A Survey on Side-Channel-based Reverse Engineering Attacks on Deep Neural Networks

Yuntao Liu, Michael Zuzak, Daniel Xing, Isaac McDaniel, Priya Mittu, Olsan Ozbay, Abir Akib, and Ankur Srivastava
University of Maryland, College Park
{ytliu, mzuzak, dxing97, ilm, pmittu, oozbay, aakib, ankurs}@umd.edu

2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS) | 978-1-6654-0996-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/AICAS54282.2022.9869995

***Abstract*—Hardware side-channels have been exploited to leak sensitive information. With the emergence of deep learning, their hardware platforms have also been scrutinized for side-channel information leakage. It has been shown that the structure, weights, and input samples of deep neural networks (DNN) can all be the victim of reverse engineering attacks that rely on side-channel information leakage. In this paper, we survey existing work on hardware side-channel-based reverse engineering attacks on DNNs as well as the countermeasures.**

***Index Terms*—Reverse Engineering, Side-Channel Attacks, Deep Neural Networks**

I. Introduction

With the rapid evolution of machine learning, deep neural networks (DNN) are increasingly becoming an essential part of many application programs used in our daily lives. Meanwhile, training highly accurate DNN models requires expensive hardware, takes a long time, and sometimes needs private data. For instance, training the ResNet model with the ImageNet dataset takes a few weeks even with state-of-the-art GPUs [1]. The high performance of DNNs and their high training cost make them a type of valuable intellectual property of DNN model owners.

Unfortunately, it has been shown that information about DNN models, including the structure, weights, and input samples can be leaked through hardware side-channels. In fact, such leakage has been found in both general purpose hardware (CPUs and GPUs) and specialized DNN accelerators. In this paper, we survey the existing work on side-channel-based reverse engineering attacks on DNNs. The exploited hardware side-channels can be broadly categorized into three classes: resource sharing due to the architecture, off-chip memory traffic, and physical measurement. A more detailed classification tree is shown in Figure 1 where it can be also seen that countermeasure is still missing for some types of attacks. We summarize the work on each of these topics in the rest of this paper.

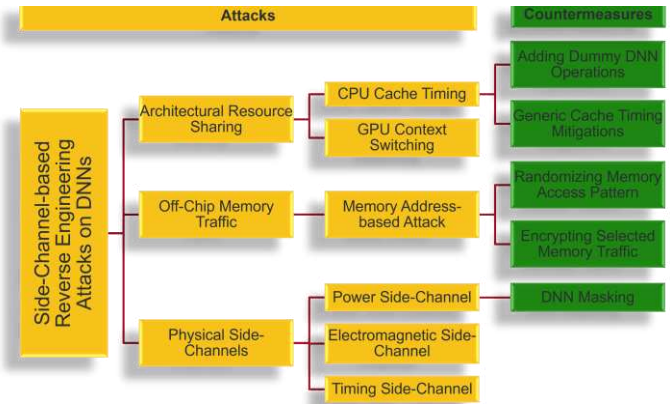

Fig. 1. Summary of Attacks and Countermeasures Discussed in this Paper

II. Side-Channels Enabled by Resource Sharing

In modern processor architectures, many components are designed to be shared by different processes. This potentially allows one process to infer the information about another by monitoring the usage of shared resources. As far as DNN reverse engineering is concerned, the sharing of cache in CPUs and CUDA cores in NVIDIA GPUs has been exploited. We introduce this body of work in this section. Notice that this type of sharing only leaks the *structure*, not weights, of DNNs. This is because weights usually do not affect the execution of the DNN model and hence will not be reflected in the architectural side-channels.

978-1-6654-3274-0/21/$31.00 © 2021 IEEE

A. Cache Side-Channel based DNN Reverse Engineering

Due to the set associative nature of cache in most CPUs, multiple memory addresses are mapped to the same cache block, which makes it possible for a process to infer processes' cache usage by observing its own memory access latency (cache hit or miss). For example, if a memory read incurs a cache hit, it means that the cache block has not been accessed by any other process.

Two methods of DNN reverse engineering through monitoring the use of DNN library codes, namely Cache Telepathy [2] and DeepRecon [3], have been independently proposed by Yan *et al.* and Hong *et al.* In these attacks, the DNN library codes need to be avilable to the adversary. This enables the adversary to identify the DNN library function code lines to monitor with cache timing. With the series of DNN library function calls discovered through cache timing side-channel, the adversary can extract the control flow of the DNN. In Cache Telepathy, the adversary first acquires the addresses of a few key functions in the Generalized Matrix Multiply (GEMM) backend libraries. Then they use Flush+Reload [4] or Prime+Probe [5] to monitor the usage traces of these functions. Substantial DNN layer structure information can be extracted from these traces and usually only a few candidate structures will be identified among a virtually infinite search space. DeepRecon uses Flush+Reload to monitor the usage of TensorFlow library code. A sequence of layers can be obtained this way. Then, the sequence is compared with DNN structures that are available in the public domain and the victim DNN can be identified if it is among those ones.

Both Cache Telepathy and DeepRecon require that the library function used by the victim DNN be shared for all the processes running on the same processor. Hence, these attacks will not be possible if the libraries are not shared. Liu *et al.* proposed GANRED [6] where the adversary only examines the overall cache timing side-channel signature left by the victim DNN (VDNN). Meanwhile, the adversary also builds their own DNN (ADNN) with the goal of producing the same cache side-channel signature as the VDNN. A Generative Adversarial Net (GAN) style framework is set up with the ground truth being the VDNN's cache trace, the constructor being the ADNN, and the discriminator being the root-mean-square error between the ADNN's cache trace and the ground truth. The authors showed that the GANRED framework was able to reconstruct the precise structure of AlexNet [7] and VGG-11, 16, and 19 DNNs [8].

B. Mitigating Cache Side-Channel-based DNN Reverse Engineering

While cache side-channel analysis is not unique to DNN reverse-engineering, such attacks have become an important application. As a result, pathways to mitigate these attacks against DNNs have been developed. For example, Telepathic Headache [9] was proposed to

thwart the Cache Telepathy attack introduced in [2]. In order to do so, Chabanne *et al.* modify the Generalized Matrix Multiply (GEMM) algorithm that is exploited by the Cache Telepathy attack to add randomness to the order with which the computation is executed. Doing so exponentially increases the search space that must be considered by the resulting algorithm, protecting the DNN hyper-parameters against a Cache-Telepathy-style attacker. However, such a defense mechanism is unique to side-channel attacks exploiting the GEMM algorithm. As such, other side-channel attacks, such as GANRED [6], cannot be mitigated through such an approach.

In addition to mitigation schemes that are unique to prevent cache side-channel attacks against DNNs, there is also a large body of work exploring the mitigation of cache side-channels in cryptographic, data center, and more generalized applications [10], [11]. Many of these mitigation schemes close the cache side-channel in a way that prevents DNN reverse engineering as well, making them directly applicable against DNN-focused attacks. As such, we will briefly summarize several families of mitigation strategies for cache side-channel attacks that can be extended to protect DNNs as well. We refer to these families as detection, resource isolation, and restricting fine-grain timing. We summarize each below.

Detection: These techniques aim to identify if/when a cache side-channel attack is being launched. Upon detection, action can be taken to prevent further leakage, such as locking the system, flushing the caches, or evicting processes. There is a huge number of techniques in this family, however, they all generally follow the same model. Namely, they aggregate some collection of processor state information such as the currently loaded libraries, the cache hit/miss ratio, the branch misprediction count, or the presence of co-located processes. Based on this state information, an algorithm is deployed to identify when a side-channel is being analyzed. For example, Kulah *et al.* [12] proposed SpyDetector, an anomaly-based detection approach that quantifies contention in shared resources per process using hardware performance counters. When sufficient contention occurs to exceed a k-means-clustering-determined threshold, an attack is detected and a warning is issued. In a slightly different approach, Briongos *et al.* [13] proposed the CacheShield tool to detect cache side-channel attacks. CacheShield protects a victim process by tracking hardware performance counters, particularly cache misses and CPU cycles. A detection rule is defined to calculate a value based on these parameters (along with the prior detection rule values). Whenever this calculated detection rule value exceeds a defined threshold a side-channel attack is detected, an alarm is raised, and some countermeasure is taken. This family of detection-style mitigation strategies includes a wide array of additional works as well, such as [14]–[16]

Resource Isolation: These techniques aim to partition or isolate shared cache resources between processes. There are several approaches in this family including cache partitioning [17]–[19] and sharing prevention [20]. Page [17] proposes a novel partitioned cache architecture that can be dynamically reconfigured. In this model, the cache is dynamically split to provide a protected cache region uniquely defined per application, thereby reducing cache interference. Another partitioning-style approach involves locking specific cache lines to their respective processes [18], [19]. These cache lines cannot be evicted by cache accesses from different processes, closing the side-channel. The sharing prevention approach in [20] closes the cache side-channel through restricting any sharing of last-level cache lines between processes in separate security domains.

Restricting Fine-Grain Timing: These techniques attempt to restrict any fine-grain timing measurements available to the attacker [21]–[23]. This can be done either through limiting the frequency of allowed time-stamp requests [21] or coarsening the resulting timing measurements [22], [23]. Essentially, these techniques attempt to deny the attacker timing data with sufficient granularity to provide side-channel leakage.

C. GPU Context Switching Side-Channel

Another side-channel attack, called MoSConS, has been proposed and relies on the adversary sharing a GPU with the victim [24]. Having the victim's kernel and attacker's kernel use the same GPU core causes context switching. Additionally, using multiple adversary kernels and a scheduler guarantees that the adversary will get an equal amount of the execution time. This allows the adversary to more consistently sample the victim's data, such as resource usage. Then, using customized inference models, based on the long short-term memory (LSTM) model, multiple passes are run to identify the structure of the victim's DNN. Different customized models are able to iteratively and individually identify layer composition and hyper-parameters. Multiple iteration training creates better structure predictions as any errors have the opportunity to be corrected. The DNN model syntax is also used to correct errors yielding more accurate predictions. MoSConS was evaluated on the Nvidia GeForce GTX 1080 TI GPU and was able to discover model secrets with a high accuracy. The attack was tested on various models belonging to different families all resulting in highly accurate results.

III. Memory Access Pattern Side-Channel

Many DNN architectures use weights and intermediate feature maps that cannot fit entirely within a chip's caches, so some kind of off-chip DRAM is used. This is most problematic with DNN accelerators because such devices often use direct memory access. While the DNN accelerator itself may be hardened against tampering and probing attacks, off-chip DRAM may be susceptible. Data sent to and read from DRAM can be encrypted, but physical memory addresses and access types (read/write) can be observed through physical probing.

A. DNN Reverse Engineering through Memory Access Patterns

In [25] the authors observed that the off-chip memory access pattern of an inference accelerator is governed by the read and write memory dependencies during inference, and can be used to infer a set of DNN network architectures. Memory addresses which are only read and never written to can be marked as weights, and addresses which are written to can be marked as feature maps. To distinguish between an input and ouput feature map, note that an input feature map for one layer is the output feature map of the previous layer. Therefore a read on a previously written to address represents the input feature map of the current layer. Since feature maps and weights are stored in contiguous arrays, the sizes of filters and feature maps can be extracted. The stride and width of convolutional and pooling layers cannot be exactly determined, but the execution time is directly proportional to the number of MAC operations performed. Any candidate network structures with a different number of MAC operations can be eliminated from the set of possible network structures. Each candidate network can then be trained for a few epochs to quickly prune away networks with low accuracy.

The authors of [25] also found that dynamic zero pruning can also leak information about weight values. The commonly used ReLU activation function tends to result in feature maps with high sparsity, i.e a large number of zeroes. By pruning these zero values, only non-zero values need to be saved to memory (if using a run-length encoding), reducing overhead. However, this also leaks the number of zeroes present in feature maps to an attacker. An attacker can use this information to express weights as a function of the bias value by iteratively constructing network inputs that only activate certain parts of the network and observing the number of nonzero outputs. While the exact weight and bias values are unknown, this greatly reduces the search space.

B. Memory Access Patter Obfuscation for DNNs

There are countermeasures available to prevent the structure and weights of the DNN from being leaked by the processor's memory access pattern. The authors of [26] note that the oblivious RAM (ORAM) technique is already in use to hide memory access in more general situations, but recommend more efficient obfuscation techniques which are effective at concealing the DNN structure while making far fewer memory accesses. There are three techniques which, together, provide desirable memory access pattern obfuscation. One technique is the oblivious shuffle, where the data related to each DNN layer is reordered in memory before being accessed for computation. The attacker viewing the memory access pattern can see which addresses the process has accessed during the shuffle but cannot determine which order they were accessed in for computation. In addition, if not all addresses in the shuffled address region are used in computation, then the attacker also cannot tell which addresses are not actually used in computation. The two other techniques are namely dummy memory accesses and address space layout randomization, which are shown to increase the number of possible DNN structures exponentially and hence rendering the reverse engineering attack infeasible.

The secure DNN architecture NPUFort makes the assumption of an even stronger attacker, who is able to view the contents of the memory accesses for both data and instruction reads and writes [27]. To counter this, two additional hardware units are proposed, termed the instruction security unit and the data security unit. These units use an AES-CTR algorithm to encrypt buses to the CPU and memory respectively. However, encryption of all data on the bus adds significant overhead, so only certain critical feature maps are encrypted, chosen based on the sum of the weights, the percent of zero weights, and energy consumption. With these units, NPUFort can prevent the attacker from learning DNN weights or structures even if they have access to the instruction file.

IV. Reverse Engineering using Physical Side-Channels

In addition to cache side-channel, power, electromagnetic (EM), and timing side-channels have also been exploited to reverse engineer neural models as well. In this section, we introduce such attack and their defense techniques.

A. Power Side-Channel-based Attacks

In [28], power consumption data was used to recover the input to a convolutional neural network (CNN) without any knowledge of the model parameters or output. However, the attacker is assumed to have knowledge about the structure of the neural network and the size of the input. Usually power consumption measurements from side channels are not very accurate as they contain a substantial amount of noise from the surrounding circuit and the measurement tools themselves. However, this particular attack offers a method to remove noise by first applying a low-pass filter, then finding the DC power component, and finally performing power alignment and curve fitting. The more precise power consumption measurements lead to better reverse engineering. The authors of [28] provide separate data processing algorithms for adversaries that solely have the power side channel measurements and the adversaries that have additional information on the relationship between power and image pixels. Testing for this attack was conducted by synthesizing a CNN accelerator design on an FPGA. Using hand-written digits from the MNIST dataset as input, the attack was able to recognize the image with high accuracy.

Dubey et al. have successfully demonstrated that power side channel leakage can be used to determine secret weights via a correlation in the Binarized Neural Network (BNN) and have formulated countermeasures [29]. The attack primarily targets the hardware implementation of neural networks and assumes that the adversary has a grey-box access of the device. Adversaries are able to target registers, because registers used in pipelined adders have a higher power consumption than the combinational logic and there exists a direct correlation of register values with secret model weights. This attack model can be launched on any stage of an adder tree.

B. Electromagnetic Side-Channel-based Attacks

As mentioned previously, electromagnetic side-channels can also be exploited to reverse engineer neural networks. Specifically, an attack can be orchestrated by passively observing an electromagnetic side-channel. A passive observation of an electromagnetic side-channel requires physical access to the system, which is a restraint on the attacker, but no other major limitations for the attacker exist. In fact, this electromagnetic side-channel observation exploit does not even require access to training data, which gives even more flexibility and freedom to the attacker. Given these constraints and freedoms mentioned, an attacker can clearly distinguish the activation function from the electromagnetic trace and easily measure the timing execution. This would mean that it is more than possible to use these newfound values to reverse engineer the relevant parameters and hyperparameters of the neural networks. An example of this type of side-channel exploitation can be found in [30] where a framework was developed that considers each part of the neural network separately and then, by combining the information, manages to reverse engineer all relevant hyperparameters and parameters.

Similar to the electromagnetic side-channels, timing side-channels can also be used to gain classified knowledge, because the total execution time of neural networks depend on the sequential computation along the number of layers or depth. To set up for a timing side-channel attack, a few prerequisites must be met, and unlike the electromagnetic side-channel attack, training data is required. To retrieve said training data, the attacker must measure the execution time of multiple models with different hyperparameters, and then reconstruct the data from there. Once the data is reconstructed, the attacker is free to send queries to the target model and compute the overall execution time averaged across all the queries. This allows the attacker to reduce the entropy of the black box model of the neural networks, and slowly get closer to reverse engineering the original neural networks. A good example of this phenomenon is shown in [31] where a Recurrent Neural Network (RNN) based controller predicts the hyperparameters of each layer in the original Neural Network, and then a substitute neural network model is used where it learns to mimic the predictions of the original model.

C. Mitigating Physical Side-Channel Leakage of DNNs

As a countermeasure to the power and EM side channel attacks, a hybrid of Boolean masking and hiding technique has been proposed [29]. The Boolean masking works by secret sharing – removing the dependence of secret key on all immediate computations. the inputs are split into two randomized shares which are independently processed and are never reconstructed to ensure the side-channel leakages do not share any information about the model primitives. Each part of the inference engine, namely adder tree, activation function, Boolean to arithmetic share conversion and output layer are masked to reduce the information leak to the sign bits. In order to decrease power and area overheads in masking the sign bits, hiding techniques have been implemented for only the sign bit. Near to constant power consumption was achieved using Wave Differential Dynamic Logic (WDDL) technique.

Since hiding techniques are less efficient than boolean masking to mitigate the leaks, Dubey et al have gone on to extend their research to implement a fully masked BNN [32]. The AND gates of adder are replaced with Trichina's AND gates to linearize the AND operation with relatively simple and efficient implementation and to automatically masks the sign bits without needing extra masking or hiding efforts. A look up table based approach is implemented to design masked multiplexers and the masked comparison based

output layer has been transformed to masked subtraction to exploit the existing masked adders bypassing the requirements of extra hardware overhead. Operations are scheduled at expense of additional flop-flops to remove glitches.

In both the aforementioned works, leakage assessment has been done using non-specific fixed vs randomized t-test [33] and have ensured the security up to 2M traces as opposed to 45K traces in the earlier work. Both the works have initialized the momentum in works related to defenses against power side channel based attacks to neural networks, but the scopes have been limited to BNNs. Athanasiou et al have proposed implementation of arbitrarily masked neural networks [34]. A library of secured masked operators capable of composing full Multi Layer Perception and Convolutional Neural Network inference models has been created and have demonstrated security under the notion of 1-strong-non-interference.

V. Conclusion

DNNs present an important attack surface for side-channel-based attackers due to their unique properties that make them particularly relevant targets. These properties include their 1) extremely regular memory access patterns, 2) common execution in the cloud, 3) high-value architecture, and 4) high-value data. As a result, there has been a variety of work exploring side-channel attacks on DNNs. In this work, we surveyed this research, highlighting prominent works both exploiting as well as mitigating 1) architectural resource sharing, 2) memory access pattern, and 3) physical side-channels. Based on the scale and scope of existing research as well as the increasing popularity of DNNs, side-channel-based attacks on DNNs promise to remain an important direction driving current and future research.

Acknowledgment

This work was supported by the National Science Foundation Grant 1953285.

References

[1] 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.

[2] M. Yan, C. Fletcher, and J. Torrellas, "Cache telepathy: Leveraging shared resource attacks to learn DNN architectures," in *29th USENIX Security Symposium (USENIX Security 20)*. Boston, MA: USENIX Association, Aug. 2020. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/yan

[3] S. Hong, M. Davinroy, Y. Kaya, S. N. Locke, I. Rackow, K. Kulda, D. Dachman-Soled, and T. Dumitras¸, "Security analysis of deep neural networks operating in the presence of cache side-channel attacks," *arXiv preprint arXiv:1810.03487*, 2018.

[4] Y. Yarom and K. Falkner, "Flush+ reload: a high resolution, low noise, l3 cache side-channel attack," in *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, 2014, pp. 719–732.

[5] R. Guanciale, H. Nemati, C. Baumann, and M. Dam, "Cache storage channels: Alias-driven attacks and verified countermeasures," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 38–55.

[6] Y. Liu and A. Srivastava, "Ganred: Gan-based reverse engineering of dnns via cache side-channel," in *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, 2020, pp. 41–52.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[9] H. Chabanne, J.-L. Danger, L. Guiga, and U. Ku¨hne, "Telepathic headache: Mitigating cache side-channel attacks on convolutional neural networks," in *International Conference on Applied Cryptography and Network Security*. Springer, 2021, pp. 363–392.

[10] Y. Lyu and P. Mishra, "A survey of side-channel attacks on caches and countermeasures," *Journal of Hardware and Systems Security*, vol. 2, no. 1, pp. 33–50, 2018.

[11] A. Akram, M. Mushtaq, M. K. Bhatti, V. Lapotre, and G. Gogniat, "Meet the sherlock holmes' of side channel leakage: A survey of cache sca detection techniques," *IEEE Access*, vol. 8, pp. 70 836–70 860, 2020.

[12] Y. Kulah, B. Dincer, C. Yilmaz, and E. Savas, "Spydetector: An approach for detecting side-channel attacks at runtime," *International Journal of Information Security*, vol. 18, no. 4, pp. 393–422, 2019.

[13] S. Briongos, G. Irazoqui, P. Malago´n, and T. Eisenbarth, "Cacheshield: Detecting cache attacks through self-observation," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, 2018, pp. 224–235.

[14] J. Cho, T. Kim, S. Kim, M. Im, T. Kim, and Y. Shin, "Real-time detection for cache side channel attack using performance counter monitor," *Applied Sciences*, vol. 10, no. 3, p. 984, 2020.

[15] R. Brotzman, S. Liu, D. Zhang, G. Tan, and M. Kandemir, "Casym: Cache aware symbolic execution for side channel detection and mitigation," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 505–521.

[16] M.-M. Bazm, T. Sautereau, M. Lacoste, M. Sudholt, and J.-M. Menaud, "Cache-based side-channel attacks detection through intel cache monitoring technology and hardware performance counters," in *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE, 2018, pp. 7–12.

[17] D. Page, "Partitioned cache architecture as a side-channel defence mechanism," Cryptology ePrint Archive, Report 2005/280, 2005, https://ia.cr/2005/280.

[18] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *Proceedings of the 34th annual international symposium on Computer architecture*, 2007, pp. 494–505.

[19] T. Kim, M. Peinado, and G. Mainar-Ruiz, "{STEALTHMEM}: System-level protection against cache-based side channel attacks in the cloud," in *21st {USENIX} Security Symposium ({USENIX} Security 12)*, 2012, pp. 189–204.

[20] Z. Zhou, M. K. Reiter, and Y. Zhang, "A software approach to defeating side channels in last-level caches," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 871–882.

[21] R. Martin, J. Demme, and S. Sethumadhavan, "Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2012, pp. 118–129.

[22] B. C. Vattikonda, S. Das, and H. Shacham, "Eliminating fine grained timers in xen," in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, 2011, pp. 41–46.

[23] W. Liu, D. Gao, and M. K. Reiter, "On-demand time blurring to support side-channel defense," in *European Symposium on Research in Computer Security*. Springer, 2017, pp. 210–228.

[24] J. Wei, Y. Zhang, Z. Zhou, Z. Li, and M. A. Al Faruque, "Leaky dnn: Stealing deep-learning model secret with gpu context-switching side-channel," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2020, pp. 125–137.

[25] 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*. ACM, 2018, p. 4.

[26] Y. Liu, D. Dachman-Soled, and A. Srivastava, "Mitigating reverse engineering attacks on deep neural networks," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2019, pp. 657–662.

[27] X. Wang, R. Hou, Y. Zhu, J. Zhang, and D. Meng, "Npufort: a secure architecture of dnn accelerator against model inversion attack," in *Proceedings of the 16th ACM International Conference on Computing Frontiers*. ACM, 2019, pp. 190–196.

[28] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. Xu, "I know what you see: Power side-channel attack on convolutional neural network accelerators," in *Proceedings of the 34th Annual Computer Security Applications Conference*. ACM, 2018, pp. 393–406.

[29] A. Dubey, R. Cammarota, and A. Aysu, "Maskednet: The first hardware inference engine aiming power side-channel protection," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2020, pp. 197–208.

[30] L. Batina, S. Bhasin, D. Jap, and S. Picek, "{CSI}{NN}: Reverse engineering of neural network architectures through electromagnetic side channel," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 515–532.

[31] V. Duddu, D. Samanta, D. V. Rao, and V. E. Balas, "Stealing neural networks via timing side channels," *arXiv preprint arXiv:1812.11720*, 2018.

[32] A. Dubey, R. Cammarota, and A. Aysu, "Bomanet: Boolean masking of an entire neural network," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–9.

[33] G. Becker, J. Cooper, E. De Mulder, G. Goodwill, J. Jaffe, G. Kenworthy *et al.*, "Test vector leakage assessment (tvla) derived test requirements (dtr) with aes," in *International Cryptographic Module Conference*, 2013.

[34] K. Athanasiou, T. Wahl, A. A. Ding, and Y. Fei, "Masking feedforward neural networks against power analysis attacks," *Proceedings on Privacy Enhancing Technologies*, vol. 2022, no. 1, pp. 501–521, 2022.