A Systematic View of Leakage Risks in Deep Neural Network Systems

Xing Hu [1], Ling Liang [2], chen xiaobing [2], Lei Deng [2], Yu Ji [2], Yufei Ding [2], Zidong Du [2], Qi Guo [2], Timothy Sherwood [2], and Yuan Xie [2]

[1]State Key Lab-oratory of Computer Architecture
[2]Affiliation not available

October 30, 2023

Abstract

As deep neural networks (DNNs) continue their reach into a wide range of application domains, the neural network architecture of DNN models becomes an increasingly sensitive subject, due to either intellectual property protection or risks of adversarial attacks. In observing the large gap between the architectural surfaces exploration and the model integrity study, this paper first presents the formulated schema of the model leakage risks. Then, we propose DeepSniffer, a learning-based model extraction framework, to obtain the complete model architecture information without any prior knowledge of the victim model. It is robust to architectural and system noises introduced by the complex memory hierarchy and diverse run-time system optimizations. Taking GPU platforms as a showcase, DeepSniffer performs model extraction by learning both the architecture-level execution features of kernels and the inter-layer temporal association information introduced by the common practice of DNN design. We demonstrate that DeepSniffer works experimentally in the context of an off-the-shelf Nvidia GPU platform running a variety of DNN models. The extracted models are directly helpful to the attempting of crafting adversarial inputs. The DeepSniffer project has been released in https://github.com/xinghu7788/DeepSniffer.

A Systematic View of Leakage Risks in Deep Neural Network Systems

Xing Hu, Ling Liang, Xiaobing Chen, Lei Deng, Yu Ji, Yufei Ding, Zidong Du, Qi Guo, Timothy Sherwood, Yuan Xie, *Fellow, IEEE*

Abstract—As deep neural networks (DNNs) continue their reach into a wide range of application domains, the neural network architecture of DNN models becomes an increasingly sensitive subject, due to either intellectual property protection or risks of adversarial attacks. In observing the large gap between the architectural surfaces exploration and the model integrity study, this paper first presents the formulated schema of the model leakage risks. Then, we propose DeepSniffer, a learning-based model extraction framework, to obtain the complete model architecture information without any prior knowledge of the victim model. It is robust to architectural and system noises introduced by the complex memory hierarchy and diverse run-time system optimizations. Taking GPU platforms as a showcase, DeepSniffer performs model extraction by learning both the architecture-level execution features of kernels and the inter-layer temporal association information introduced by the common practice of DNN design. We demonstrate that DeepSniffer works experimentally in the context of an off-the-shelf Nvidia GPU platform running a variety of DNN models. The extracted models are directly helpful to the attempting of crafting adversarial inputs. The DeepSniffer project has been released in https://github.com/xinghu7788/DeepSniffer.

Index Terms—domain-specific architecture; deep learning security; model security;

1 Introduction

Machine learning approaches, especially deep neural networks (DNNs), are transforming a wide range of application domains, such as computer vision [1], [2], speech recognition [3], and language processing [4]. In computer vision, specifically, maturing DNN technologies have achieved better prediction accuracy than human-beings [5], which start to power existing industries. For example, autonomous driving, whose market is predicted to leap to $77 billion (25% of the whole automotive market) by 2035 [6], has attracted the attention of giants including Tesla, Audi, and Waymo [7]. Due to the promising opportunities and rising importance of DNN techniques, DNN vulnerability emerges as an important problem – especially for mission-critical applications [8].

The high level structure of a DNN model might be described by a few, but very important, characteristics (e.g. the connection topology and dimensionality of the network). These characteristics shape the behavior of the resulting model is significant ways and, if learned by an adversary, give them a significant foothold into exploring and exploiting other vulnerabilities. By *extracting* such information, attackers can not only counterfeit the intellectual property of the DNN design, but can also perform more efficient adversarial attacks that manipulate the DNN model to output malicious outputs [9], [10]. Due to the significance of these model characteristics, model extraction attacks represent an important class of threat to consider when deploying a DNN system accessible to untrusted parties [10], [11], [12].

Algorithm-level studies mainly perform model extraction through detecting the decision boundary of the victim black-box DNN models [10]. However, such approaches demand significant computational resources and huge time overhead. Even worse, this approach cannot accommodate state-of-the-art DNNs with complex topologies, e.g., DenseNet [13] and ResNet [5], due to the enlarged search space of possible network architectures.

However, DNN models are deployed in the DNN system stack, which may reveal the architectural events or behaviors (referred to as architectural hints in the following) during execution. Architectural execution hints and algorithmic decision boundary collaboratively depict the internal model characteristics. Based on our prior work [14], This work aims to explore the leakage risks in DNN systems in a systematic way and answer the following fundamental questions: how do the algorithmic vulnerabilities and system vulnerabilities combine to produce model leakage risks and how do we then quantitatively evaluate the increased leakage risks. We also showcase an end-to-end attack study, which indicates the synergy of algorithmic vulnerability and system vulnerability make this problem practical and urgent.

First, we experimentally explore the criticality of model characteristics and present the formulation of the model leakage risks in DNN systems. The key idea is to evaluate the gap between the two data representations of the model and the system attack surface. For instance, if the adversary-visible architectural hints in one DNN system have a high

The preliminary version is published in ASPLOS2020. *Xing Hu, Xiaobing Chen, Zidong Du, Qi Guo are with State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China.(E-mail:huxing,chenxiaobing,duzidong,guoqi@ict.ac.cn) Ling Liang, Lei Deng and Yuan Xie are with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, USA. (E-mail: lingliang, leideng, yuanxie@cs.ucsb.edu). Yufei Ding and Timothy Sherwood are with the Department of Computer Science, University of California, Santa Barbara, USA. (E-mail: yufeiding@cs.ucsb.edu, sherwood@cs.ucsb.edu). Yu Ji is with Tsinghua University, China, also with University of California, Santa Barbara, USA. (E-mail: jiy15@mails.tsinghua.edu.cn).*

correlation with model characteristics, this DNN system has higher leakage risks. Such a simple, yet effective problem formulation can be generally applicable to many DNN explorational attacks.

Further, we propose **DeepSniffer**, a framework to obtain the **complete** model architecture with **no prior knowledge** of the victim model and it is **robust** to system-level and architecture-level noises. The complete model architecture extraction includes the following steps: *run-time layer sequence identification*, *layer topology reconstruction*, and *dimension size estimation*. Among these steps, the *run-time layer sequence identification* is the most fundamental one and is missing in previous work [15], [16], [17], since they either take the layer type or neural network architecture as known information or impractically assume that the single-layer architecture hints can be easily distinguished. We map the *run-time layer sequence identification* to a sequence-to-sequence prediction problem and address it using learning-based approaches. One of the most important differences that DeepSniffer distinguishes from previous work is that it decouples layer sequence prediction from dimension size prediction, thus being more generally applicable and robust to noises.

We also propose and experimentally demonstrate end-to-end attacks in the context of an off-the-shelf Nvidia GPU platform with full system stacks, which urges the demand to design secure architecture and system to ensure DNN security. In summary, we make the following contributions:

- We propose the scheme of exploring the model leakage risks of DNN systems. We analyze the criticality of various model characteristics, and formalize the architecture leakage risks of model extraction.
- We propose DeepSniffer framework to utilize architectural hints for model extraction. It maps the fundamental step of model extraction, i.e, run-time layer sequence identification, to a sequence-to-sequence prediction problem and adopts learning-based approaches to perform accurate and robust run-time layer sequence prediction.
- We showcase the effectiveness of DeepSniffer to conduct model extraction under one specific attack scenario. We experimentally demonstrate our methodologies on an off-the-shelf GPU platform. With the easy-to-get off-chip bus communication information, the extracted network architectures exhibit very small differences from those of the victim DNN models.
- We perform two end-to-end attack scenarios to show that the extracted neural network architectures boost adversarial attack effectiveness, improving the attack success rate (ASR) significantly. For adversarial example attack, the ASR is boosted from 14.6%∼25.5% to 75.9% compared to cases without neural network architecture knowledge. For adversarial patch attack, the ASR is boosted from 41% to 83%.

2 A Systematic Approach to DNN Vulnerabilities

In this section, we described a more systematic approach to reasoning about DNN vulnerabilities, as shown in Figure 1. We specifically consider a model under which an algorithmic DNN is deployed on a DNN platform that takes in the environmental inputs and executes in a way that produces predicted results as outputs. Thus, the DNN system, including hardware platform, OS/VM/docker, and framework, provide the actual attack surfaces for the adversary who either spies on the model information or tempers the input data.

To explore the relation and synergy between the architectural vulnerability and algorithmic model vulnerability, this work aims to answer the following three essential questions:
1) What model characteristics are more critical in terms of DNN security? The target protection objects in DNN systems are not a well understood as those in traditional security problems. For example, it is clear that secret keys are highly critical for cybersecurity and must be protected with highest priority. However, it is not obvious what is the most critical data of a DNN model. Network architecture, model parameters, and hyper-parameters determine the decision boundary of DNN models, all of which might be leveraged by the adversary to perform DNN attacks.
2) What attack surfaces are exposed in DNN systems? Attack surfaces give a chance for the adversary to explore or temper the model characteristics. We systematize the potential attack surfaces through the DNN system stack in both cloud and edge scenarios.
3) How do we formalize the leakage risks of DNN systems? The state of DNN deployment evolves rapidly with dedicated system stacks and brand new application scenarios introduced on incredibly short time scales. It is essential to formalize how the attack surfaces damage the model characteristics privacy or integrity. Many prior attack studies claim that they reduce the search spaces of the model characteristics. However, the quantitative measure of the attack effectiveness is missing and it is unclear whether the system attack really affects the model security.

The following subsections explore these questions through a mix of mathematical analysis and experiment, and set the stage for understanding a deeper dive into a powerful new approach to model extraction in later sections.

2.1 Model Vulnerability

Many algorithm-level studies contribute to explore the vulnerability of DNN models, which can be classified into two categories for different spotlights on model privacy and model integrity. The former category of studies aims to explore the internal information of the DNN models. The latter aims to induce the model to output unexpected results by either adversarial inputs or tempering the model training process. This work focuses on the former one, model extraction attack, since obtaining model internal information can help to turn a black-box integrity attack into a white or grey-box attack with astonishing boosted attack success rate, and thus is essential for further advanced attack methodologies to damage model integrity.

2.1.1 Model Characteristics

Model extraction attacks aim to explore the model characteristics of DNNs for establishing a near-equivalent DNN model with a similar decision boundary [11]. Model characteristics include: **Network sketch** consists of layer types and

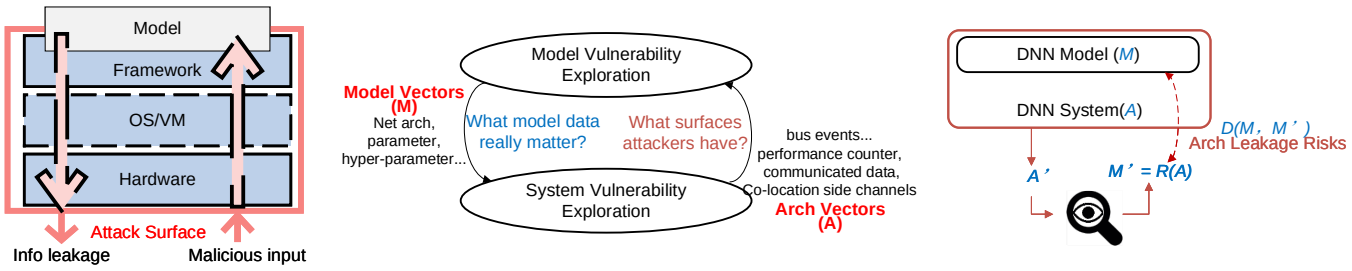


Fig. 1. Schema overview. (a) DNN system stack; (b) The correlation between model security and system vulnerability; (c) The formulated schema of model leakage risks in DNN system.

the connection topology between layers. **Dimension sizes** include layer dimensions (the number of channels, feature map size, weight kernel size, stride, and padding in each layer). The network sketch and dimension sizes constitute the network architecture knowledge. **Parameters** include the weights, biases, and Batch Normalization (BN) parameters. They are updated during the stochastic gradient descent (SGD) in the training process. Hyper-parameters refer to the configurations during training, including the learning rate, regularization factors, and momentum coefficients, etc. All these model characteristics may have their own impact on the decision boundary of a DNN model. With the extracted model characteristics, the adversary is able to build the substitute models for adversarial examples generation and then use these examples to attack the victim black-box model [8], [18], [19], [20].

2.1.2 Security Sensitivity of Model Characteristics

We perform experiments to empirically show how does the knowledge of model characteristics affect the effectiveness of integrity attack. The detailed experimental setup is Section 6.3. With ResNet18 as the golden victim model, we test the adversarial attacks under the following scenarios: 1) **'White-box'** with all the details of the victim DNN model; 2) **'W/O-Para'** refers to the scenarios perform an adversarial attack with without the knowledge of parameters (weights and BN parameters), but knowing the network architectures exactly; 3)**'W/O-Dim'** refers to the scenarios perform adversarial attack without the knowledge of parameters and precise dimension sizes of network architectures, but knowing the network sketch with different layer dimensions; 4) **'W/O-NetArch'** refers to the scenario perform adversarial attack without any information of the network architecture, even the network model family. Results show that, white box attack has very high attack success rate (nearly 100% without adversarial training), which indicates the importance of protecting model privacy. Without knowledge of parameters, we can retrain the neural network models based on the network architecture information, the attack success rate drops to 70% on average. With the network sketch (layer type and interconnection), the attack success rate remains about 70%. Without the network sketch, an adversary has to use random substitute model architectures that have distinct decision boundary with the victim. Hence the attack success rate drops drastically to 17% on average. The results indicate that the network sketch is very important for model privacy protection and adversarial attack effectiveness. Such an observation is consistent with prior studies [9], [10], which reveal that network sketch is essential for attack transferability.

In summary, among all of the model characteristics, the network architecture is the most fundamental one for DNN security. In addition to boosting the effectiveness of advanced adversarial attacks, previous studies also demonstrate that with the knowledge of the network architecture, the adversary is able to explore the extraction of model parameters, hyper-parameters, and even training data [11], [12]. In this work, we select the network architecture as the target information, because network architecture is not both fundamental for model security and more relevant to the DNN system and architectural hints.

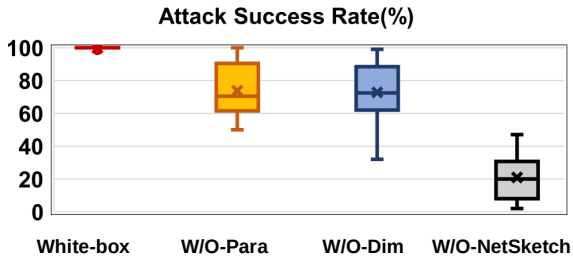


Fig. 2. Attack success rate distribution with different knowledge of neural network models.

2.2 Architectural Leakage Surfaces

In the real deployment scenarios, the DNN system stack provides additional attack surfaces other than the visible input and output pairs of the victim model. Across the system stack, both hardware-level, framework-level, and service-level vulnerabilities provide threats knobs for the adversary to explore the internal states of DNN models:

Hardware vulnerabilities. Hardware platforms may reveal architectural vectors through the physical signal snooping in buses or memory, or side channel attacks in shared queues/memory. For instances, the attacker can obtain the bus events and the communication data through bus snooping. Based on the side channel of the power consumption, the adversary can infer the utilization rate of processing units or the input vectors. With the side channel of operation execution time, the control flow, data dependency, cache miss rate, or the pipeline stall may be inferred by the adversary.

Framework vulnerabilities: Deep learning frameworks provide basic operations and commonly-used building blocks of neural network layers, which ease the programming burden of the developers. However, the implementation complexity of the framework and libraries may lead to software vulnerabilities, which are not only introduced by the deep learning framework, but also the dependency packages used by them. For instance, librosa, numpy, OpenCV, and Libjasper are commonly used to accomplish tasks such as video and audio processing or model data representations in the deep learning framework. Several types of flaws in the libraries or frameworks, such as heap overflow, integer overflow, result in denial-of-service attacks, evasion attacks, or system compromises, as summarized in previous work [21].
Service vulnerabilities: DNN techniques can be deployed in both the edge devices and cloud with machine learning as a service (MaaLS). In addition to the edge devices that are physically accessible, the cloud services may exhibit shared tenancy vulnerabilities and poor access control vulnerabilities. Additionally, the APIs of training services also reveal the decision boundary information.

TABLE 1
Vulnerabilities in the DL system stack

Service	Shared tenancy vulnerabilities; Poor access control vulnerabilities; API information leakage
Framework	Integer overflow; Heap overflow; Crash; DoS
Hardware	Bus snooping; Memory eavesdropping; Side channels: Time or delay; Power and current; Performance counter; EM emission; Optical

As a brief summary, different architectural vectors are available under different attack surfaces in the system stack (as summarized in Table 1). However, it is unclear how these architectural vectors relevant to model characteristics and what risks do they bring towards DNN models. Hence, we present the formulated schema of the model leakage risks in DNN systems in the following subsection.

2.3 Quantitatively Analysis of Attack Objective

DNN architecture provides additional attack surfaces for advanced and practical attack models, while model security research inspires the system/architecture security work to understand what are the critical model features for attack or protection. It is essential and urgent to explore the deep learning system security with collaboratively taking care of the model security and system security. In this work, we consider how the system leakage surface reveals the internal model characteristics which assist the adversary to perform model extraction attack.

Specifically, we propose the formalization description of model extraction as illustrated in Figure 1c. A DNN model M is described as a computational graph in the deep learning framework and being processed in the hardware accelerator with triggering architecture events A. The adversary may obtain the architecture hint vectors A' in the attack surfaces. The model extraction technique R tries to reverse the architecture vectors to obtain the predicted model $M' = R(A')$, with the minimum distance between M' and M. The distance between M' and M reveals the architectural leakage risks in this model extraction attack. This problem can be derived from the optimization of cross entropy between M' and M.

In the following, We propose DeepSniffer framework to perform model extraction based on available architecture hints. Then, we take a demonstrative case study to show the overall DNN system paradigm by proposing the attack model and effective attack methodology. We not only quantitatively evaluate the effectiveness of the proposed method based on such formalization, but also showcase an end-to-end attack to validate the rationality of the methodology.

3 A Case Study of Model Extraction

3.1 Attack Model

In this work, we showcase the adoption of DeepSniffer in one attack scenario, the detailed threat model of which is as shown in Figure 3. It is a common practice to train a powerful DNN model in the cloud and deploy it in many edge devices. When the attacker physically accesses one edge platform encapsulating a victim DNN model for model extraction, it is able to attack all the other devices sharing the same DNN model.Such "Hack-one-Attack-all-others" threats are practical and destructive.

We adopt the commonly-used GPU platforms as the basic architecture of edge devices, which are widely used in many industrial products, including most of the existing L3 autopilot systems [7], [22]. In such an architecture, the CPU and GPU are connected by the PCIe bus, and the host and device memories are attached to the CPU and GPU through DDR and GDDR memory buses respectively, as shown in Figure 3b [23]. With bus snooping technologies [24], the adversary passively monitors the memory bus and PCIe events and obtain the following architectural hints:by observing the memory access trace through the GDDR memory bus, the adversary obtains the kernel read/write access volume (R_v/W_v) and memory address traces. Since there are control messages passing through the PCIe bus when a kernel is launched and completed, the adversary can determine the kernel execution latency (Exe_{Lat}) by monitoring the time between kernel launching and completing.

Note that, we consider a threat model in which the adversary does not have any knowledge about the victim models including what family the DNN models belong to, what software code those models are implemented with, or any other information about the operation of the device under attack that is not directly exposed through externally accessible connections. The extraction attack is fully passive and only has the ability to observe architectural side-channel information over time.

3.2 Challenges

It is a challenging task to recover the model architecture through the architectural hints because of the system and architectural noises through the DNN system stack. The detailed system stack is shown in Figure 4a. The computational graph of a DNN model is processed by the deep learning framework, hardware primitive libraries, and hardware platform. First, the deep learning framework optimizes the network architecture of the DNN model to

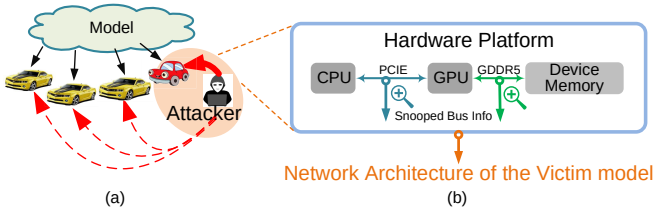


Fig. 3. Illustration of the attack model. (a) Hack-one, attack-all-others with the extracted model. (b) GPU platform overview.

form a framework-level computational graph of layers that is a representation of a composite function as a graph of connected layer operations. The framework then transforms this high-level computational graph abstraction to hardware primitives of run-time layer execution sequence. Then, the run-time hardware primitive libraries, such as cuDNN library [25], launch the well-optimized kernel sequence according to the layer type. Finally, such kernel sequences are executed on the hardware platform, which exhibits architectural hints, including the memory access pattern and the kernel execution latency. Both the architectural design and the run-time system stack introduces noises in the kernel-level architecture vectors which hinder the attacker to perform accurately model extraction.

Architectural noise: The comprehensive memory system optimization, such as the shuffling address mapping and complex memory hierarchy, raises the difficulty to obtain and identify the complete memory traces for accurate dimension size estimation.

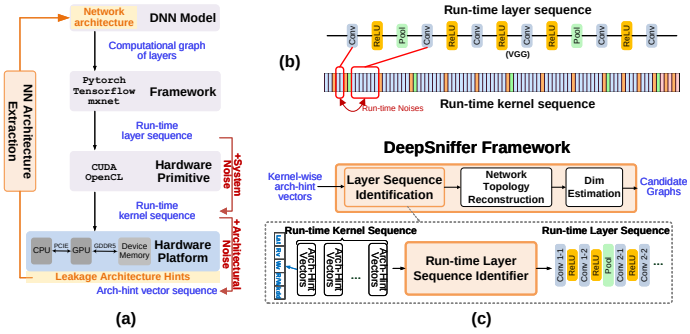


Fig. 4. (a) Computational graph transformations through DNN system stack. (b) System noise during run-time layer sequence to kernel sequence transformations. (c) DeepSniffer overview.

System noise: System run-time dynamics introduces the noises to the architectural hint sequence in the further step. DNN layers are transformed into GPU-kernels dynamically during run-time, with various implementations (e.g. Winograd and FFT). The dynamic, not one-to-one correspondence mapping between layers and kernels makes it difficult to even figure out the number of layers and layer boundary in a kernel sequence, not to mention the corresponding layer dimension size. To analyze the influence of such dynamics, we perform experiments on an off-the-shelf GPU platform with PyTorch [26] and cuDNN [25]. Figure 4b shows the transformations from the layer sequence of DNN models to the run-time kernel sequence, taking the VGG and Inception as illustrative examples. We observe that run-time kernel implementations vary across different models and even across time for the same model. For example, in the run-time kernel sequence of Figure 4b, the blue bars represent Conv kernels. The boxed two sets of Conv kernels in the VGG kernel sequence are different from each other with different implementations.

4 DeepSniffer Design

We envision that accurately predicting the dimension sizes for identifying the model architecture is infeasible in complex system stacks with the existence of architectural and system noises, while such a methodology is commonly used in previous studies [16], [17] that target a specific architecture. Instead, we propose the DeepSniffer framework for effective model extraction without relying on the accurate dimension parameter estimation that is hence more robust and generally applicable.

Design Overview: The design overview of DeepSniffer is shown in Figure 4c, with the following three stages: 1) run-time layer sequence identification; 2) layer topology reconstruction; and 3) dimension size estimation. DeepSniffer proposes a run-time layer sequence identification methodology that learns the single kernel feature of architectural hints during kernel execution and inter-kernel/layer context probability for higher prediction accuracy. With the predicted layer sequence, DeepSniff then performs the layer topology reconstruction and dimension size estimation to get the complete DNN architecture.

As the most fundamental step, run-time layer sequence identification translates the kernel-grained architectural hint sequence back to the run-time layer sequence. Based on the observation that it can be mapped to a typical sequence-to-sequence translation problem, we propose a run-time layer sequence identification methodology based on deep learning techniques that learn both the single kernel feature and inter-kernel/layer context association for high prediction accuracy. Unlike previous work [16], [17], the run-time layer sequence prediction does not rely on the exact calculation of the dimension size parameters and but aims to learn the computational graph transformations through the systems stack. The following sections introduce these three steps of model extraction in DeepSniffer in details.

4.1 Run-time Layer Sequence Identification

After comprehensively investigating modern DNN models, we consider the following layers in this work: Conv (convolution), FC (fully-connected), BN (batch normalization), ReLU (rectified linear unit), Pool (pooling), Add, and Concat (concatenation), because most of the state-of-art neural network architectures can be represented by these basic layers [2], [5], [27], [28], [29], [30]. Note that it is easy to integrate other layers into DeepSniffer if necessary.

4.1.1 *Problem Formalization*

Formally, the run-time layer sequence identification problem can be described as follows: We obtain the architectural hint vectors of kernel sequence X with temporal length of T as an input. At each time step t, kernel feature X_t $(0 \leq t < T)$ can be described as a multiple-dimension tuple of architectural hints. Note that this tuple can be

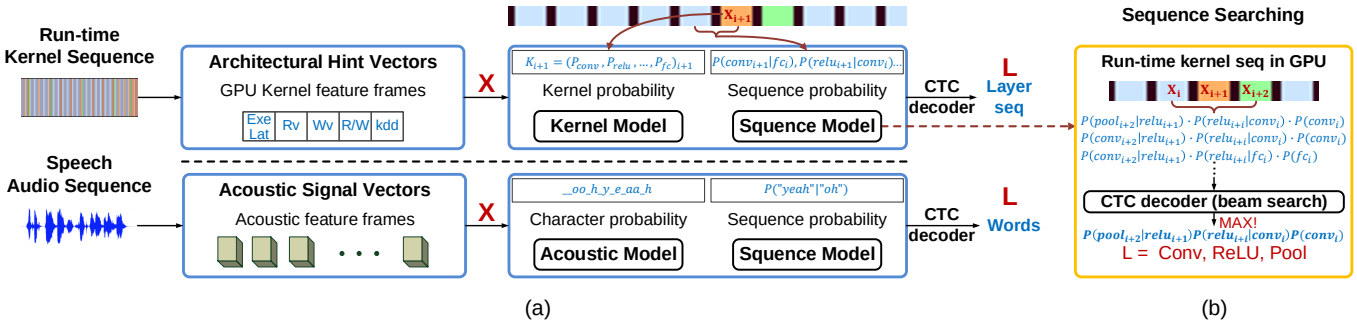


Fig. 5. Context-aware layer sequence identification. (a) Identification flow (map the layer sequence identification to a speech recognition problem); (b) CTC decoder searches the sequence with highest probability.

extended if the attack scenarios expose more architectural hints. The label space is a set of layer sequences comprised of all typical layers. The goal is to train a layer sequence identifier h to predict the run-time layer sequence (L) having the minimal distance to the ground-truth layer sequence (L^*).

The run-time layer sequence identification involves two internal correlation models: *kernel model* and *layer-sequence model*. The *kernel model* correlates the relationship between the architectural hints and the kernel type. The *layer-sequence model* correlates the probabilistic distribution between the layers. We observe that the process of predicting the run-time layer sequence is similar to that of the speech recognition, as shown in Figure 5, which also involves two parts: an acoustic model converting acoustic signals to phonemes and a language model computing sequence probabilistic distribution on the words or sentences. Therefore this problem can be mapped to a speech recognition problem due to the high similarity of these two problems. Based on this insight, DeepSniffer leverages the auto speech recognition (ASR) techniques [31], [32] as a tool for run-time layer sequence identification. In the following subsection, we first show the intrinsic features of these two models.

4.1.2 Kernel and Layer Features

Architectural Hints of A Single Kernel. During DNN model execution, every layer conducts a series of kernel operation(s) for the input data and delivers output results to the next kernel(s), thus dataflow volume through kernels and the computation complexity constitutes the major parts of kernel features. As introduced in Section 3, we can determine the following architecture hints in Attack Scenario-1: 1) Kernel execution time ($\mathbf{Exe_{Lat}}$); 2) The kernel read volume ($\mathbf{R_v}$) and write volume ($\mathbf{W_v}$) through the memory bus; We can also calculate: 3) Input/output data volume ratio ($\mathbf{I_v/O_v}$) of each kernel, where the output volume (O_v) is equal to the write volume of this kernel and input volume (I_v) is equal to the write volume of the previously executed kernel. For the bus snooping attack scenario, we additionally use 4) kernel dependency distance ($\mathbf{kdd}$) to indicate the topology influence. kdd is defined as the maximum distance between this kernel and the previous dependent kernels during the kernel sequence execution, which is a metric to encode the layer topology information in the kernel features. We regard this tuple (Exe_{Lat}, R_v, W_v, I_v/O_v, kdd) as one frame of kernel features.

We observe that although the kernels of different layers have their own features according to their functionality, it is still challenging to predict which layer a kernel belongs to, based on *kernel model* only. Our experiment results show that, on average, 30% of kernels are identified incorrectly with the executed features only and this error rate increases drastically with deeper network architectures (above 50%). The details of the experimental results are shown in Section 5.2.3. In summary, it is challenging to accurately predict layer sequence by considering single kernel architectural features only.

Inter-Layer Sequence Context. We observe that the temporal association of the layer sequence offers the opportunity for better model extraction. Specifically, given the previous layer, there is a non-uniform likelihood for the following layer type, which is referred to as the *inter-layer context*. Such temporal association information between layers (aka. layer context) is inherently brought by the DNN model design philosophy. For example, there is a small likelihood that an FC layer follows a Conv layer in DNN models, because it does not make sense to have two consecutive linear transformation layers. Recalling the design philosophy of some typical NN models, e.g., VGG [2], ResNet [5], GoogleNet [27], and DenseNet [13], there are some common empirical evidence in building the network architecture: 1) the architecture consists of several basic blocks iteratively connected, 2) the basic blocks usually include linear operation first (Conv, FC), possibly the following normalization to improve the convergence (BN), then non-linear transformation (ReLU), then possible down-sampling of the feature map (Pool), and possible tensor reduction or merge (Add, Concat).

Although DNN architectures evolve rapidly, the basic design philosophy remains the same. Even for the state-of-the-art autoML technical direction of Neural Architecture Search (NAS), which uses the reinforcement learning search method to optimize the DNN architecture, it also follows the similar empirical experience [29]. Therefore, such layer context generally exists in the network architecture design, which can be leveraged for layer identification.

4.1.3 Context-aware Layer Sequence Identification

Based on the analysis of kernel and inter-layer features, we adopt the Long Short-Term Memory (LSTM) model with a Connectionist Temporal Classification (CTC) decoder to build the context-aware layer sequence identifier h. The combination of the LSTM model and CTC decoder

is commonly used in automatic speech recognition [31], [32]. As shown in Figure 5, given the input sequence $X = (X_1, .., X_T)$, the object function of training layer sequence identifier h is to minimize the CTC cost for a given target layer sequence L^*. The CTC cost is calculated as follows:

$$cost\,(X) = -logP\,(L^*|h\,(X)) \quad (1)$$

where $P\,(L^*|h\,(X))$ denotes the total probability of an emission result L^* in the presence of input X.

An Example for Layer Sequence Prediction. The layer sequence prediction workflow is simplified as shown in Figure 5a. For the (i)th frame of the kernel sequence, its kernel architectural hint vectors are X_i. The layer sequence identifier first conducts the kernel classification based on X_i and obtains its probability distribution K_i of being Conv, ReLU, BN, Pool, Concat, Add, and FC.

$$K_i = \{P_{conv}, P_{relu}, P_{bn}, P_{pool}, P_{concat}, P_{add}, P_{fc}\}_i \quad (2)$$

The layer sequence identifier then uses a sequence model to estimate the conditional probability with the probability distribution of prior kernels: (K_0, K_1, ..., K_i). With the whole kernel feature sequence, the CTC decoder uses the beam search to find out the layer sequence with the largest conditional possibility as output (L). The layer sequence predictor has better prediction accuracy when there is less difference between the predicted layer sequence (L) and the ground-truth layer sequence (L*). The experimental details of the model training, validation, and testing are introduced in Section 5.1.

In the further step, we illustrate the detailed working mechanisms of a simplified CTC decoder in Figure 5b. In the monitor window (X_i, X_{i+1}, X_{i+2}), the CTC decoder searches throughout the searching space containing all of the potential layer sequences, such as (Conv, ReLU, Pool), (FC, ReLU, Conv), (Conv, Conv, Conv), etc. Then it outputs the layer sequence with the largest probability as output, which is (Conv, ReLU, Pool) in this case. In real cases, the CTC decoder is more complex and it considers the reduplication removing and adopts advanced searching algorithms [31], [32].

4.2 Layer Topology Reconstruction

DeepSniffer then reconstructs the layer topology by monitoring the memory access pattern of the layers. In this subsection, we show how the memory traffic reveals the interconnections between layers.

Before going deeper, we first explain the definition of the layer interconnections. In the computational graph of a neural network architecture, if the feature map data of layer a is fed as the input of layer b, there should be a directed topology connection from a to b. Since this work focuses on the inference stage, there is only forward propagation across the whole network architecture. We first analyze the cache behaviors of feature map data and report the following observations:

Observation-1: Only feature map data (activation data) can introduce read-after-write (RAW) memory access pattern in the memory bus. There are several types of memory traffic data during the DNN inference: input images, weight parameters, and feature map data. Only feature map data is updated during inference. Feature map data is written to the memory hierarchy and read as the input data of the following layer(s). The input image and parameter data are not updated during the entire inference procedure. Therefore, the RAW memory access pattern is introduced only by the feature map data.

Observation-2: Feature map data has a very high possibility to introduce RAW access pattern, especially for the convergent and divergent layers. We examine the read cache misses of the feature map data in kernels of convergent and divergent layers at branches. The convergent layer receives feature map data from layers in different branches. For example, Add and Concat are the main convergent layers in neural network models. The read cache-miss rate of an Add layer is more than 98% and that of a Concat layer is more than 50%, as shown in Figure 6. The divergent layer outputs feature map data to several successor layers on different branches. We observe that GPU kernels execute the layers through one branch by one branch manner. Moreover, the memory traffic volume in the convergent layer and the successors of divergent layers have much higher memory traffic volume than the ground-truth weight data size. Since the CUDA library implements extreme data reuse optimizations that prioritize the weight tensor, the feature map needs to be flushed into memory and then read again due to a long reuse distance [23].

These two observations indicate that the RAW access pattern can be used to determine the interconnections among different layers. We propose a layer topology reconstruction scheme as follows: DeepSniffer scans all the layers in the run-time layer sequence. For layer i, all the addresses of its read requests constitute $ReadSet_i$ and that of write requests constitute $WriteSet_i$. DeepSniffer searches all its antecedent layers $layer_j \in (layer_0, layer_1, ..., layer_{i-1})$ in the sequence and checks whether $ReadSet_i \cap WriteSet_j = \emptyset$. If it is not $\emptyset$, DeepSniffer adds the connection between layer i and layer j. At the end, DeepSniffer checks whether there is any layer that doesn't have any successors in the topology, and eliminates the orphan layers by adding the connection to their following layer in the run-time layer sequence.

Note that, we do not require the complete memory address trace of all the feature map data, but only partial segments in order to identify the connections between different layers, which is robust to the memory traffic filtering.

4.3 Dimension Size Estimation

After completing the first two steps, we obtain the skeleton of the neural network architecture. Then DeepSniffer performs the dimension size estimation. Noted, we do not need the exact precise dimension sizes according to the analysis of Section 2.1.2, but just make a rough estimation about the potential dimension spaces. The dimension size estimation includes the following two steps: 1) Layer feature map size prediction; 2) Dimension space exploration. In this section, we explain how to estimate the dimension size parameters according to the memory traffics.

Layer feature map size prediction is based on the estimation of ReLU sizes. We observe that, for most DNN models, ReLU kernels have a stable high cache miss rate, surpassing

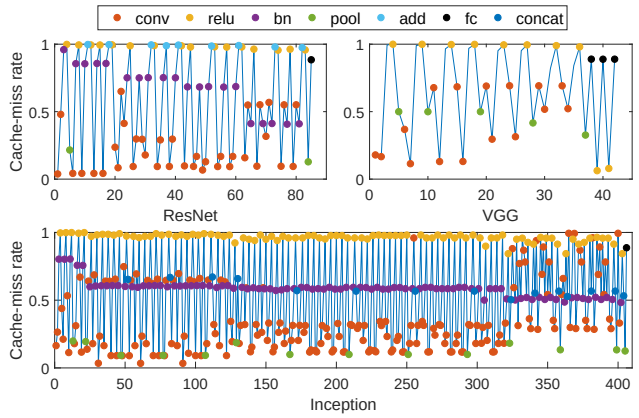


Fig. 6. Read cache-miss rate of kernels in VGG11, ResNet18, and Inception.

98%, as shown in Figure 6. Hence, the read volume through the bus R_v is almost the same as the input feature map size of the DNN model. Then the write volume W_v can be estimated which is equal to R_v. Based on this observation, we can obtain the input and output sizes of ReLU layers. Then, given the input size of a ReLU layer, the output size of the previous BN/Add/Conv/FC layer and the input size of the next Conv/FC layer can be estimated. Since the ReLU layer is almost a standard layer in every basic block, the feature map sizes of the layers in the victim model can be estimated by broadcasting the ReLU size to their adjacent layers.

With the constructed layer topology and input/output size of every layer, we calculate the following dimension space: the input (output) channel size IC_i (OC_i), the input (output) height IH_i (OH_i), the input (output) width IW_i (OW_i), the weight size ($K \times K$), and the convolution padding P and stride S.

Based on the fact that the input size of each layer is the same as the output size of the previous layer, and the basic tensor computation constraints,we are able to search the possible solution for every layer. Since we target the computer vision applications, the $IC_0 = 3$. We assume the feature map height and weight are the same and stride=1 (which are the common configuration in lots of DNN models). By iterating over possible kernel sizes (1, 3, 5 ..), we can estimate the other configuration parameters with the basic tensor computing constraints.

Notice that, we neither assure nor need to obtain the precise dimension size parameters. Instead, we randomly select the possible sets of dimension parameters that satisfy the tensor computation constraints as the configuration of the extracted DNN architecture. With the accurate extracted network sketch, we can achieve good attack effectiveness though the dimension parameters are different from the victim model (More analysis in Section 6.3).

5 Model Extraction Effectiveness

In this section, we evaluate the effectiveness of the proposed network architecture extraction technique.

5.1 Setup

To validate the feasibility of stealing the memory information during inference execution, we conduct the experiments on the hardware platform equipped with Nvidia K40 GPU [33]. The DNN models are implemented based on PyTorch framework [26], with CUDA8.0 [34] and cuDNN optimization library [25]. We use the GPU performance counter [35] to emulate bus snooping for kernel execution latency, kernel write, and read access volume information collection.

As an initial step for network architecture extraction, we first train the layer sequence identifier based on an LSTM-CTC model for layer sequence identification. The detailed training procedure is as follows.

Training: In order to prepare the training data, we first generate 8500 random computational graphs of DNN models and obtain the kernel architectural features.Two kinds of randomness are considered during random graph generation: topological randomness and dimensional randomness. At every step, the generator randomly selects one type of block from sequential, Add, and Concat blocks. The sequential block candidates include (Conv, ReLU), (FC, ReLU), and (Conv, ReLU, Pool) with or without BN. The FC layer only occurs when the feature map size is smaller than a threshold. The Add block is randomly built based on the sequential blocks with shortcut connections. The Concat block is built with randomly generated subtrack number, possibly within Add blocks and sequential blocks. The dimension size parameters – such as the channel, stride, padding, and weight size of Conv and neuron size of FC layer – are randomly generated to improve the diversity of the random graphs. The input size of the first layer and the output size of the last layer are fixed during random graph generation, considering that they are usually fixed in one specific target platform. We randomly select 80% of the random graphs as the ***training set*** and other 20% as the ***validation set*** to validate whether the training is overfitting or not.

Testing: To verify the effectiveness and generalization of our layer sequence identifier framework, we examine various commonly-used DNN models as the ***test set***, including VGG [2], ResNet [5], and Nasnet [29] to cover the representative state-of-the-art DNN models.

5.2 Layer Sequence Identification Accuracy

In this section, we first evaluate the layer sequence identification accuracy. Then we analyze the importance of the layer context information and the influence of noises in architectural hints.

5.2.1 Evaluation Metric

We quantify the prediction accuracy with the layer prediction error rate (LER), similar to those being used in speech recognition problems. It is the mean normalized edit distance between the predicted sequence and label sequence which quantifies the prediction accuracy [31], [32]. The detailed prediction calculation is as follows [32].

$$LER = \frac{ED\left(L, L^*\right)}{|L^*|} \tag{3}$$

where $ED\left(L, L^*\right)$ is the edit distance between the predicted layer sequences L and the ground-truth layer sequence L^*,

i.e. the minimum number of insertions, substitutions, and deletions required to change L into L^*. $|L^*|$ is the length of the ground-truth layer sequence.

5.2.2 Run-time Layer Sequence Prediction Accuracy

We first evaluate the accuracy of the randomly generated DNN models.For DNN models with chained topology only, the average prediction error rate of layer sequence identification is about 0.06. For neural networks with shortcut and concat topology, the average LER of layer sequence identification is about 0.06 and 0.1, respectively. The LER is higher for DNN models with more complex topology.

Furthermore, we evaluate the accuracy in identifying several state-of-the-art DNN models, as shown in Table 3. The LER of AlexNet and VGG19 are 0.02 and 0.017 respectively, as shown in Table 3. For ResNet families, the prediction LER is lower than 0.07. For NasNet, the LER increases slightly due to the much deeper and complex connections. We take ResNet34 as an example to present the detailed results in Table 2. In summary, our proposed method is generally effective in correctly identifying the layer sequence. There may exist a small deviation between the predicted sequence and ground-truth sequence. Thus we conduct end-to-end experiments in Section 6, which shows that the extracted neural network architecture, although having a little deviation from the victim architecture, can still boost the attacking effectiveness.

TABLE 2
Identification results

DNN Model	Ground-truth Sequence	Predicted Sequence
ResNet18 (ErrorRate 0.032)	Conv BN ReLU MaxPool Conv BN ReLU Conv BN ADD ReLU Conv BN ReLU Conv BN ADD ReLU Conv BN ReLU Conv BN Conv BN Add ReLU Conv BN ReLU Conv BN ADD ReLU Conv BN ReLU Conv BN Conv BN Add ReLU Conv BN ReLU Conv BN ADD ReLU Conv BN ReLU Conv BN Conv BN Add ReLU Conv BN ReLU Conv BN FC	Conv ~~BN~~ ReLU MaxPool Conv BN ReLU Conv BN ADD ReLU Conv BN ReLU Conv BN ADD ReLU Conv BN ReLU Conv BN Conv BN Add ReLU Conv BN ReLU Conv BN ADD ReLU Conv BN ReLU Conv BN Conv BN Add ReLU Conv BN ReLU Conv BN ReLU FC

TABLE 3
Prediction error rate on typical networks.

Chained Topology		Complex Topology			
AlexNet	VGG19	ResNet34	ResNet101	ResNet152	Nasnet_large
0.020	0.017	0.040	0.067	0.068	0.144

5.2.3 Why is Inter-Layer Context Important?

To analyze the importance of inter-layer context information in this section, we compare the prediction error rate of two methods: a context-aware identifier considering layer context in our work and a single-kernel identifier based on a multi-layered perception model. The key difference between these two identifiers is whether including the sequence model in Figure 5.

We compare both the prediction error rate along the identifier training processes from the 1st to 100th epochs (Figure 7a and prediction error rate for DNN models with different architectures (Figure 7b). We draw two conclusions from this experiment: 1) DeepSniffer can achieve much better prediction accuracy with considering the layer context information. The results show that the average LER of the context-aware identifier is three times lower than the single-layer identifier (Figure 7a). 2) Layer context information is increasingly important when identifying more complex network architectures. As shown in Figure 7b, compared to the simple network architecture with only chain typologies, the more complex architectures with remote connections (e.g. Add or Concat) cause higher error rates. For the single-layer identifier, the LER dramatically increases when the network is more complex (from 0.18 to 0.5); while, for the context-aware identifier, the average LER demonstrates a non-significant increase (from 0.065 to 0.104). The experimental results indicate that the layer context with inter-layer temporal association is a very important information source, especially for the deeper and more complex neural networks.

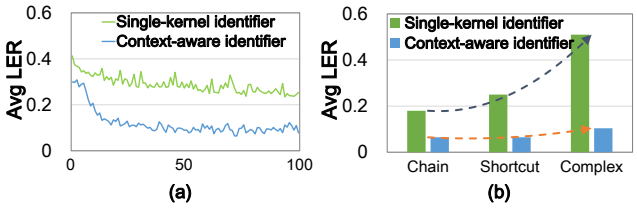


Fig. 7. (a) Average prediction error rate comparison between single-kernel identifier and context-aware identifier during training process. (b) Average prediction rate comparison with different victim DNNs

5.3 Model Size Estimation

In this section, we show the feature map size estimation results of the input and output for every layer, which is the prerequisite for dimension space estimation. For both the side-channel and bus snooping attacks, the input and output feature map sizes of every layer in the DNN model are calculated based on the ReLU memory traffic volume. Therefore, we show estimation accuracy under bus snooping scenario as an example in Figure 8, which is calculated as 1 minus the deviation between the estimated size and actual size. For Conv, BN, ReLU, Add, and Concat, the estimation accuracy can reach up to 98%. The FC presents lower accuracy since the FC layer is usually at the end of the network and the neuron number decreases. Thus, the activation data of the ReLU layer may be filtered, and it is not accurate to use ReLU read transactions to estimate the FC size. We use the read access volume to predict the input and output sizes of FC layers instead. The dimension size prediction results may be platform-dependent. However, we take the dimension size prediction as the less important step than the other two and experimentally validate the effectiveness of the extracted architectures with imprecise dimension sizes.

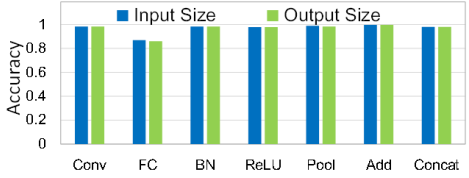


Fig. 8. Layer input and output feature map size estimation (normalized to the ground-truth size).

6 How Valuable are Extracted Models?

The extracted network architectures not only endanger the intelligence property. Even worse, they can be adopted to perform advanced attacks in the further step. In this study, we test one of the most important attack model, explorational attacks, as the end-to-end attack case to show the effectiveness of the extracted neural network architectures.

6.1 Advanced Attacks with Extracted DNN Archs

Explorational attack crafts the inputs with malicious perturbations to mislead the victim model to produce an arbitrary (*untargeted attack*) [18] or a pre-assigned incorrect output (*targeted attack*) [36], [37], [38]. It does not poison the victim model and is one of the most important attack model during the inference stage [8]. According to different restriction of input perturbations, it can be classified into two major categories: adversarial example attack and adversarial patch attack.

Adversarial example attacks add input-dependent, but imperceptible noises in the input images to manipulate the outputs [18]. Adversarial example attack have been studied broadly and being considered as one of the most important attack models to explore the DNN vulnerability.

Adversarial patch attacks add input-independent patch perturbation in the input images to manipulate the victim model to output malicious results [39]. Compared to adversarial example attacks, patch attacks are more practical due to the reasons: 1) Better universality across different input images. Such a scene-independent feature enables practical physical-world attack without prior knowledge of the scene. 2) Better robustness to environmental noises and geometric distortion. It is not only effective in a static figure input, but also in complex scenarios like walking, sitting, and running.

Attack flow: For both of these two attacks, to perform the adversarial attack against a black-box model, the adversary normally builds a substitute model first by querying the input and output of the victim model. Then the adversary generates the adversarial examples based on the white-box substitute model [36], [40], [41]. Finally, they use these adversarial examples to attack the black-box model.

In summary, the transfer-based adversarial attack flow is illustrated in Figure 9, which consists of the following steps:

1): Build substitute models. Baseline selects the typical network architectures from model zoo to build the substitute model, while DeepSniffer trains substitute models with the extracted network architectures, as shown in Figure 9 .

2): Adversarial Attacks. Both adversarial example attacks and adversarial patch attacks have the same attack flow: first, generate the adversarial examples or adversarial patches on the subsitute models. Then, apply the adversarial inputs to the black-box victim model.

In this section, we show that the adversarial attack efficiency can be significantly improved with the extracted network architectures.

6.2 Attack Setup

Configurations: In these experiments, we use ResNet18 [5] as the victim model for targeted attacks. Our work adopts the extracted neural network architecture to build the substitute models. For comparison, the baseline examines the substitute models established from the following networks: VGG family [2] (`VGG11`, `VGG13`, `VGG16`, `VGG19`), $ResNet$ family [5] (`ResNet34`, `ResNet50`, `ResNet101`, `ResNet152`), $DenseNet$ family [13] (`DenseNet121`, `DenseNet161`, `DenseNet169`, `DenseNet201`), `SqueezeNet` [42], and `Inception` [27].

Extracted DNN Architectures: Based on the architectural hints of ResNet18, we extract DNN architectures following the three steps: run-time sequence identification, layer topology reconstruction, and dimension estimation, as shown in the Figure 10. In the run-time layer sequence identification, DeepSniffer accurately predicts the layer sequence with small errors in red color. In dimension size estimation, we randomly select four dimension sets from the potential dimension space which satisfy the layer size and constraints. The four dimension sets are different from the original victim ResNet18. Therefore, we validate the effectiveness of these extracted neural network models that are slightly different from the original victim model in the following.

6.3 Adversarial Example Attack Effectiveness

First, we randomly select 10 classes, each class with 100 images from ImageNet dataset [43] as the original inputs for targeted attack tests. Then, we compare the attack effectiveness of adversarial examples generated by the following five solutions: ensembled substitute models from VGG family, $DenseNet$ family, Mix architectures (squeezeNet, inception, AlexNet, DenseNet), $ResNet$ family, and from extracted architectures using our proposed model extraction.

The attack success rate results are shown in Table 4. We report several observations: 1) The attack success rate is generally low for the cases without network architecture knowledge. The adversarial examples generated by substitute models with VGG family, $DenseNet$ family, and Mix architectures only complete successful attacks in 14%–25.5% of the cases. 2) With some knowledge of the victim architecture, the attack success rate is significantly improved. For example, the substitute models within $ResNet$ family achieve the attack success rate of 43%. 3) With our extracted network architectures – although it still has differences from the original network – the attack success rate is boosted to 75.9%. These results indicate that our model extraction significantly improves the success rate of consequent adversarial attacks.

In a further step, we take a deep look at the targeted attack which leads the images in Class-755 to be misclassified as Class-255 in the ImageNet dataset. We explore the effectiveness of ensembled models with various substitute combinations, by randomly picking four substitute models from the candidate model zoo. The results are shown in the blue bars of Figure 11. We also compare the results to the cases using substitute models 1) from VGG family; 2) from $DenseNet$ family; 3) from squeezeNet, inception, AlexNet, and DenseNet ('Mix' bar in the figure); 4) from $ResNet$ family; and 5) from extracted cognate ResNet18 model (our method) to generate the adversarial examples. As shown in Figure 11, the average success rate of random cases is only 17% and the best random-picking case just

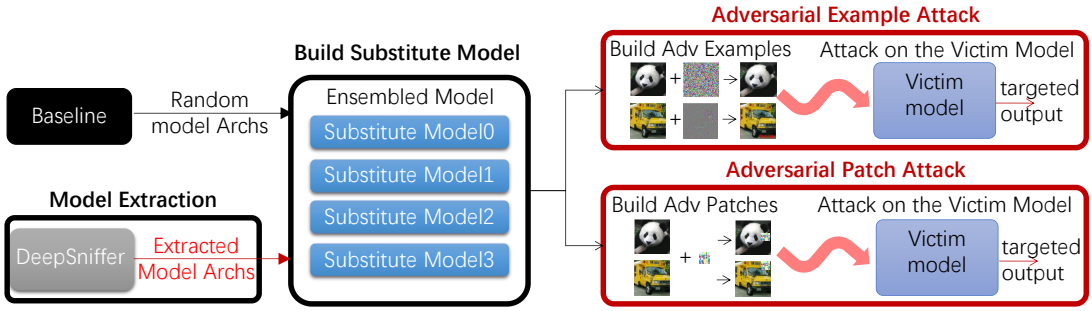


Fig. 9. Flows of adversarial example attack and patch attack.

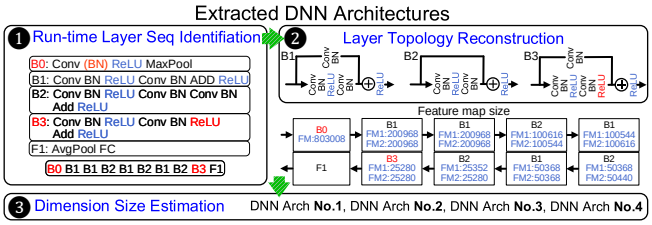


Fig. 10. Extracted DNN architectures.

achieves the attack success rate of 34%. We observe that all good cases in random-picking (attack success rate ¿ 20%) include substitute models from ResNet family. Our method with accurate extracted DNN models performs the best attack success rate across all the cases, 40% larger than the best random-picking case and $ResNet$ family cases. To summarize, with the help of the effective and accurate model extraction, the consequent adversarial attack achieves a much better attack success rate. Therefore, it is extremely important to protect the neural network architectures in the DNN system stack, which can boost the adversarial attack effectiveness.

TABLE 4
Success rate with different substitute models.

	VGG family	DenseNet family	Mix	ResNet family	Extracted DNN
Success rate	18.1%	25.5%	14.6%	43%	75.9%

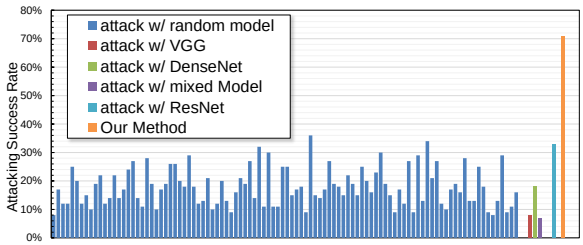


Fig. 11. Explore the targeted attack success rate across different cases. Our method performs best.

6.4 Adversarial Patch Attack Effectiveness

We additionally test the effectiveness of extracted network architectures on adversarial patch attacks. Patch attack is to generate a uniform adversarial patch for any arbitrary inputs that manipulate the victim model to output targeted or untargeted labels [39]. The major difference between an adversarial example and patch attack is that the adversarial patches are not dependent on the inputs while every different input desires a customized adversarial example. Adversarial patch attacks eliminate the necessity to obtain the attack scene in advance, which is more robust and feasible in physical world.

We randomly select 10 classes, each class with 100 images from ImageNet dataset for adversarial patch attack. We examine the transferability across both different models and input images. Specifically, the victim model is ResNet18. We first generate adversarial patches on the training dataset for VGG family, DenseNet family, Mix architectures, ResNet family, and our extracted model. Then, we test the model transferability of these patches by applying adversarial patches on the training set and take the patched images to the victim model. The attack success rate results are shown in the category of 'TrainSet' in Figure12. We also test data transferability by applying patches on testing dataset and take the patched images into the victim model. The attack success rate results are shown in the category of 'TestSet' in Figure12.

Consistently, the attack success rate for adversarial patch attack is generally low for the cases without network architecture knowledge. Under the training set, the adversarial patch generated by substitute models with VGG family, DenseNet family, and Mix architectures complete success attacks in 41% - 44% of the cases. With the knowledge of the victim architecture, the attack success rate is improved largely. For example, the substitute models within ResNet family achieve the attack success rate of 62%. With the extracted model architectures by DeepSniffer, the attack success rate is 83%. In addition to the model transferability, we also test the transferability of the adversarial patch on the test input set. In the test set, the adversarial patch generated by the DeepSniffer can still achieve 82% of attack success rate. Such experiment results not only reveal the importance of model knowledge protection, but also demonstrate the effectiveness of the model extraction technique in this work.

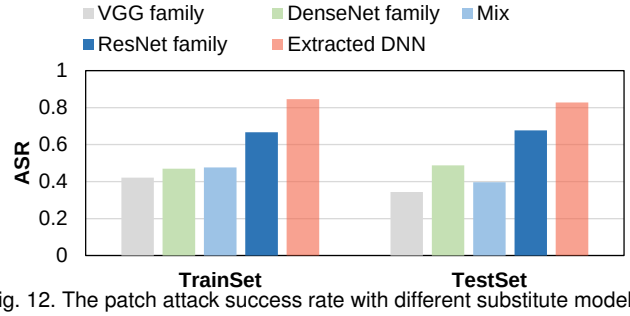


Fig. 12. The patch attack success rate with different substitute models.

7 Analysis of Defensive Approaches

7.1 Microarchitecture Methodologies

There are a few architectural memory protection methods. Oblivious Memory: To reduce the information leakage on the bus, previous work proposes oblivious RAM (ORAM) [44] which hides the memory access pattern by encrypting the data addresses. With ORAM, attackers cannot identify two operations even when they are accessing the same physical address [44]. However, ORAM techniques incur a significant memory bandwidth overhead (up to an astonishing 10x), which is impractical for bandwidth-sensitive GPUs. Dummy Read/Write Operations: Another potential defense solution is to introduce fake memory traffic to disturb the statistics of memory events. Unfortunately, the noises exert only a small degradation of the layer sequence prediction accuracy. As such, fake RAW operations to obfuscate the layer dependencies identification may be a more fruitful defensive technique to explore.

Robustness to Hint Noises We conduct experiments to analyze the accuracy sensitivity of the identifier taking in the kernel features with noises. Taking the bus snooping attack as an example, When kernel execution feature statistics are affected by random noises within 5%, 10%, 20%, or 30% of amplitude, the average error rate of the layer prediction increases from 0.08 to 0.16. The results indicate that the layer sequence identifier is not sensitive to architectural hint noises.

7.2 System Methodologies

The essence of our work is to learn the compilation and scheduling graphs of the system stack. Although the computational graphs go through multiple levels of the system stack, we demonstrate that it is still possible to recover the original computational graph based on the raw information stolen from the hardware. At the system level, one could: 1) customize the overall NN system stack with TVM, which is able to implement the graph level optimization for the operations and the data layout [45]. The internal optimization possibly increases the difficulty for the attackers to learn the scheduling and compilation graph, or 2) make security-oriented scheduling between different batches during the front-end graph optimization. Although such optimizations may have a negative impact on performance, they may obfuscate the adversary a view of kernel information.

8 Related Work

The security problem for the DNN system has emerged as an urgent and severe problem, since DNN techniques have infiltrated into many security-critical applications. The related research mainly comes from the algorithm and hardware aspects.

Algorithm researchers originally observe that model characteristics leakage may heavily reveal the vulnerability of DNN models to attackers. By extracting such information, attackers not only counterfeit the intellectual property of the DNN design, but also perform more efficient adversarial attacks that manipulate the DNN model to output malicious outputs [9], [10]. Due to the significance of model characteristics, model extraction attack that steals the model characteristics of victim models, consequently, becomes an important attack model for exploring the DNN model vulnerability [10], [11], [12]. The network architecture extraction is the most fundamental step for stealing other model characteristics, such as the parameter, and hyperparameter, or even training data [11], [12]. It is also challenging that relies on the meta-learning and network structure searching, which introduces significant computing overhead [10].

Exploiting the exposed information in the hardware attack surfaces can strengthen the model extraction effectiveness. Several studies are proposed, either aiming to conduct model extraction [17] or input inversion based on memory access pattern and power traces. However, their methodologies rely on the specific design features in hardware platforms and cannot be generally applicable to GPU platforms with the full system stack. CathyTelepathy [16] explores side-channel techniques in caches to reduce the hyper-parameter space of victim DNN models by inferring the configurations of GEMM operations. *Naghibijouybari et al.* show that side-channel effect in GPU platform can reveal the neuron numbers [15]. However, none of these studies show that how these obtained statistics are useful to the attacking effectiveness. This work is the FIRST to propose the DNN model extraction framework and experimentally conduct an end-to-end attack on an off-the-shelf GPU platform immune to full system stack noises. Inspired by DeepSniffer, more studies leverage learning-based methodology to perform model extraction [46]. The formalized schema in this work can be extended to comprehensive passive attacks that explore what DNN system attack surfaces matter and how bad are the architectural risk

9 Conclusion

The widespread use of neural network-based applications raises stronger and stronger incentive for attackers to extract the neural network architectures of DNN models. In observing the limitations of previous work, we propose a robust learning-based methodology to extract the DNN architecture. Through the acquisition of memory access events from bus snooping, layer sequence identification by the LSTM-CTC model, layer topology connection according to the memory access pattern, and layer dimension estimation under data volume constraints, we demonstrate one can accurately recover a similar network architecture as the attack starting point. These reconstructed neural network architectures present significant increase in attack success rates, which demonstrate the importance of establishing secure DNN system stack.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," CoRR, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[3] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig, "The microsoft 2016 conversational speech recognition system," in Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on. IEEE, 2017, pp. 5255–5259.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in Advances in Neural Information Processing Systems, 2017, pp. 6000–6010.

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

[6] G. M. Consulting, "Autonomous vehicle adoption study," 2016.

[7] TechCrunch, "Nvidia is powering the world's first level 3 self-driving production car." 2017.

[8] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," CoRR, vol. abs/1801.00553, 2018.

[9] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," ICLR, vol. abs/1611.02770, 2017.

[10] S. J. Oh, M. Augustin, B. Schiele, and M. Fritz, "Towards reverse-engineering black-box neural networks," ICLR, vol. abs/1605.07277, 2018. [Online]. Available: https://arxiv.org/abs/1711.01768

[11] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in Proceedings of the 25th USENIX Conference on Security Symposium, ser. SEC'16. Berkeley, CA, USA: USENIX Association, 2016, pp. 601–618.

[12] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning," CoRR, vol. abs/1802.05351, 2018.

[13] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in CVPR 2017.

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

[15] H. Naghibijouybari, A. Neupane, Z. Qian, and N. Abu-Ghazaleh, "Rendered insecure: Gpu side channel attacks are practical," in Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '18. New York, NY, USA: ACM, 2018, pp. 2139–2153.

[16] M. Yan, C. W. Fletcher, and J. Torrellas, "Cache telepathy: Leveraging shared resource attacks to learn DNN architectures," CoRR, vol. abs/1808.04761, 2018.

[17] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," in Proceedings of the 55th Annual Design Automation Conference, ser. DAC '18. New York, NY, USA: ACM, 2018, pp. 4:1–4:6.

[18] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," Proceedings of the International Conference on Learning Representations, 2015.

[19] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," CoRR, vol. abs/1512.00567, 2015.

[20] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," arXiv preprint arXiv:1611.03814, 2016.

[21] Q. Xiao, K. Li, D. Zhang, and W. Xu, "Security risks in deep learning implementations," in 2018 IEEE Security and Privacy Workshops (SPW). IEEE, 2018, pp. 123–128.

[22] Waymo, "Introducing waymo's suite of custom-build, self-driving hardware," 2017. [Online]. Available: https://medium.com/waymo/introducing-waymos-suite-of-custom-built-self-driving-hardware-c47d1714563/

[23] M. Rhu, M. O'Connor, N. Chatterjee, J. Pool, Y. Kwon, and S. W. Keckler, "Compressing dma engine: Leveraging activation sparsity for training deep neural networks," in 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), Feb 2018, pp. 78–91.

[24] "Hmtt: Hybrid memory trace toolkit," 2019. [Online]. Available: http://asg.ict.ac.cn/hmtt/

[25] Nvidia., "Nvidia cudnn gpu accelerated deep learning," 2017. [Online]. Available: https://developer.nvidia.com/cudnn

[26] PyTorch., "Pytorch tutorials." [Online]. Available: http://pytorch.org/tutorials/

[27] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning." in AAAI, 2017, pp. 4278–4284.

[28] X. Zhang, Z. Li, C. C. Loy, and D. Lin, "Polynet: A pursuit of structural diversity in very deep networks," CoRR, vol. abs/1611.05725, 2016.

[29] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," CoRR, vol. abs/1707.07012, 2017.

[30] J.-H. Tao, Z.-D. Du, Q. Guo, H.-Y. Lan, L. Zhang, S.-Y. Zhou, C. Liu, H.-F. Liu, S. Tang, and A. Rush, "Benchip: Benchmarking intelligence processors," arXiv preprint arXiv:1710.08315, 2017.

[31] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32, ser. ICML'14, 2014, pp. II–1764–1772.

[32] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in Proceedings of the 23rd International Conference on Machine Learning, ser. ICML '06. New York, NY, USA: ACM, 2006, pp. 369–376.

[33] NVIDIA, "Nvidia tesla k40 active gpu accelerator," http://www.pny.com/nvidia-tesla-k40-active-gpu-accelerator, 2016.

[34] N. Wilt, The cuda handbook: A comprehensive guide to gpu programming. Pearson Education, 2013.

[35] Nvidia., "Cuda toolkit documentation." [Online]. Available: http://docs.nvidia.com/cuda/profiler-users-guide/index.html

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

[37] S. Alfeld, X. Zhu, and P. Barford, "Data poisoning attacks against autoregressive models," in Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, ser. AAAI'16. AAAI Press, 2016, pp. 1452–1458.

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

[39] K. Xu, G. Zhang, S. Liu, Q. Fan, M. Sun, H. Chen, P.-Y. Chen, Y. Wang, and X. Lin, "Adversarial t-shirt! evading person detectors in a physical world," in ECCV, 2020, pp. 665–681.

[40] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, ser. ASIA CCS '17. New York, NY, USA: ACM, 2017, pp. 506–519.

[41] N. Papernot, P. D. McDaniel, and I. J. Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," CoRR, vol. abs/1605.07277, 2016. [Online]. Available: http://arxiv.org/abs/1605.07277

[42] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," arXiv preprint arXiv:1602.07360, 2016.

[43] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2009. IEEE, 2009, pp. 248–255.

[44] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path oram: An extremely simple oblivious ram protocol," in Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security, ser. CCS '13. New York, NY, USA: ACM, 2013, pp. 299–310.

[45] T. Chen, T. Moreau, Z. Jiang, H. Shen, E. Q. Yan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: end-to-end optimization stack for deep learning," CoRR, vol. abs/1802.04799, 2018.

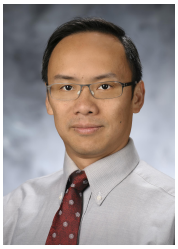
[46] S. Banerjee, P. Ramrakhyani, S. Wei, and M. Tiwari, "Sesame: Software defined enclaves to secure inference accelerators with multi-tenant execution," arXiv preprint arXiv:2007.06751, 2020.

Xing Hu received the B.S. degree from Huazhong University of Science and Technology, Wuhan, China, and Ph.D. degree from University of Chinese Academy of Sciences, Beijing, China, in 2009 and 2014, respectively. She is currently an associate professor of State Key Laboratory of Computer Architecture, Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, China. Her current research interests include domain-specific hardware architectures and deep learning system security.

Ling Liang received the B.E. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2015, and M.S. degree from University of Southern California, CA, USA, in 2017. He is currently pursuing the Ph.D. degree at Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA. His current research interests include machine learning security and computer architecture.

Lei Deng received the B.E. degree from University of Science and Technology of China (USTC), Hefei, China in 2012, and the Ph.D. degree from Tsinghua University (THU), Beijing, China in 2017. He is currently a Postdoctoral Fellow at the Department of Electrical and Computer Engineering, University of California, Santa Barbara (UCSB), CA, USA. His research interests span the area of brain-inspired computing, machine learning, neuromorphic chip, computer architecture, tensor analysis, and complex networks.

Timothy Sherwood is a Professor of Computer Science at the University of California, Santa Barbara. He is a co-founder of the hardware security startup Tortuga Logic and the 2016 ACM SIGARCH Maurice Wilkes Awardee “for contributions to novel program analysis advancing architectural modeling and security.” Professor Sherwood received a B.S. degree in computer science from UC Davis, and a M.S. and Ph.D. degrees from UC San Diego.



Yuan Xie received his Ph.D. degrees from Electrical Engineering Department, Princeton University, Princeton, NJ, USA in 2002. He was with IBM, Armonk, NY, USA, from 2002 to 2003, and AMD Research China Lab, Beijing, China, from 2012 to 2013. He was a Professor with Pennsylvania State University, State College, PA, USA, from 2003 to 2014. He is currently a Professor with the Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA, USA.

Dr. Xie is an expert in computer architecture who has been inducted to ISCA/MICRO/HPCA Hall of Fame. He has been an IEEE Fellow since 2015. He served as the TPC Chair for HPCA 2018 and he is Editor-in-Chief for ACM Journal on Emerging Technologies in Computing Systems (JETC), Senior Associate Editor (SAE) for ACM Transactions on Design Automations for Electronics Systems (TODAES), and Associate Editor for IEEE Transactions on Computers (TC). His current research interests include computer architecture, Electronic Design Automation, and VLSI design.