Tenet: A Neural Network Model Extraction Attack in Multi-core Architecture

Chengsi Gao^{1,4}, Bing Li², Ying Wang^{*1,5}, Weiwei Chen^{1,4}, Lei Zhang^{1,3}

¹Institute of Computing Technology, Chinese Academy of Sciences

²Capital Normal University ³Jeejio(Ningbo) Technology ⁴University of Chinese Academy of Sciences

⁵SKLCA, Institute of Computing Technology, Chinese Academy of Sciences

gaochengsi17z@ict.ac.cn

ABSTRACT

As neural networks (NNs) are being widely deployed in many cloud-oriented systems for safety-critical tasks, the privacy and security of NNs become significant concerns to users in the cloud platform that shares the computation infrastructure such as memory resource. In this work, we observed that the memory timing channel in the shared memory of cloud multi-core architecture poses the risk of network model information leakage. Based on the observation, we propose a learning-based method to steal the model architecture of the NNs by exploiting the memory timing channel without any high-level privilege or physical access. We first trained an end-to-end measurement network offline to learn the relation between memory timing information and NNs model architecture. Then, we performed an online attack and reconstructed the target model using the prediction from the measurement network. We evaluated the proposed attack method on a multi-core architecture simulator. The experimental results show that our learning-based attack method can reconstruct the target model with high accuracy and improve the adversarial attack success rate by 42.4%.

CCS CONCEPTS

• **Security and privacy** → *Side-channel analysis and countermeasures*; • **Computing methodologies** → *Neural networks*; • **Computer systems organization** → *Multicore architectures*.

KEYWORDS

Deep learning security, Machine learning, Multi-core, Memory timing channel

ACM Reference Format:

Chengsi Gao, Bing Li, Ying Wang, Weiwei Chen, and Lei Zhang. 2021. Tenet: A Neural Network Model Extraction Attack in Multi-core Architecture. In *Proceedings of the Great Lakes Symposium on VLSI 2021 (GLSVLSI '21)*, June 22–25, 2021, Virtual Event, USA. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3453688.3461512>

*Corresponding Author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://permissions.acm.org).
GLSVLSI '21, June 22–25, 2021, Virtual Event, USA

© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8393-6/21/06...\$15.00
<https://doi.org/10.1145/3453688.3461512>

1 INTRODUCTION

The rapid progress of deep learning on computing vision and language process have motivated the deployment of neural networks (NNs) in many security-critical missions, such as autonomous driving [19] and malware detection [12]. Therefore, the security issue of NN has become an important problem and attracted increasing attention from industry and research communities.

Recent works have revealed the significance of NN model information in the NN security [7, 14]. With the model information, attackers can infer the input and training data, fake the Intellectual Property (IP) of model owners, and launch enhanced adversarial attacks which are serious threats to NNs [16] and will cause fatal prediction errors in critical missions [7, 15, 21]. Therefore, the model extraction attack receives increasing attentions in the NN community. More seriously, as the application of the public cloud platform, the new environment provides many opportunities for attackers to conduct the model extraction attack. When millions of clients simultaneously run on the same servers and lease NN models to conduct inference/prediction tasks, the cloud providers are difficult to recognize the malicious users who intend to carry out the model extraction attack.

Due to the importance of the NN models, many prior attack methods are developed by exploiting the weaknesses in NNs for model extraction attacks from both software and hardware perspectives [3, 7, 8]. However, developing model extraction attack on public cloud platform faces some new challenges. 1) Attackers have no privilege to access the hardware physical infrastructure; 2) Cloud platforms usually employ the multi-core hardware to serve NN-based tasks and possibly adopt emerging NN accelerators [4, 11, 24], which has different architecture from previous works.

To tackle above issues, we analyse the memory access behaviors in the cloud-oriented multi-core architecture and observe that the shared memory bus can be exploited to steal the confidential NN model information in the running tasks. Based on this observation, we design a learning-based method, *Tenet*, to perform powerful model extraction attack against the target victim model.

The main contributions of this paper are as follows:

- We first demonstrate that the memory timing information in the shared memory channel exposes the knowledge of model architecture, which can be exploited by the malicious tenants to reverse-engineer the layer structure of neural networks;
- We propose a learning-based model extraction attack method, *Tenet*, which exploits the memory timing information of the multi-core architecture to steal the accurate model architecture of the victim neural networks;

Figure 1: Multi-core architecture with neural network accelerators. MC: Memory Controller; R: Router; A and B are two different cores that are allocated to two concurrent tenants which are the attacker and the target, respectively.

- We evaluated the proposed *Tenet* in a cycle-accurate multi-core architecture simulator. The experimental results show that the extracted network models are similar to the target models and the adversarial attack success rate is significantly improved when attackers having the extracted models.

2 BACKGROUND & RELATED WORK

2.1 Model Extraction Attack

Model extraction attack targets to recover a similar or even same architecture model to the victim target model and is becoming an appealing attack method. Though the effective information in NN also includes 1) Model parameters: the weights, bias, etc.; 2) Training data: the private data in the training set used for model training, prior works demonstrate that the model parameters and even training data can be inferred from the information of model architecture [20, 21]. Thus, we focus on developing the attack method that extracts the model architecture. Besides, model extraction attack plays an important role for further attacks. With a reconstructed network architecture similar to the target model, the adversarial attack success rate can be effectively improved [7, 15, 17].

To extract the NN model architecture, some prior works have considered various attacks [3, 7, 8, 23]. Unfortunately, they can not be directly applied in a public cloud platform that employs the multi-core architecture. For example, the works [3, 7, 8] require physical access to the hardware or user-injected components in the system to conduct external side-channel measurements, such as electromagnetic (EM), bus snooping, or hardware Trojan. However, the adversary in the public cloud platform is a common user who has no privilege to contact the physical hardware infrastructure. The work [23] tries to recover a DNN's architecture using the shared cache side channel and it assumes that all the Conv and Fc layers are implemented by blocked GEMM. However, for cloud platforms integrated with NN accelerators [4, 5, 11], such an assumption is not practical because the NN accelerators usually implement the Conv operations in their individual dataflow.

2.2 Memory Timing Channel

The memory timing channel, existing in the multi-core architecture, may leak information through memory access interference when shared [9, 22]. Previous works have exploited the memory timing channel to carry out attacks. The work [22] presents a side-channel attack against the RSA decryption algorithm by intentionally creating the memory access contention on the shared memory controller.

Figure 2: Each layer completion time of N_{prob}

Researchers also present that attackers can frequently steal abundant information through memory timing channels [9].

We notice that the hardware architecture of the cloud platform essentially is the multi-core architecture, where multiple cores share one or more memory controller(s) and execute the client NN tasks separately at the same time. The adversary can run its own task to attack when different clients use the NN model to run inference tasks. Thus, we explore the memory timing channel to probe the NN architecture information leakage and then develop a learning-based model extraction attack method against NN models for the multi-core architecture. To the best of our knowledge, this is the first work to discuss *model extraction attack for the multi-core architecture*.

3 MOTIVATION STUDY

In this section, we investigate the memory timing channel in multi-core architecture on which multiple network inference tasks simultaneously run. Meanwhile, we present the model information leakage through the memory timing channel.

Figure 1 shows an example of the multi-core architecture integrated with NN accelerators in the public cloud platform [1, 4]. Two neural networks are running on Core A and Core B, respectively, and access global memory through the shared memory channel. To investigate the model information leakage through the memory timing channel, we repetitively measured the inference latency of one network (N_{prob}) while varying the dimension and layer type of the other network (N_{target}) running on the other core. To catch the sufficient amount of memory interference behaviors, N_{prob} is designed to be a memory-intensive network (the detailed structure is given in Section 5.1.1). The layer completion time of N_{prob} with different settings is shown in Figure 2. The analysis on the results shows that a neural network's information can be easily inferred by recording the memory latency of the neural network running on the other core.

In Figure 2(a), N_{prob} is a five-layer network, and N_{target} has one convolutional layer, of which the convolutional kernel is changed in its geometry to interfere with N_{prob} in different degrees. For N_{target} , C64X3X3 represents a convolution filter whose channels, filter height, and width are 64, 3, and 3, separately. In Figure 2(b), the architecture of N_{target} varies to contain different types of layers, including a convolutional layer (C), a stack of convolutional layer and average pooling layer (AP), an element-wise addition layer (referred as Add for simplicity), and a stack of Add layer and AP layer. For illustration, AP2x2 represents an average pooling layer whose kernel size and stride are 2 and 2, separately. Here, each layer's completion time in N_{prob} is normalized to the layer completion

time of N_{prob} when running independently. Each layer completion time of N_{prob} is denoted as a sequence T , $T = \{t_0, t_1, t_2, t_3, t_4\}$.

As can be seen from Figure 2, the time sequences T of N_{prob} , i.e., the color lines in Figure 2 are different along with various dimensions (Figure 2(a)) and types (Figure 2(b)) of N_{target} 's layer. Because of the memory interference between N_{prob} and N_{target} , the completion time of N_{prob} is affected by the memory access behavior of N_{target} . Since the dimension and type of a neural network layer determine its intensity of the memory access, varying the architecture of N_{target} will cause different layer completion time of N_{prob} , which is shown as the different distribution of the time sequence T in Figure 2.

Once attackers know the quantitative relationship between the completion time sequence T of N_{prob} and the architecture of N_{target} , they can infer the layer type and dimension of N_{target} by measuring the completion time of N_{prob} . However, the variety of neural networks running on the cloud platform makes it impossible to manually build the correlation function between N_{prob} 's completion time and the architecture of N_{target} . Thus, we propose a learning-based model extraction attack, which can automatically learn the relationship and steal the model architecture information.

4 THREAT MODEL

In this paper, we mainly focus on the cloud-oriented multi-core architecture integrated with NN accelerators. The threat model is as shown in Figure 1. The attackers rely on the following assumptions.

The attackers can measure the layer completion time via a software timer module. This is feasible as recent cloud-oriented platforms like cloud TPU [1], allow users to upload their individual code. In this case, attackers can embed an accurate timer in their code segments to measure completion time.

Second, we assume the hardware platform used during the training and attack phase is the same. Because our attack method is based on supervised learning, so attackers must ensure their training and inference data conform to the identical distribution. The same hardware platform can be built off-line by the attackers according to the target platform's specifications; besides, attackers can directly apply for resources in target platform to obtain the training set since the target platform is cloud-shared.

5 IMPLEMENTATION OF ATTACKS

In this paper, we propose an attack method to extract the model architecture of the target network, N_{target} . This attack method consists of three parts: a designed probing network that is used to capture the memory interference from N_{target} (① in Figure 3); a trained measurement network that is used to estimate the layer structure of N_{target} (② in Figure 3); and a network topology reconstruction algorithm that reconstructs the network architecture with the layer information (③ in Figure 3). The entire attack flow has two phases:

(1) **Offline Training:** This phase is to learn the relationship between the layer completion time of N_{prob} and target model architecture information. First, we construct the probing network N_{prob} . Then we prepared massive sample networks with different architectures and collected the layer completion time of N_{prob} when running N_{prob} together with various sample networks on

Table 1: Collected data for the training of measurement network

Name	Description	Type
T	Time sequence of N_{prob}	Input
S	Layer structure sequence of sample networks	Label

Figure 3: Online model extraction attack flow

the multi-core hardware (denoted as the sequence T in Table 1) and layers' structure of the sample networks (denoted as the layer structure sequence S in Table 1). With the collected data, we design and train a measurement neural network to learn the correlation between the sequence T and the layer structure of neural networks.

(2) **Online Attack:** An illustration of the online attack flow is shown in Figure 3. To launch attacks, the attackers deploy the N_{prob} on one core of the target platform that the N_{target} runs on. N_{prob} will run concurrently with N_{target} , and each layer's completion time of N_{prob} will be collected as a sequence T , $T = \{t_0, t_1, \dots, t_{n-1}\}$. Then, the attacker feeds T into the measurement network to get the predicted layer structure sequence S , $S = \{s_0, s_1, \dots, s_{m-1}\}$. Each element of S represents one tuple containing the type and dimension of one layer in N_{target} . In the end, attackers will reconstruct the architecture of N_{target} by searching through S .

5.1 Offline Training Phase

5.1.1 **Probing Network Design.** N_{prob} aims to capture the memory interference from N_{target} . Hence, each layer of N_{prob} shall be memory intensive. When we use the runtime to describe memory intensity, the proportion of the computing time to the total runtime, which is denoted as P_c , shall be small for each layer of N_{prob} . For the convolutional layer, P_c can be formulated as follows:

$$P_c \propto \alpha * Size_{OFM}^2 * Size_{Ker}^2 \propto \alpha * \left(\frac{Size_{IFM} - Size_{Ker}}{Stride_{Ker}} \right)^2 * Size_{Ker}^2, \quad (1)$$

Where the α is a constant factor decided by the hardware, the $Size_{OFM}$, $Size_{Ker}$, $Size_{IFM}$, and $Stride_{Ker}$ are the size of output, kernel, input, and the stride of the kernel, respectively. Thus, a convolutional layer with a large stride and small kernel size will

have a small P_c , which indicates a memory-intensive layer. When kernel size is set a small value, i.e., 1; in contrast, the kernel stride will be set large, like the size of the input feature map. Then the Equation 1 will be converted into Equation 2.

$$P_c \propto \alpha * \left(1 - \frac{1}{Size_{IFM}}\right)^2 \quad (2)$$

According to Equation 2, to decrease P_c , we should increase the $Size_{IFM}$. Regretfully, too large $Size_{IFM}$ leads to a long loading time of input. Consequently, during the computing of one layer of N_{prob} , one layer of N_{prob} may be interfered with multiple layers from N_{target} , increasing the difficulty of training measurement network and decreasing the prediction accuracy for some small N_{target} .

In this work, we did trials with different $Size_{IFM}$ and found that the $P_c = 2\%$ is the best and used this to design N_{prob} . Besides, the length of N_{prob} , T , should be large so that N_{prob} can capture the memory interference from all layers for deep N_{target} . In this work, we set the layer number to 1000 which can satisfy most types of network models and also ensure the learning efficiency of the measurement network.

5.1.2 Sample Networks Preparation. To allow the measurement network to predict the desired network information, we prepared sufficient training data for it to learn. These data come from the sample networks that are generated through an automatic network generator.

We adopt a random generator to iteratively generate the topology and the layer description for the desired sample networks. The generator will pick one type of block at each iteration. The block has two types: the sequential or the residual. The sequential block denotes a single layer or a stack of non-residual layers, like Conv, FC, Conv-ReLu, or Conv-ReLu-Pooling layers, etc. The residual block is similar to [6]. After the block type is decided, the generator will determine the number of layers in each branch of the residual block and the layer's dimension in the selected block, like the kernel size and kernel stride for Conv and Pooling. In the generation phase, there can be some constraints on the input/output dimension and other design parameters, such as the maximum depth of the network and the maximum size of the feature map.

With the generated sample networks, we run N_{prob} and each sample network concurrently on the same hardware platform to collect the training data (shown in Table 1).

5.1.3 Measurement Network Design. The measurement network learns the correlation between the layer completion time of N_{prob} and the layer structure sequence S of the co-running network. Then in the online attack phase, it will be used to predict the layer sequence S of the N_{target} .

Similar to the work [7], our sequence S prediction problem can be viewed as a speech recognition problem as they both maximize the probability of a sequence under certain rules with the input context. For example, in our sequence S prediction problem, there are some rules in the network architecture design, 1) the height/width of the weight matrix is more likely to be an odd number; 2) a pooling layer is more likely to be followed by a convolutional layer rather than a pooling layer. Therefore, we can leverage the techniques in auto speech recognition (ASR) for the sequence S prediction. In

Algorithm 1: Target Architecture Reconstruction

Input: Layer descriptive sequence S .
Output: Reconstructed target architecture sets:
 $ArchSets = \{Arch_1, Arch_2, ..., Arch_k\}$.

```

1  $ArchSets = \{\emptyset\}$ ;  $PreAdd = 0$ ;
2  $S' = ConvertToChain(S)$ ;
3 for  $l$  in  $0 : length(S')$  do
4    $L = S'[l]$ ;
5   if  $type(L) == Add$  then
6      $endpt_1 = l - 1$ ;
7     for  $startpt$  in  $PreAdd : l - 1$  do
8       for  $endpt_2$  in  $startpt : l - 1$  do
9          $res\_block, esize_1, esize_2 =$ 
           $ConstResbk(S', startpt, endpt_1, endpt_2)$ ; if
10           $esize_1 == esize_2$  then
11             $Arch = ReconsArch(ArchSets, res\_block)$ ;
12             $ArchSets += Arch$ ;
13          end
14        end
15      end
16     $PreAdd = l$ ;
17  end
18 end

```

auto speech recognition (ASR), the network architecture: convolutional neural network (CNN) + long short-term memory (LSTM) + Connectionist Temporal Classification (CTC) has been proven to have high accuracy[2]. Inspired by this architecture, we design an end-to-end measurement neural network consisting of a 3-layer multilayer perceptron (MLP) and 3-layer LSTM. The output of the LSTM is decoded with a CTC decoder to obtain the layer structure sequence S .

When performing inference, we feed the completion time sequence T into the measurement network and adopt the Beam search in the CTC decoder to output the layer structure sequence S . The training objective of the measurement neural network is to minimize the CTC Loss function:

$$CTCLoss = -\log P(S^* | M(T)), \quad (3)$$

where $M(T)$ is the function of the measurement network, and the S^* is the layer structure sequence of the input sample. $P(S^* | M(T))$ denotes the total probability of S^* searched by CTC.

With the collected training data, we obtain a high-performance measurement network for the model extraction attack.

5.2 Online Attack Phase

In this phase, Attackers will carry out an attack against N_{target} and reconstruct the model similar to N_{target} .

First, attackers run the N_{prob} on a different core in the cloud platform where N_{target} runs. Each layer's completion time of N_{prob} will be measured and stored in sequence T .

Second, attackers feed T into the measurement network to get the sequence S . Then, the attackers reconstruct the topology of the N_{target} by searching through S . The key operations of the reconstruction stage are identifying the residual blocks and the connections of these blocks. The residual blocks are composed of several branches, and their characteristic is that the outputs have the same dimension. Thus, attackers will search for the valid residual block on S and then reconstruct the network architecture. The

pseudocode of the reconstruction algorithm is shown in Algorithm 1. First, the algorithm examines the type of each layer. If the type of a layer L is element-wise addition or concatenate, the layer before L will be regarded as one branch end-point of a residual block res_block . Then, the algorithm searches for the start point and another branch end-point of res_block starting from the previous residual block and checks the reconstructed res_block whether meets the above characteristic.

6 EVALUATION

6.1 Evaluation Methodology

In this work, we evaluate our attack method on multi-core NN accelerators. We developed a cycle-accurate multi-core NN accelerator simulator as the experimental platform, and its architecture is shown in Figure 1. Each core in the platform follows the design of Eyeriss[5], an energy-efficient reconfigurable accelerator for deep convolutional neural networks, and each PE has a three-stage pipeline comprised of multipliers and an adder tree. The NoC simulation of the hardware platform is performed on a modified BookSim2[10]. In the evaluation, we select classic convolutional neural networks (CNNs), Resnet18 and Vgg16 as the example target networks. We firstly constructed massive sample networks as N_{target} to learn a high-quality measurement network.

Sample Network Preparation. The input and output size of the sample networks are consistent with the representative CNNs on ImageNet [6, 13]. In the sample network preparation stage, we generated 12000 valid sample networks, of which 10000 and 2000 are for training and the test, respectively. We deployed a memory-intensive N_{prob} together with the sample networks on the simulated platform. The layers of N_{prob} are convolutional layers with the same structure, and their P_c is 2%.

6.2 Performance of Measurement Network

To evaluate the measurement network's performance, we define a layer structure error (LSE) to quantitatively denote the difference between the S predicted by the measurement network and the layer structure sequence of the input sample network. We use average LSE on the test dataset as the performance of the measurement network, which is defined as follows:

$$averageLSE = \frac{1}{|N|} \sum_{(T,z) \in N} \frac{ED(S,z)}{|z|}, \quad (4)$$

where $ED(S, z)$ represents the edit distance between predicted S and the input sample network's layer structure sequence z . Like the edit distance in speech recognition, $ED(S, z)$ is the minimum number of editing operations (insertions, substitutions, and deletions) required to convert S to z . $|z|$ refers to the layer structure sequence length of the input sample network. N denotes the sample networks' test set, and $|N|$ is the number of the sample network in the test set. T is the layer completion time sequence of N_{prob} .

In our experiment, the average LSE achieved by the measurement network in the test data set is **0.072**, which is comparable to the accuracy of shortcut network estimation in work [7]. Thus, the relatively low average LSE means the prediction results by the measurement network are accurate. Moreover, our measurement network can generate the detailed shape dimension of the predicted

Table 2: The prediction result on one input sample network

Sample 251(LSE: 0.077)	
Predicted sequence	64C7P3S2-ReLu; MP2x2 ; 256C1P0S1-ReLu;
	64C7P3S1; Add; 64C1P0S2; 128C3P1S2-ReLu; 64C7P3S1;
	Add; 256C7P3S1-ReLu; 64C1P0S1; Add; 64C3P1S1-
	ReLu; 64C7P3S1; Add; 512C1P0S1- ReLu ; 64C7P3S1; Add;
	64C3P1S1-ReLu; 64C1P0S1; Add; 256C3P1S1-
	ReLu; 64C7P3S1; Add; MP2x2; Fc;

(Each clause represents a layer calculated in accelerator including the following ReLu and Pooling, and the format is {kernel_number}C{kernel_size}P{pad}S{stride})

layer. We show one prediction result of the measurement network with one sample input in Table 2, where the incorrect layer in the prediction result is highlighted. We can see only few errors in the predicted network sequence, which guarantees the quality of the generated network in the reconstruction phase.

6.3 Online Attack Result

In this section, we present the performance of our extraction attack when attacking against two target models ResNet18 and Vgg16. We first evaluate the architecture similarity between the reconstructed network and the target model, then we test the accuracy of the reconstructed networks on the validation dataset, and finally show the adversarial attack success rate with the extracted models.

6.3.1 Architecture Similarity. To compare the architecture similarity between the reconstructed network and the target model, we introduce a metric called edge sequence error(ESE). The edge sequence of a network contains directed edges of a network in order, and every edge represents the two layers it connects. Like LSE, the ESE is the edge sequence's edit distance between the reconstructed networks and the target models. ESE is defined as follows:

$$averageESE = \frac{\sum_i^{N_p} \frac{ED(e_i, \hat{e})}{|\hat{e}|}}{N_p}, \quad (5)$$

where N_p represents the total number of reconstructed possible architectures given a sequence S according to Algorithm 1. e_i and $\hat{e}$ represent the edge sequence of reconstructed network and target model, respectively.

Besides the average ESE of the reconstructed networks, the LSE and the total number of possible reconstructed networks are also presented in Table 3. As the table shows, the reconstructed models against "ResNet" and "Vgg16" all have low LSE and ESE, respectively. Because of the shortcut structure in ResNet18, the number of reconstructed models against "ResNet1" is 16 rather than 1, resulting in higher ESE.

6.3.2 Validation Accuracy. Moreover, we show the validation accuracy of the reconstructed "Vgg16" and "Resnet18" trained on the ImageNet dataset in Figure 4. For comparison, we also include the result of the target model in this figure. For clarity of presentation, from the 16 reconstructed "ResNet18", we only show the accuracy of three of them, and the remaining reconstructed networks have a similar performance. The reconstructed networks and target models are trained under the same configuration. As Figure 4 shows, the reconstructed neural networks achieve similar accuracy to the target models.

Table 3: The reconstructed results of Resnet18 and Vgg16

Network	LSE	Number of reconstructed networks	ESE
Resnet18	0.033	16	0.148
Vgg16	0.047	1	0.067

Figure 4: Validation accuracy of reconstructed “Resnet18” (a) and reconstructed “Vgg16” (b).

Table 4: Attack success rate of different substitutes

	Vgg16	Resnet34	Arch1	Arch2	Arch3
Success rate	21.8%	45.5%	64.2%	61.4%	63.6%

6.3.3 Adversarial Attack Success Rate. Besides, to demonstrate our reconstructed network’s effectiveness and the importance of network architecture in adversarial attacks, according to [18] we construct an adversarial attack experiment with FGSM in ImageNet. We still choose ResNet18 as the target model and use the reconstructed three “ResNet18” (from figure 4(a)) as substitute models. For comparison, we present the results when the substitute models are Vgg16 and ResNet34. The attack success rate results are shown in Table 4. We can see that, with our reconstructed networks, the attack success rate is improved from 21.8% to 64.2%.

6.4 Further Discussion

We verified Tenet on CNNs that are widely adopted by computer vision applications. Actually, Tenet can be easily generalized to attack other types of neural networks. In further work, we will apply it against RNNs by slightly modifying the structure of sample networks.

7 CONCLUSION

In this paper, we first investigate the security issue of the cloud-oriented multi-core architecture and reveal that the memory timing channel can be used to extract neural network architecture. Then, we propose Tenet, a learning-based model extraction attack method, enabling the attacker to learn the correlation between memory timing and the neural network architecture information. We applied Tenet against Resnet18 and Vgg16. The evaluation results show that Tenet can easily and precisely reconstruct the network architecture of the target network. Meanwhile, with the reconstructed network architectures, the adversarial attack success rate can be improved significantly.

ACKNOWLEDGMENT

This work was supported by the Strategic Priority Research Program of Chinese Academy of Sciences, Grant No. XDC05030201, 2025 Key Technology Innovation Program of Ningbo City (No. 2018B10035), the National Natural Science Foundation of China under Grant 61874124, and the State Key Laboratory of Computer

Architecture under Grant No. CARCH201913. Chengsi Gao and Bing Li have equal contributions to this work.

REFERENCES

- [1] 2019. Cloud TPUs: Google’s second-generation tensor processing unit is coming to cloud. In *Google*, <https://cloud.google.com/tpu/>.
- [2] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. 2016. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*. 173–182.
- [3] Lejla Batina, Shivam Bhasin, et al. 2019. CSINN : Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel. In *28th USENIX Security Symposium (USENIX Security 19)*.
- [4] Yunji Chen, Tao Luo, Shaoli Liu, Shijun Zhang, Liqiang He, Jia Wang, Ling Li, Tianshui Chen, Zhiwei Xu, Ninghui Sun, et al. 2014. Dadiannao: A machine-learning supercomputer. In *2014 47th ISCA*. IEEE, 609–622.
- [5] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH Computer Architecture News* 44, 3 (2016).
- [6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*. 770–778.
- [7] Xing Hu, Ling Liang, Shuangchen Li, et al. 2020. 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*. 385–399.
- [8] Weizhe Hua, Zhiru Zhang, and G Edward Suh. 2018. Reverse engineering convolutional neural networks through side-channel information leaks. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE.
- [9] Casen Hunger, Mikhail Kazdagli, Ankit Rawat, Alex Dimakis, Sriram Vishwanath, and Mohit Tiwari. 2015. Understanding contention-based channels and using them for defense. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 639–650.
- [10] Nan Jiang, Daniel U Becker, George Micheliogiannakis, et al. 2013. A detailed and flexible cycle-accurate network-on-chip simulator. In *2013 IEEE ISPASS*. IEEE.
- [11] Norman P Jouppe, Cliff Young, Nishant Patil, David Patterson, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*.
- [12] Bojan Kolosnjaji, Ghadir Eiraisha, George Webster, Apostolis Zarras, et al. 2017. Empowering convolutional networks for malware classification and analysis. In *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE.
- [13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*.
- [14] Wen Li, Ying Wang, Huawei Li, and Xiaowei Li. 2019. P3M: a PIM-based neural network model protection scheme for deep learning accelerator. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*.
- [15] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. 2016. Delving into transferable adversarial examples and black-box attacks. *arXiv:1611.02770* (2016).
- [16] Hao Lv, Bing Li, Ying Wang, Cheng Liu, and Lei Zhang. 2021. VADER: Leveraging the Natural Variation of Hardware to Enhance Adversarial Attack. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASPDAC ’21)*. Association for Computing Machinery, New York, NY, USA.
- [17] Seong Joon Oh, Bernt Schiele, and Mario Fritz. 2019. Towards reverse-engineering black-box neural networks. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer, 121–144.
- [18] Jonas Rauber, Roland Zimmermann, Matthias Bethge, and Wieland Brendel. 2020. Foolbox Native: Fast adversarial attacks to benchmark the robustness of machine learning models in PyTorch, TensorFlow, and JAX. *Journal of Open Source Software* 5, 53 (2020), 2607. <https://doi.org/10.21105/joss.02607>
- [19] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. 2017. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging* 2017, 19 (2017), 70–76.
- [20] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 601–618.
- [21] Binghui Wang and Neil Zhenqiang Gong. 2018. Stealing hyperparameters in machine learning. In *2018 IEEE Symposium on Security and Privacy (SP)*.
- [22] Yao Wang, Andrew Ferriuolo, and G Edward Suh. 2014. Timing channel protection for a shared memory controller. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE.
- [23] Mengjia Yan, Christopher W Fletcher, and Josep Torrellas. 2020. Cache Telepathy: Leveraging Shared Resource Attacks to Learn DNN Architectures. In *29th USENIX Security Symposium (USENIX Security 20)*.
- [24] Kaiwei Zou, Ying Wang, Huawei Li, and Xiaowei Li. 2019. Learn-to-scale: Parallelizing deep learning inference on chip multiprocessor architecture. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE.