Too Noisy To Extract : Pitfalls of Model Extraction Attacks

Mujahid Al Rafi, Yuan Feng, Hyeran Jeon
University of California Merced
Email: {mrafi, yfeng44, hjeon7}@ucmerced.edu

***Abstract*—Deep learning solutions become one of the most important intellectual properties of the solution providers. A few studies raised the alarm about the leakage of important model parameters through model extraction attacks. This paper discusses the practicality of such attacks by exploring the baseline assumptions of the existing attack scenarios.**

1. Introduction

Recently, concerns are increasing about the hardware attacks that exploit the side-channels formed by the unique structure of micro-architectures. Various attack models have been demonstrated that make the designers and vendors revise their solutions to protect against any private and secure information leakage. These side-channel attacks show so-called the *best case* leakages. This means that valid information can be leaked and recovered only when the presumed conditions are all satisfied. Therefore, it is important to understand the baseline assumptions of each attack model. For example, cache side-channel attacks such as PRIME+PROBE assume that the attacker is aware of the victim code structure as well as the cache configuration. Without such information, the attack cannot be implemented. In this paper, we would like to discuss the validness of assumptions of *deep learning model extraction attacks* and raise an open question if such attacks are practical or just illusions.

2. Methodologies

We used DeepSniffer [1] framework released in a public repository [2] to test model extractions. We evaluated ResNet-50 and ResNet-101 models that are downloaded from different sources: DeepSniffer [2], Google TensorFlow Hub [3], [4], MXNet Repository [5], and NVIDIA NGC Catalog [6]. We intentionally collected models designed for different deep learning frameworks to understand the impact of frameworks, such as Pytorch, TensorFlow, and MXNet. We ran the experiments on a labtop that has an NVIDIA GeForce GTX 1660Ti (Turing) GPU with CUDA v11.5. While running the models, we collected three key architecture statistics, kernel execution time, kernel read volume, and write volume via Nsight profiler. We also collected the names of kernels to find the sources of statistical differences among the models. We used a cat image as an inference input for all models.

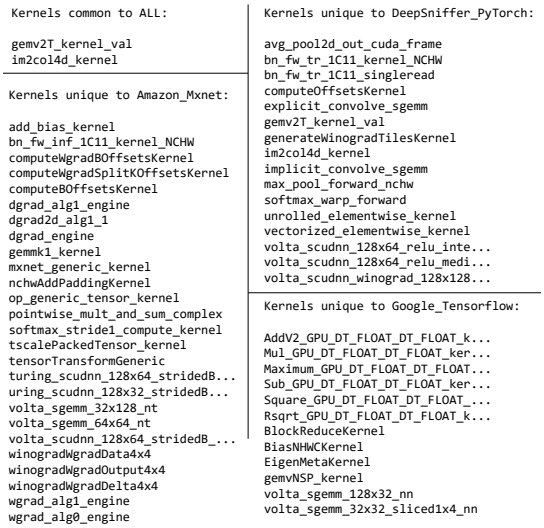

Figure 1: Common and Unique Kernels used by ResNet-101 models Designed for Different Frameworks

3. Impact of Frameworks and Vendor-Specific Optimization

With the rising impact of deep learning, the deep learning model performance is one of the most important factors that determine the market revenue of machine learning solution vendors. Thus, the model topology and the core hyperparameters need to be protected as propriety intellectual properties. Recent a few studies demonstrated that it is feasible to steal deep learning model topology, hyperparameter, and training dataset [1], [7], [8]. These attack models, namely *model extraction attack*, *membership inference attack*, and *hyperparmeter stealing attack* leverage side channels through performance counters, cache access timings, and bus probing. Most of these studies exploited unique architectural behaviors of each layer computation to recognize the number and types of layers. For example, DeepSniffer [1] used the execution time, memory reads/writes, and input dependencies of individual kernels (GPU functions) for extracting convolutional neural network (CNN) models. In such attack models, the baseline assumption is that the attacker does not have any knowledge about the model structure, but can measure the architecture activities and is aware that each layer is executed with a set of statically scheduled GPU kernels.

Figure 2: Layer Prediction Errors for ResNet-101 Models

However, we observed that the existing studies overlooked the impact of framework and vendor-specific optimizations. Though most of the deep learning solutions use common libraries such as cuDNN, Winograd, cuBLAS, and so on, the choice of functions is quite diverse in different versions of the same models as can be seen in Figure 1. The figure shows the list of GPU kernels executed commonly and uniquely by three versions of ResNet-101 models. Though all the models were evaluated on the same machine with the same CUDA version, the models used only two common kernels. Such differences are sourced from their preference for algorithms. For example, MXNet models use more Winograd algorithms, TensorFlow models tend to use their GPU backends, and PyTorch models use more cuDNN library functions. Though the core algorithms of these functions may be similar, the different implementations and data formats lead to significantly different architectural behaviors as can be seen in Figure 3. From this figure that shows the execution time and memory reads/writes of the first 20 kernels, we could not observe common patterns across the frameworks. DeepSniffer also reported a similar observation that individual kernel-level recognition performance is not good enough. They used data dependencies among kernels to improve the prediction accuracy. But, we found that some frameworks especially TensorFlow use a few small kernels that do not write to memory. Many of these kernels seem to be some sort of meta-functions that improve efficiencies of the core functions' computations through input transpose and index rearrangement. With these functions that do not have data outputs in the middle of executions, it would be hard to track the data dependencies.

Though the existing studies optimistically(?) predicted that framework difference may have an ignorable impact on model extraction performance, we observed counter results. Figure 2 shows significantly higher errors when applying DeepSniffer attack model for a ResNet-101 downloaded from Google TensorFlow Hub. Table 1 and Table 2 show the error rates and the kernel execution differences of the tested models. DeepSniffer works well with PyTorch models. However, it fails to extract models designed for different frameworks. Even for the PyTorch solution, if vendors add optimizations, the prediction accuracy drops significantly as can be seen in the Nvidia PyTorch Model entry.

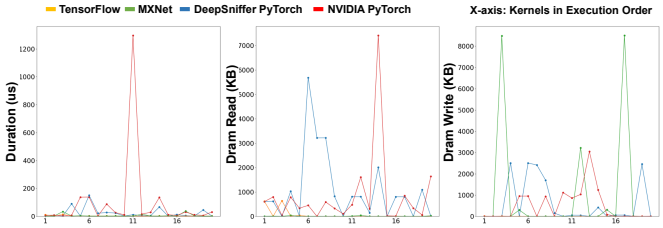

Figure 3: Profile Results of ResNet-50

These results show that attackers should either know the framework and the existence of vendor-specific optimizations or train the attack model with profiling data of various frameworks. This finding is good news for the solution providers because the models do not need to be significantly redesigned to protect from attacks. However, this doesn't mean that you are safe enough because it is not impossible for attackers to understand the differences of the frameworks and optimizations.

TABLE 1: Kernel Profiling & Layer Prediction Error Rates of ResNet-101 Models Implemented with Different Frameworks

Models	Error Rate (LER)	Kernel Seq. Length	# of Unique Kernels
DeepSniffer Original Results [1]	0.067	443	16
DeepSniffer Pytorch Model [2]	0.485	494	16
Google Tensorflow Model [3]	5.733	3926	50
Amazon Mxnet Model [5]	2.988	3043	59

TABLE 2: Kernel Profiling & Layer Prediction Error Rates of ResNet-50 Models Implemented with Different Frameworks

Models	Error Rate (LER)	Kernel Seq. Length	# of Unique Kernels
DeepSniffer Original Results [1]	0.091	222	16
DeepSniffer Pytorch Model [2]	0.567	256	16
Nvidia PyTorch Model [6]	2.628	1235	38
Google Tensorflow Model [4]	6.274	3399	50
Amazon Mxnet Model [5]	6.768	2652	59

References

[1] X. H. et al., "DeepSniffer: A DNN Model Extraction Framework Based on Learning Architectural Hints," in *ASPLOS*, 2020.

[2] DeepSniffer ResNet Models. [Online]. Available: https://github.com/xinghu7788/DeepSniffer/

[3] TensorFlow ResNet Models. [Online]. Available: https://tfhub.dev/google/imagenet/resnet_v1_101/classification/5

[4] TensorFlow ResNet Models. [Online]. Available: https://tfhub.dev/google/imagenet/resnet_v2_50/classification/5

[5] MXNet ResNet Models. [Online]. Available: https://github.com/apache/incubator-mxnet/blob/master/python/mxnet/gluon/model_zoo/vision/resnet.py

[6] NVIDIA ResNet v1.5 for PyTorch. [Online]. Available: https://catalog.ngc.nvidia.com/orgs/nvidia/resources/resnet_50_v1_5_for_pytorch

[7] H. N. et al., "Rendered Insecure: GPU Side Channel Attacks are Practical," in *CCS*, 2018.

[8] M. Y. et al., "Cache Telepathy: Leveraging Shared Resource Attacks to Learn DNN Architectures," in *USENIX Security*, 2020.