [61], they mostly focus on image-based ML and datasets, and test on the common datasets, including CIFAR [73] and MNIST [39]. Even with different learning algorithms available, it is still hard to enable model stealing with generic types of input queries. In addition, model-based extraction attacks may fail, given the advances in different protection techniques, including adaptive misinformation, query distribution detection, and so on. Next, we describe existing works with such defenses.

3.4.2 Defender's perspective

Existing works in this category mostly focus on two directions: (1) minimizing prediction information leakage, and (2) adding cost and overhead to attacks.

In minimizing prediction information leakage, PRADA [71] serves as the first proactive defense method. PRADA's defense is based on the pattern of the attacker's query distribution, such as query proximity. It analyzes the distribution of consecutive queries and issues an alert when the query distribution is abnormal. Users can choose to return incorrect predictions, terminate queries, and so on. Since the data distribution generated in natural life is significantly different from the attacker's training set, PRADA has shown excellent identification and defense capabilities against abnormal queries, often achieving 100% accuracy. In a different approach, Prediction Poison [97] proposed a model-agnostic defense. It introduces targeted perturbations on posterior probabilities

and maximizes the angle deviation between the generated gradient signal and the original gradient signal. With a 1-2% accuracy loss, Prediction Poison reduces the accuracy of the substitute model by up to 65%, and query efficiency by 13.53%. GRAD2 [87] is also a perturbation-based defense. It uses gradient redirection to selectively alter the trajectory of model-stealing attacks. GRAD2 achieves small utility losses and low computational costs. In addition, adaptive misinformation [72] has been proposed to mislead model-stealing attackers. If a model input is outside of a pre-defined distribution, the predictions will be modified. Adaptive Misinformation maintains the high accuracy of the protected models while reducing the accuracy of the attacker's cloned model to 14.3%.

In adding cost and overhead to attacks, differential privacy [92, 99, 136] offers protection against model stealing attacks by introducing privacy-preserving noise into the training and inference processes of ML models. During ML training, controlled noise to the data and gradients makes it difficult for attackers to reconstruct a model. During ML inference, additional noise is introduced to the model's outputs, preventing attackers from learning specific internal parameters. In theory, this technique should significantly increase the complexity of stealing the model's architecture and parameters. However, empirical research conducted by Liu et al. [81] suggest that while DP and similar methods are effective at protecting the privacy of training data, their impact on model extraction attacks remains uncertain.

In general, the above defense solutions can help mitigate model-based extraction attacks. Through analyzing the distribution of input queries, these solutions can achieve a high accuracy. The challenges are (1) attackers may craft input queries that fit well into the pre-defined input categories, making adaptive misinformation and obfuscation techniques ineffective; (2) an unknown object from a pre-defined category may be incorrectly labeled as out-of-distribution category, and experience low-accuracy inference results. For example, attacks can query with unannotated public data [98] and do not demonstrate obvious patterns to bypass PRADA [71]. Therefore, it is hard for defenders to define a complete and accurate set of categories.

3.4.3 Findings

We have found that model-based extraction attacks have become more automated and with high fidelity, with the development of intelligent query agents [86]. However, the attacks may not be practical on IoT devices. Even with the optimized number of queries (e.g. 10k queries), the power consumption of devices will see an observable increase [20], and defenders may detect and terminate the attacker's querying processes.

For both attacks and defenses, the performance of the proposed solutions depends on the distribution of training data, the algorithm in generating the input, and output pairs, the number of queries, and the complexity of the victim models. It is hard for these solutions to be generalized, even with the large number of research papers available (we have only covered representative ones here). The effectiveness of the above solutions, therefore, can only apply to a small scale of models.

Since both attacks and defenses in this category are not specifically designed for IoT devices, we expect future research to identify unique challenges in IoT-specific model-based extraction security.

4 Evaluation

The goal of this section is to measure the gap between research and practice in defending model extraction attacks. Specifically, we aim to answer the following questions: (1) can research projects be reproduced in practice? (2) are proposed model extraction attacks and defenses effective in practice? and (3) what are their computation complexity and power consumption?

To obtain real-world ML models, we collected around 210K Android Application Packages (APKs) from AndroZoo [21]. The APKs were collected from popular Android stores spanning the period from 2020 to the current date. We used ModelXray [112] to filter out ML models from these APKs and obtained 16.5K models in total. Since different apps may use exactly the same models, we use hashes to identify the same models and obtain 3K **unique models** after de-duplication. The models come from different ML frameworks, and allow us to perform evaluation on existing projects described later.

4.1 Reproducibility.

We first use reproducibility as a criterion to measure the gap between research and practice in defending model extraction attacks. We will investigate (1) whether a research project is open source, (2) whether open-source projects can be reproduced with the results, and (3) what the challenges are in reproducing the projects.

Finding source code. We try three approaches: (1) go through the paper and search for the provided code link, (2) search paper title through Papers With Code [13], and (3) reach out to the author's team for code if the paper is newly published. If none of them produces a result, we consider the source code as not available.

The criteria for reproducibility. We consider a project as not reproducible if we experience system/hardware incompatibility while the system/hardware is not available anymore or has lost maintenance. In other cases, there might be erroneous code or documentation, or outdated library. We then allow two graduate students majoring in cybersecurity to test the code for a maximum of one week. If the code still fails to produce the paper-reported results, we consider the project as not reproducible.

Based on the above criteria, we statistically analyzed the existing papers, and summarized the results of open source and reproducible projects in Figure 2. As the figure shows, app-based attack and defense projects are the most reproducible ones, while device-based and communication-based projects have much lower percentages of being open source and reproducible. Projects that are not reproducible often lack core modules, have invalid links for downloading models or data, or face compatibility issues with older package versions or newer hardware requirements. The most challenging projects to reproduce are those released more than five years ago.

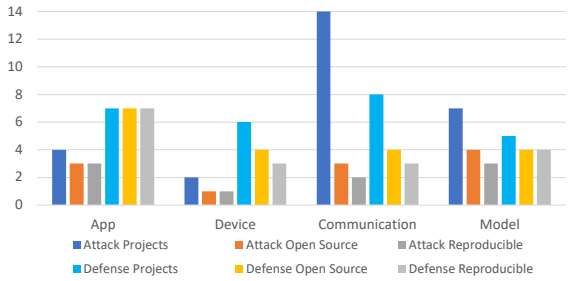

Figure 2: Open source and reproducible projects in attacks and defenses.

4.2 Effectiveness of Model Extraction Attacks.

To understand whether existing model extraction attacks are practical, for each threat model, we test and evaluate representative projects (from those reproducible) with real world applications. We consider a project representative if (i) it is popular on Github and its method is shared by most projects in the threat model, and (ii) it is the only reproducible project in the threat model. This criteria applies to defense projects as well.

4.2.1 App-based attack: ModelXray

ModelXray [105] is an app-based attack with diverse features, including package decomposing, ML model and library analysis, statistical data generation, and multi-task processing.

Results: We used our whole app collection to test ModelXray in extracting model files from app packages. We used the same criteria for plaintext model detection and obtained the results in Figure 3. Astonishingly, the attack success rate continues to rise even with heightened awareness and the availability of more defenses. Overall, the success rates is 33.83% for unique models (after we de-duplicate models with the same hashes), and 48.81% for all models, which are consistent with those reported by ModelXray (34% to 76%). It reveals that since the publication of ModelXray, the app-based attack is still a critical issue—a wide range of apps still do not protect their models at all.

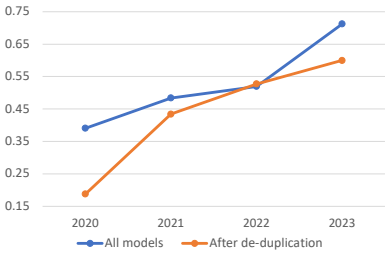

Figure 3: Success rate of ModelXray in the past four years.

Challenges: Relating this result to the app-based defenses in Table 2, we hypothesize the lack of protection to the following reasons: (1) the available protections are mostly vendor-specific [17, 106], (2) the few research solutions focus on TensorFlow [16], while our model collection shows only a small portion (e.g., 7%) of models are pure TensorFlow models. Even worse, we found that research popular frameworks (e.g., TensorFlow, PyTorch) have seen a decreased market portion in the last four years, while new ML frameworks are increasingly developed.

4.2.2 Device-based attack: ModelXtractor

ModelXtractor [105] uses Frida to extract ML models from memory, when applications are running and loading ML models. ModelXtractor can extract plaintext models even if the models have been encrypted several times before distributing to the app store.

Results: ModelXtractor is tailored to extract models in ProtoBuf (.pb, .pbtxt) and FlatBuffer (.tflite) formats. Despite the large number of models we collected, we found only 27 unique models that are encrypted and in these two formats. Since ModelXtractor requires a lot of manual effort in triggering the models, we opted to randomly select 10 models for our experiments.

Challenges: We were not able to extract useful and complete models due to the following reasons: apps cannot be instrumented, or do not trigger the ML functions; some apps require registration with phone numbers from certain countries that we could not obtain; banking apps require a local bank account to trigger ML functionalities. Additionally, ModelXtractor makes it hard to determine the start and end points of model buffers in memory. It relies on keyword searching, such as “TFL2" or “TFL3" as version numbers for TFLite models. With the fast development of new ML frameworks, ModelXtractor will see more difficulty in extracting models from memory. In fact, Deng’s work [41] with source code analysis on ML functions and program slicing can be a good complement to ModelXtractor. However, Deng’s work does not have open-source code available.

4.2.3 Comm-based attacks: DeepSniffer and DeepSteal

DeepSniffer [64] uses side-channel information, such as memory and bus monitoring, to snoop ML model structures. It functions during model executions and does not depend on specialized hardware. DeepSteal [104] leverages memory side-channel information to steal the model weight. It exploits hardware fault vulnerabilities using a rowhammer attack.

Results: DeepSniffer requires model checkpoints in *.ckpt* and *.pth.tar* formats, while DeepSteal requires PyTorch models in *.pt* format. With DeepSniffer, we successfully replicated the results reported by the paper. However, when testing with our real-world model checkpoints, DeepSniffer failed to infer layer sequences due to incompatible log files. With DeepSteal, we cannot run *.pt* models because only four types of hardware architectures (outdated) are available.

Challenges: Besides the log incompatibility issue, DeepSniffer requires retraining of predictors. The predictors may vary from one device to another and can be affected by factors such as obfuscation, trusted execution environments, model conversion, and other encryption methods. Also, the process of collecting memory access events, running the LSTM-CTC model for sequence identification, analyzing memory access patterns, and estimating layer dimensions could require non-trivial computational resources and energy demands.

4.2.4 Model-based attack: ML-Doctor

ML-Doctor [81] provides the newest and complete tool sets for model-based attacks. ML-Doctor assumes that (1) attackers have knowledge of training data and can use in-distribution data to perform attacks, and (2) target models are in black-box, but the inference function is accessible to attackers.

Results: ML-Doctor is designed to run Pytorch models in *.pt* format. We found only 8 unique models in our real-world model collection in this format. However, none of the 8 models could be loaded to ML-Doctor. We therefore repeated the same setting as reported in the paper, and achieved accuracy results ranging from 92% to 94%. However, the computational complexity of the model-stealing attack remains undetermined due to limited publicly available datasets. The study by Liu et al. (2022) [81] suggests that dataset complexity influences generalization, thereby affecting the computational cost of model-stealing techniques. Overall, the analysis indicates that the extraction attack is effective, and the results are consistent with those reported in the paper but it fails to cover broader validation across various ML models.

Challenges: ML-Doctor reveals that model-based defenses have been primarily theoretical rather than extending to real-world models. ML-Doctor may experience *false success* issues, meaning that the attack accuracy is not equal to the quality of capturing the utility or the specialized capabilities of the target models. This can happen when the target model randomly assigns labels to input data that are out of a pre-defined distribution.

While model extraction attacks offer ambiguous benefits, attackers face the growing sophistication of defensive strategies. Clues for identifying malicious queries, such as frequency, location, and data similarity, can be easily spotted and neutralized by defenders.

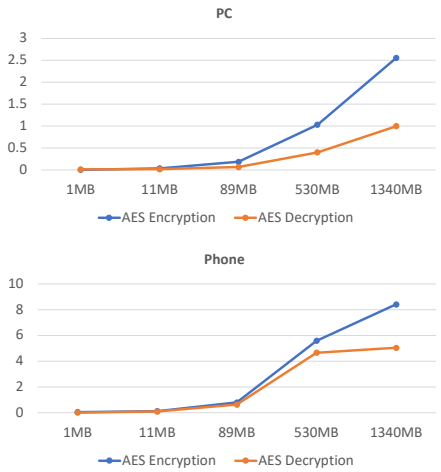

Figure 4: Execution time of encrypting and decrypting ML models with AES.

4.3 Effectiveness of Model Extraction Defenses

4.3.1 App-based defense: AES

App-based defenses, as reported in Table 2, use similar AES libraries. We select a lightweight package [16] using AES that allows us to quickly encrypt and decrypt TensorFlow model files.

Results: To assess the practicality of AES encryption, we compared performance on a desktop (Intel i5-12400, 64GB RAM) and a mobile device (Pixel 6) using various file sizes, as shown in Figure 4. We randomly selected five real-world model files in different sizes and recorded the encryption/decryption time. On Pixel 6, encrypting and decrypting a 1MB file took only 0.05 and 0.02 seconds, respectively. With the average APK size being around 2MB, encryption/decryption can be completed in approximately 0.1 seconds, demonstrating fast and practical AES implementation for on-device security.

Challenges: While encryption is effective in protecting ML models during distribution, it becomes limited when the model is loaded into memory for inference. Attackers who can access the system's memory may extract the model in decrypted format. To address this challenge, various new encryption techniques are continually emerging. The new methods aim to offer an additional layer of protection, even when the model is loaded into memory for operations [65].

4.3.2 Device and comm-based defense: ShadowNet

ShadowNet [111] is selected due to its support for Trusted Execution Environment (TEE) to protect model privacy and uses hardware acceleration to speed up the performance of on-device ML. ShadowNet secures non-linear layers in TEE and runs linear layers in GPU. It applies a linear transformation to the weight data sent between TEE and GPU. Other works use similar techniques but do not protect model weights [117], or lack the support for GPU acceleration.

Results: ShadowNet supports TFLite and Darknet frameworks to run in TEE-emulated mode. It also offers a model-transforming pipeline that transforms models into ShadowNet-enabled models. We were able to reproduce the results of ShadowNet using the same models as used in the paper, including MobileNet, AlexNet and MiniVGG. The amount of manual effort in setting up ShadowNet is huge, including (1) model preparation with Keras API, (2) choosing an obfuscation scheme on a model, (3) splitting model based on a set of rules, (4) generating model weights for the TEE part, and (5) writing the client application and trusted application for the model with ShadowNet libraries. We were not able to test our collected real-world models. According to the documentation, both CNN and RNN models in TFLite models from Android can be seamlessly integrated.

Challenges: The amount of effort in using ShadowNet is more than expected. The steps in transforming models to ShadowNet-enabled models prevent ShadowNet's wide adoption. In addition, ShadowNet's TEE-enabled mode is only supported on TFLite, while models from other ML frameworks will need to be transformed or even compressed, which will reduce the original models' accuracy. ShadowNet's prototype was tested on the Hikey board 960, while the other types of hardware may not be compatible.

4.3.3 Model-based defenses: Prediction-Poison and Adaptive Misinformation

Prediction-Poison(PP) [97] is selected due to its good performance on a variety of models and its effectiveness in poisoning the training objectives of the attacker by perturbing the predictions. Adaptive Misinformation (AM) [72] deliberately introduces inaccuracies for queries identified as Out-Of-Distribution (OOD) while ensuring that In-Distribution (ID) queries receive correct predictions. Because a significant portion of the adversary's queries fall into the OOD category, the mislabeled data leads the adversary to train low-quality student models with poor accuracy. Other defenses [71] will incur a slight delay of attack by increasing the number of queries and achieving low accuracy in protection.

Results: Both defenses aim to reduce the accuracy of the adversary's student model with a minimum impact on the defender's model accuracy. We used the Knockoff Nets attack strategy to evaluate the [96] Prediction-Poison (PP) [97] and Adaptive Misinformation (AM) [72]. Both defenses are limited to Pytorch models stored in .pth/.pt format. We examined the results on a surrogate dataset for undefended victim models provided by the author like MNIST, CIFAR10, ResNet-18, LeNET, and CUB200. Our investigation revealed that PP has overall lower clone accuracy whereas AM has lower computational overhead compared to PP. Additionally, both AM and PP have minimal defender accuracy loss (< 1% - 0.5%). However, we encountered errors while loading the other extracted models stored in .pt/.pth formats. The code will ask for an additional log file that was not available.

Challenges: Compared to previous works, PP and AM are effective techniques. But with an increase in the query budget and optimizing parameters, they may not scale well on resource-intensive devices or with large complex target models. The introduced computational overhead and the slowing down of model inference tasks could make them less practical in wide adoption. Additionally, an adaptive attacker may bypass the detection of misinformation.

In summary, the transition from open-source projects to practice requires a comprehensive examination of various factors. We have examined the effectiveness of attacks and defenses from several representative projects. Later work in evaluating the runtime performance, the required resources, and the deployment effort should be warranted.

4.4 Other Metrics

4.4.1 Computation complexity

Computation complexity is an important metric in determining attack feasibility in victim devices and defense performance. We analyze the computation complexity of the aforementioned projects which require access to victim devices.

DeepSniffer's complexity can be represented as $O(k + f(n) + b*n)$, where k is the number of kernel classes, $f(n)$ is the complexity function of the sequence model, and $(b*n)$ represents the search algorithm with the width (b) and the length of the sequence (n). DeepSteal reduces the search space of model weights W with a rowhammer attack and has an overall time complexity of $O(RowHammerAttacks)$ + $O(W + T*B)$, where (W) is the leaked weight bits, and $(T*B)$ is the number of training iterations and batches. ML-Doctor's complexity equals training a student model, which is $O(m*d*e)$, where m is the number of queries, d is the combined size of the network (total number of weights and biases), and e is the number of epochs.

The complexity of AES depends on key size, block size, number of rounds, and model size. The former three factors are usually fixed numbers (e.g., 128, 256), so the complexity becomes $O(m)$ where m is the model size. ShadowNet's complexity can be represented as $O(TEE + r*l)$, which depends on the TEE implementation, the number of linear layers l in target ML models, and a constant number r as the trans-

formation ratio. AM and PP train a process to minimize the worst-case loss over perturbations and the complexity equals $O(g*h)$, where $O(g)$ is the inner optimization in finding the worst-case perturbation, and $O(h)$ is the outer optimization in updating the model parameters.

4.4.2 Power consumption

Energy efficiency has been overlooked in existing model extraction attacks and defenses, although these efforts have considered efficient algorithms or code optimizations to accelerate execution or reduce computation overhead. We measure the power consumption of different ML model extraction attacks and defenses, using Intel PCM. Some methods are ignored because they do not need access to victim devices, such as ModelXray, or require a specific hardware architecture (e.g., DeepSteal) or device (e.g., ShadowNet) that are unavailable. Table 3 shows the power consumption in Joules before and after attacks or defenses. Specifically, AES consumes significantly less power compared with others. AM is more energy efficient compared to PP. The average consumption is around 30 Joules.

Table 3: Power consumption of different projects.

Project	Model	Before (J)	After (J)
DeepSniffer	ResNet-18	0.45	29.98
ML-Doctor	a simple CNN	0.70	33.81
AES	ResNet-18	0.41	3.28
PP	LeNet	0.42	33.47
AM	LeNet	0.77	29.24

5 Future Research Directions

Multi-user sharing on-device ML. In this paper, we focus on on-device ML models that are used by a single user. However, there are other cases where an IoT application with on-device ML is used by multiple users. For example, a smart home with different IoT devices can be shared by home residents [135]. Another example is a healthcare service using an IoT device that can be shared among multiple patients or staff in the hospital [76]. Sharing on-device ML may raise new security challenges which may lead to privacy violations for the app users if an attacker is one of the users who share the model. In this case, the attacker may try to attack the model to collect sensitive information about the model users (patients, for example). New defense techniques are needed if on-device ML is used in a multi-user environment to maintain the security and privacy of the users.

On-device ML in Federated Learning. In some on-device ML applications, the future updates for on-device ML models depend on locally trained models by different users to improve the accuracy of the model results. To make updating the ML model possible, federated learning (FL) is usually used to enable model training and inference to occur directly on the device [125]. FL regularly collects general information about all locally trained models to be used in the future update from the ML-provider server. Model extraction from the on-device ML apps by an untrusted user is still possible in a large number of IoT apps. More work is needed with the help of techniques like Multi-party Computation (MPC) to prevent such attacks on the on-device ML apps. We suggested using MPC since MPC plays an important role in privacy preserving when multiple-party involved in the system.

On-device models with early exit. Early exit policies and the architecture in ML allow large deep neural network (DNN) models to run on resource-constrained devices. This improves ML inference performance and data transmission efficiency. Existing works have partitioned ML models to introduce early exits across multiple systems [25, 52]. This includes transformer-based neural networks [24, 126, 139]. However, early exit models suffer from model extraction attacks as well. For example, model-based attacks can query the early-exit model with different types of input data and use the input, output pairs to infer the exit places and possible parameters. Existing efforts have been proposed to fingerprint early-exit models via inference time and have shown uniqueness and robustness on different model architectures [43]. In addition, because early exit policies are data-dependent, data patterns can be used to reveal model architecture or even model parameters. To defend against such attacks, we suggest future research to explore shuffling, randomization, and perturbation-based techniques [82], for example, exit point randomization, inference result perturbation, and inference with noises. We expect possible performance loss, including decreased inference speed and accuracy. We suggest following existing studies [43] to balance between model privacy and accuracy.

Cryptography-based defenses. Cryptography-based defenses provide a strong mathematical back-end to secure ML models from various attacks. For example, Homomorphic encryption (HE) [19, 30, 102] has been explored to ensure the privacy of both ML models and their input data. Even though the state-of-the-art HE and MPC algorithms are still too slow to use in practice, existing efforts have been explored to reduce the performance overhead. These include Partially Homomorphic Encryption (PHE) schemes for the reduced computation complexity, and optimized libraries to help mitigate the performance overhead. For example, Microsoft's SEAL (Simple Encrypted Arithmetic Library) [36] and IBM's HElib [53] provide optimized implementations for various applications. We suggest future research to explore more efficient Cryptography-based solutions to protect ML models from stealing, including AES, HE, and MPC.

ML models using on-device GPUs. GPUs are increasingly adopted by devices to accelerate ML tasks. With the advances in LLM, we expect to see more adoption of on-device GPUs in the future. However, modern GPUs lack memory protection support at the equivalent level of CPUs and can suffer

from a range of security attacks [38], including model data reconstruction from GPU dumps. We suggest future research to investigate securing on-device accelerators from model extraction attacks. For example, in general-purpose computers, GPUShield [75] provides a hardware-software cooperative region-based bounds-checking mechanism. It improves GPU memory safety by assigning a random unique ID to each buffer and storing individual bounds in the global memory. For embedded devices, D-Box [89] provides a compartmentalization solution to different tasks sharing access to DMA, allowing developers to create security policies flexibly. We believe both solutions would be interesting in isolating different ML tasks sharing the same on-device GPUs.

6 Conclusion

In this paper, we have provided a systematic review of knowledge concerning on-device ML model extraction attacks and defenses. To facilitate clear understanding, we have defined four types of threat models for both attacks and defenses. Our findings reveal that while certain attacks may not be practical or scalable, the corresponding defense solutions are often constrained in their deployment as well. Real-world ML implementation can be complex even in an on-device setting. Therefore, we have identified future research directions aimed at preventing complex model extraction attacks.

Acknowledgement

We thank the anonymous reviewers for their helpful feedback and time. This work was partially supported by the US National Science Foundation (Awards: 2039606, 2219920, 2327427) and Microsoft. We thank AndroZoo for providing the Android applications. The views expressed are those of the authors only, not of the funding agencies.

References

[1] A brief guide to mobile AI chips. https://www.theverge.com/2017/10/19/16502538/mobile-ai-chips-apple-google-huawei-qualcomm.

[2] APKTool - A tool for reverse engineering Android APK files. https://github.com/iBotPeaches/Apktool.

[3] CacheAudit - A Tool for the Static Analysis of Cache Side Channels. https://github.com/IAIK/CacheAudit.

[4] ChipWhisperer - An Open-Source Toolchain for Hardware Security Research. https://chipwhisperer.readthedocs.io/en/latest/.

[5] Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers. https://frida.re/.

[6] GNU Debugger (GDB). https://www.gnu.org/software/gdb/.

[7] IDA Pro - The IDA Disassembler and Debugger. https://www.hex-rays.com/products/ida/.

[8] IoT Development. Top 15 Internet of Things Tools and Platforms in 2023. https://www.sam-solutions.com/blog/iot-development/.

[9] jadx - DEX to Java decompiler. https://github.com/skylot/jadx.

[10] Mace: Mobile ai compute engine. mace/tools/python/encrypt.pyatmasterÂůXiaoMi/maceÂůGitHub.

[11] MemFetch - Memory Acquisition and Analysis Tool. https://github.com/504ensicsLabs/Memfetch.

[12] MLC LLM. https://mlc.ai/mlc-llm/#android.

[13] Paper With Code. https://paperswithcode.com/.

[14] SCA-Lab - Side-Channel Analysis Laboratory. https://github.com/Riscure/SCA-Lab.

[15] Strip visible string in ncnn. https://github.com/Tencent/ncnn/blob/be054aacb0cfac3725507d220c417e138803ebab/docs/how-to-use-and-FAQ/use-ncnn-with-alexnet.md.

[16] TFSecured. https://github.com/dneprDroid/tfsecured. seems for ios.

[17] Mindspore. https://github.com/mindspore-ai/mindspore/tree/08085e7d6a63fa08c548b5006a3297c08ff88def/mindspore/lite/tools, 2020.

[18] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

[19] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (Csur)*, 51(4):1–35, 2018.

[20] Anshul Agarwal, Vitobha Munigala, and Krithi Ramamritham. Observability: A principled approach to provisioning sensors in buildings. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*, pages 197–206, 2016.

[21] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 468–471, New York, NY, USA, 2016. ACM.

[22] Leonardo Babun, Z Berkay Celik, Patrick McDaniel, and A Selcuk Uluagac. Real-time analysis of privacy-(un) aware iot applications. In *Privacy Enhancing Technologies Symposium (PETS) 2021*, number 1, 2021.

[23] Tao Bai, Jinqi Luo, Jun Zhao, Bihan Wen, and Qian Wang. Recent advances in adversarial training for adversarial robustness. *arXiv preprint arXiv:2102.01356*, 2021.

[24] Arian Bakhtiarnia, Qi Zhang, and Alexandros Iosifidis. Multi-exit vision transformer for dynamic inference. *arXiv preprint arXiv:2106.15183*, 2021.

[25] Amin Banitalebi-Dehkordi, Naveen Vedula, Jian Pei, Fei Xia, Lanjun Wang, and Yong Zhang. Auto-split: A General Framework of Collaborative Edge-Cloud AI. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2543–2553, 2021.

[26] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. {CSI}{NN}: Reverse engineering of neural network architectures through electromagnetic side channel. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 515–532, 2019.

[27] Sebastian P Bayerl, Tommaso Frassetto, Patrick Jauernig, Korbinian Riedhammer, Ahmad-Reza Sadeghi, Thomas Schneider, Emmanuel Stapf, and Christian Weinert. Offline model guard: Secure and private ml on mobile devices. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 460–465. IEEE, 2020.

[28] BayesWitnesses. m2cgen: A lightweight library for transpiling trained statistical models into native code. https://github.com/BayesWitnesses/m2cgen#supported-models, 2022.

[29] Alessandro Bellini, Emanuele Bellini, Massimo Bertini, Doaa Almhaithawi, and Stefano Cuomo. Multi-party computation for privacy and security in machine learning: a practical review. In *2023 IEEE International Conference on Cyber Security and Resilience (CSR)*, pages 174–179. IEEE, 2023.

[30] Fabian Boemer, Anamaria Costache, Rosario Cammarota, and Casimir Wierzynski. nGraph-HE2: A high-throughput framework for neural network inference on encrypted data. In *Proceedings of the 7th*

ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography, pages 45–56, 2019.

[31] Ferdinand Brasser, David Gens, Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. Sanctuary: Arming trustzone with user-space enclaves. *Proceedings 2019 Network and Distributed System Security Symposium*, 2019.

[32] Cesar Federico Caiafa, Jordi Solé-Casals, Pere Marti-Puig, Sun Zhe, and Toshihisa Tanaka. Decomposition methods for machine learning with small, incomplete or noisy datasets. *Applied Sciences*, 10(23):8481, 2020.

[33] Yinzhi Cao, Alexander Fangxiao Yu, Andrew Aday, Eric Stahl, Jon Merwine, and Junfeng Yang. Efficient repair of polluted machine learning systems via causal unlearning. In *Proceedings of the 2018 on Asia conference on computer and communications security*, pages 735–747, 2018.

[34] Z Berkay Celik, Leonardo Babun, Amit Kumar Sikder, Hidayet Aksu, Gang Tan, Patrick McDaniel, and A Selcuk Uluagac. Sensitive information tracking in commodity {IoT}. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1687–1704, 2018.

[35] Deyan Chen and Hong Zhao. Data security and privacy protection issues in cloud computing. In *2012 international conference on computer science and electronics engineering*, volume 1, pages 647–651. IEEE, 2012.

[36] Hao Chen, Kim Laine, and Rachel Player. Simple encrypted arithmetic library-SEAL v2. 1. In *Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21*, pages 3–18. Springer, 2017.

[37] Yoon-Ho Choi, Peng Liu, Zitong Shang, Haizhou Wang, Zhilong Wang, Lan Zhang, Junwei Zhou, and Qingtian Zou. Using deep learning to solve computer security challenges: a survey. *Cybersecurity*, 3(1):1–32, 2020.

[38] Cristina Cismaru, Ruxandra Chiroiu Trandafir, and Emil-Ioan Slusanschi. A study of gpu memory vulnerabilities. In *2023 22nd RoEduNet Conference: Networking in Education and Research (RoEduNet)*, pages 1–9. IEEE, 2023.

[39] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pages 2921–2926. IEEE, 2017.

[40] Xiangfeng Dai, Irena Spasić, Bradley Meyer, Samuel Chapman, and Frederic Andres. Machine learning on mobile: An on-device inference app for skin cancer detection. In *2019 fourth international conference on fog and mobile edge computing (FMEC)*, pages 301–305. IEEE, 2019.

[41] Zizhuang Deng, Kai Chen, Guozhu Meng, Xiaodong Zhang, Ke Xu, and Yao Cheng. Understanding real-world threats to deep learning models in android apps. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 785–799, 2022.

[42] Marjolein Dijkstra and Erik Luijten. From predictive modelling to machine learning and reverse engineering of colloidal self-assembly. *Nature materials*, 20(6):762–773, 2021.

[43] Tian Dong, Han Qiu, Tianwei Zhang, Jiwei Li, Hewu Li, and Jialiang Lu. Fingerprinting multi-exit deep neural network models via inference time. *arXiv preprint arXiv:2110.03175*, 2021.

[44] Vasisht Duddu, Debasis Samanta, D Vijay Rao, and Valentina E Balas. Stealing neural networks via timing side channels. *arXiv preprint arXiv:1812.11720*, 2018.

[45] Jan-Erik Ekberg, Kari Kostiainen, and Nadarajah Asokan. Trusted execution environments on mobile devices. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1497–1498, 2013.

[46] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to {Byzantine-Robust} federated learning. In *29th USENIX security symposium (USENIX Security 20)*, pages 1605–1622, 2020.

[47] Mudasir A Ganaie, Minghui Hu, AK Malik, M Tanveer, and PN Suganthan. Ensemble deep learning: A review. *Engineering Applications of Artificial Intelligence*, 115:105151, 2022.

[48] M. Garcia-Bosque, G. Díez-Señorans, C. Sánchez-Azqueta, and S. Celma. Introduction to physically unclonable fuctions: Properties and applications. In *2020 European Conference on Circuit Theory and Design (ECCTD)*, pages 1–4, 2020.

[49] Ian Goodfellow, Nicolas Papernot, Sandy Huang, Yan Duan, Pieter Abbeel, and Jack Clark. Attacking machine learning with adversarial examples. *OpenAI Blog*, 24, 2017.

[50] Jiyang Guan, Jian Liang, and Ran He. Are you stealing my model? sample correlation for fingerprinting deep neural networks. *Advances in Neural Information Processing Systems*, 35:36571–36584, 2022.

[51] Trung Ha, Tran Khanh Dang, Hieu Le, and Tuan Anh Truong. Security and privacy issues in deep learning: a brief review. *SN Computer Science*, 1(5):253, 2020.

[52] Amirhossein Haddadpour, Keivan Navaie, Halim Yanikomeroglu, and H. Vincent Poor. Adaptive Federated Learning in Resource Constrained Edge Computing Systems. *IEEE Transactions on Communications*, 68(11):6875–6889, 2020.

[53] Shai Halevi and Victor Shoup. Design and implementation of a homomorphic-encryption library. *IBM Research (Manuscript)*, 6(12-15):8–36, 2013.

[54] Shigeyuki Hamori, Minami Kawai, Takahiro Kume, Yuji Murakami, and Chikara Watanabe. Ensemble learning or deep learning? application to default risk analysis. *Journal of Risk and Financial Management*, 11(1):12, 2018.

[55] Seung-Kyun Han and Jinsoo Jang. Mytee: Own the trusted execution environment on embedded devices. In *NDSS*, 2023.

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

[57] Charles Herder, Meng-Day Yu, Farinaz Koushanfar, and Srinivas Devadas. Physical unclonable functions and applications: A tutorial. *Proceedings of the IEEE*, 102(8):1126–1141, 2014.

[58] Simon Heron. Advanced encryption standard (aes). *Network Security*, 2009(12):8–12, 2009.

[59] Sanghyun Hong, Michael Davinroy, Yiğitcan Kaya, Dana Dachman-Soled, and Tudor Dumitraş. How to 0wn nas in your spare time. *arXiv preprint arXiv:2002.06776*, 2020.

[60] 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. *arXiv preprint arXiv:1810.03487*, 2018.

[61] Yen-Chang Hsu, Yilin Shen, Hongxia Jin, and Zsolt Kira. Generalized odin: Detecting out-of-distribution image without learning from out-of-distribution data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10951–10960, 2020.

[62] Hailong Hu and Jun Pang. Model extraction and defenses on generative adversarial networks. *arXiv preprint arXiv:2101.02069*, 2021.

[63] Han Hu, Yujin Huang, Qiuyuan Chen, Terry Yue zhuo, and Chunyang Chen. A first look at on-device models in ios apps. *ACM Transactions on Software Engineering and Methodology*, 2023.

[64] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, et al. Deepsniffer: A dnn model extraction framework based on learning architectural hints. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 385–399, 2020.

[65] Jiayi Hua, Yuanchun Li, and Haoyu Wang. Mmguard: Automatically protecting on-device deep learning models in android apps. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 71–77. IEEE, 2021.

[66] Weizhe Hua, Muhammad Umar, Zhiru Zhang, and G Edward Suh. Mgx: Near-zero overhead memory protection with an application to secure dnn acceleration. *CoRR*, pages 1–14, 2020.

[67] Weizhe Hua, Zhiru Zhang, and G Edward Suh. Reverse engineering convolutional neural networks through side-channel information leaks. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[68] Yujin Huang and Chunyang Chen. Smart app attack: hacking deep learning models in android apps. *IEEE Transactions on Information Forensics and Security*, 17:1827–1840, 2022.

[69] Apple Inc. Core ML. https://developer.apple.com/documentation/coreml.

[70] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pages 1209–1222, 2018.

[71] 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.

[72] Sanjay Kariyappa and Moinuddin K Qureshi. Defending against model stealing attacks with adaptive misinformation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2020.

[73] Alex Krizhevsky and Geoff Hinton. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 40(7):1–9, 2010.

[74] Keita Kurita, Paul Michel, and Graham Neubig. Weight poisoning attacks on pre-trained models. *arXiv preprint arXiv:2004.06660*, 2020.

[75] Jaewon Lee, Yonghae Kim, Jiashen Cao, Euna Kim, Jaekyu Lee, and Hyesoon Kim. Securing gpu via region-based bounds checking. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 27–41, 2022.

[76] Yong Lee and Goo Yeon Lee. Security management suitable for lifecycle of personal information in multi-user iot environment. *Sensors*, 21(22):7592, 2021.

[77] Jingtao Li, Zhezhi He, Adnan Siraj Rakin, Deliang Fan, and Chaitali Chakrabarti. Neurobfuscator: A full-stack obfuscation tool to mitigate neural architecture stealing. In *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 248–258. IEEE, 2021.

[78] Lin Li and Michael Spratling. Data augmentation alone can improve adversarial training. *arXiv preprint arXiv:2301.09879*, 2023.

[79] Gaoyang Liu, Shijie Wang, Borui Wan, Zekun Wang, and Chen Wang. Ml-stealer: Stealing prediction functionality of machine learning models with mere black-box access. In *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 532–539. IEEE, 2021.

[80] Yugeng Liu, Rui Wen, Xinlei He, Ahmed Salem, Zhikun Zhang, Michael Backes, Emiliano De Cristofaro, Mario Fritz, and Yang Zhang. ML-Doctor: Holistic risk assessment of inference attacks against machine learning models. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 4525–4542, Boston, MA, August 2022. USENIX Association.

[81] Yugeng Liu, Rui Wen, Xinlei He, Ahmed Salem, Zhikun Zhang, Michael Backes, Emiliano De Cristofaro, Mario Fritz, and Yang Zhang. {ML-Doctor}: Holistic risk assessment of inference attacks against machine learning models. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 4525–4542, 2022.

[82] Yuntao Liu, Dana Dachman-Soled, and Ankur Srivastava. Mitigating reverse engineering attacks on deep neural networks. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 657–662. IEEE, 2019.

[83] Yuntao Liu and Ankur Srivastava. Ganred: Gan-based reverse engineering of dnns via cache side-channel. In *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, pages 41–52, 2020.

[84] Rundong Luo, Yifei Wang, and Yisen Wang. Rethinking the effect of data augmentation in adversarial contrastive learning. *arXiv preprint arXiv:2303.01289*, 2023.

[85] Yukui Luo, Shijin Duan, Cheng Gongye, Yunsi Fei, and Xiaolin Xu. NnreArch: A tensor program scheduling framework against neural network architecture reverse engineering. In *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 1–9. IEEE, 2022.

[86] Chen Ma, Li Chen, and Jun-Hai Yong. Simulating unknown target models for query-efficient black-box attacks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11835–11844, 2021.

[87] Mantas Mazeika, Bo Li, and David Forsyth. How to steer your adversary: Targeted and efficient model stealing defenses with gradient redirection. In *International Conference on Machine Learning*, pages 15241–15254. PMLR, 2022.

[88] K Meenakshi and G Maragatham. A review on security attacks and protective strategies of machine learning. *Emerging Trends in Computing and Expert Technology*, pages 1076–1087, 2020.

[89] Alejandro Mera, Yi Hui Chen, Ruimin Sun, Engin Kirda, and Long Lu. D-Box: DMA-enabled Compartmentalization for Embedded Applications. In *The Network and Distributed System Security (NDSS) 2022*, 2022.

[90] MindsDB. MindsDB: Abstracting Machine Learning Models as Virtual Tables (AI-Tables) for SQL Interactions. https://mindsdb.com/?utm_medium=referral&utm_source=medium&utm_campaign=mlflow+article+2022-03, Year the resource was accessed. 2022.

[91] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. Darknetz: towards model privacy at the edge using trusted execution environments. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 161–174, 2020.

[92] Milad Nasr, Shuang Songi, Abhradeep Thakurta, Nicolas Papernot, and Nicholas Carlin. Adversary instantiation: Lower bounds for differentially private machine learning. In *2021 IEEE Symposium on security and privacy (SP)*, pages 866–882. IEEE, 2021.

[93] Mario Noseda, Lea Zimmerli, Tobias Schläpfer, and Andreas Rüst. Performance analysis of secure elements for iot. *IoT*, 3(1):1–28, 2021.

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

[95] Daryna Oliynyk, Rudolf Mayer, and Andreas Rauber. I know what you trained last summer: A survey on stealing machine learning models and defences. *ACM Computing Surveys*, 55, 04 2023.

[96] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Knockoff nets: Stealing functionality of black-box models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4954–4963, 2019.

[97] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Prediction poisoning: Towards defenses against dnn model stealing attacks. *arXiv preprint arXiv:1906.10908*, 2019.

[98] Soham Pal, Yash Gupta, Aditya Shukla, Aditya Kanade, Shirish Shevade, and Vinod Ganapathy. Activethief: Model extraction using active learning and unannotated public data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 865–872, 2020.

[99] Nicolas Papernot, Martín Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint arXiv:1610.05755*, 2016.

[100] Seetal Potluri and Aydin Aysu. Stealing neural network models through the scan chain: A new threat for ml hardware. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–8. IEEE, 2021.

[101] Arun K Pujari. *Data mining techniques*. Universities press, 2001.

[102] Bernardo Pulido-Gaytan, Andrei Tchernykh, Jorge M Cortés-Mendoza, Mikhail Babenko, Gleb Radchenko, Arutyun Avetisyan, and Alexander Yu Drozdov. Privacy-preserving neural networks with homomorphic encryption: C hallenges and opportunities. *Peer-to-Peer Networking and Applications*, 14(3):1666–1691, 2021.

[103] Pytorch. Pytorch Mobile. https://pytorch.org/mobile/home/.

[104] Adnan Siraj Rakin, Md Hafizul Islam Chowdhuryy, Fan Yao, and Deliang Fan. Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1157–1174, 2022.

[105] RiS3-Lab. ModelXRay: On-device Machine Learning Model Analyzer and Extractor for Android Apps. https://github.com/RiS3-Lab/ModelXRay/blob/main/modelxray.py.

[106] Samsung. Knox ML Encryption Tool. https://docs.samsungknox.com/dev/knox-sdk/ml-protection/model-protection-tool.htm.

[107] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

[108] Manli Shu, Yu Shen, Ming C Lin, and Tom Goldstein. Adversarial differentiable data augmentation for autonomous systems. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14069–14075. IEEE, 2021.

[109] François-Xavier Standaert. Introduction to side-channel attacks. *Secure integrated circuits and systems*, pages 27–42, 2010.

[110] Lizhi Sun, Shuocheng Wang, Hao Wu, Yuhang Gong, Fengyuan Xu, Yunxin Liu, Hao Han, and Sheng Zhong. Leap: Trustzone based developer-friendly tee for intelligent mobile apps. *IEEE Transactions on Mobile Computing*, 2022.

[111] Zhichuang Sun, Ruimin Sun, Changming Liu, Amrita Roy Chowdhury, Long Lu, and Somesh Jha. Shadownet: A secure and efficient on-device model inference system for convolutional neural networks. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1489–1505. IEEE Computer Society, 2022.

[112] Zhichuang Sun, Ruimin Sun, Long Lu, and Alan Mislove. Mind your weight (s): A large-scale study on insufficient machine learning model protection in mobile apps. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1955–1972, 2021.

[113] Fnu Suya, Saeed Mahloujifar, Anshuman Suri, David Evans, and Yuan Tian. Model-targeted poisoning attacks with provable convergence. In *International Conference on Machine Learning*, pages 10000–10010. PMLR, 2021.

[114] Nazli Tekin, Ahmet Aris, Abbas Acar, Selcuk Uluagac, and Vehbi Cagri Gungor. A review of on-device machine learning for iot: An energy perspective. *Ad Hoc Networks*, 153:103348, 2024.

[115] Nazlı Tekin, Abbas Acar, Ahmet Arış, Selcuk Uluagac, and Vehbi Gungor. Energy consumption of on-device machine learning models for iot intrusion detection. *Internet of Things*, 21:100670, 12 2022.

[116] TensorFlow. TensorFlow Lite. https://www.tensorflow.org/lite.

[117] Florian Tramer and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287*, 2018.

[118] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 601–618, 2016.

[119] Jean-Baptiste Truong, Pratyush Maini, Robert J Walls, and Nicolas Papernot. Data-free model extraction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4771–4780, 2021.

[120] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. Graviton: Trusted execution environments on gpus. In *OSDI*, pages 681–696, 2018.

[121] 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.

[122] Xingbin Wang, Rui Hou, Yifan Zhu, Jun Zhang, and Dan Meng. Npufort: A secure architecture of dnn accelerator against model inversion attack. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, pages 190–196, 2019.

[123] Zhendong Wang, Xiaoming Zeng, Xulong Tang, Danfeng Zhang, Xing Hu, and Yang Hu. Demystifying arch-hints for model extraction: An attack in unified memory system. *arXiv preprint arXiv:2208.13720*, 2022.

[124] 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. IEEE, 2020.

[125] Jie Wen, Zhixia Zhang, Yang Lan, Zhihua Cui, Jianghui Cai, and Wensheng Zhang. A survey on federated learning: challenges and applications. *International Journal of Machine Learning and Cybernetics*, 14(2):513–535, 2023.

[126] Wang Wenjian, Xiao Qian, Xue Jun, and Hu Zhikun. DynamicSleepNet: a multi-exit neural network with adaptive inference time for sleep stage classification. *Frontiers in Physiology*, 14:1171467, 2023.

[127] Wikipedia contributors. Amazon alexa — Wikipedia, the free encyclopedia, 2024. [Online; accessed 10-February-2024].

[128] Yun Xiang, Zhuangzhi Chen, Zuohui Chen, Zebin Fang, Haiyang Hao, Jinyin Chen, Yi Liu, Zhefu Wu, Qi Xuan, and Xiaoniu Yang. Open dnn box by power side-channel attack. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(11):2717–2721, 2020.

[129] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. A first look at deep learning apps on smartphones. In *The World Wide Web Conference*, pages 2125–2136, 2019.

[130] Mengjia Yan, Christopher W Fletcher, and Josep Torrellas. Cache telepathy: Leveraging shared resource attacks to learn {DNN} architectures. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2003–2020, 2020.

[131] Dingqing Yang, Prashant J Nair, and Mieszko Lis. Huffduff: Stealing pruned dnns from sparse accelerators. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 385–399, 2023.

[132] Xun Yi, Russell Paulet, Elisa Bertino, Xun Yi, Russell Paulet, and Elisa Bertino. *Homomorphic encryption*. Springer, 2014.

[133] Shunsuke Yoshimura, Kazuaki Nakamura, Naoko Nitta, and Noboru Babaguchi. Model inversion attack against a face recognition system in a black-box setting. In *2021 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1800–1807, 2021.

[134] Honggang Yu, Haocheng Ma, Kaichen Yang, Yiqiang Zhao, and Yier Jin. Deepem: Deep neural networks model recovery through em side-channel information leakage. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 209–218. IEEE, 2020.

[135] Eric Zeng and Franziska Roesner. Understanding and improving security and privacy in {multi-user} smart homes: A design exploration and {in-home} user study. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 159–176, 2019.

[136] Fangyuan Zhao, Xuebin Ren, Shusen Yang, Qing Han, Peng Zhao, and Xinyu Yang. Latent dirichlet allocation model training with differential privacy. *IEEE Transactions on Information Forensics and Security*, 16:1290–1305, 2020.

[137] Weimin Zhao, Sanaa Alwidian, and Qusay H Mahmoud. Adversarial training methods for deep learning: A systematic review. *Algorithms*, 15(8):283, 2022.

[138] Tong Zhou, Shaolei Ren, and Xiaolin Xu. Obfunas: A neural architecture search-based dnn obfuscation approach. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2022.

[139] Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. Bert loses patience: Fast and robust inference with early exit. *Advances in Neural Information Processing Systems*, 33:18330–18341, 2020.

[140] Yuankun Zhu, Yueqiang Cheng, Husheng Zhou, and Yantao Lu. Hermes attack: Steal dnn models with lossless inference accuracy. In *USENIX Security Symposium*, pages 1973–1988, 2021.