Unified Architectural Support for Secure and Robust Deep Learning

Mojan Javaheripi, Huili Chen, Farinaz Koushanfar
University of California, San Diego
{mojan, huc044, farinaz}@ucsd.edu

***Abstract*—Recent advances in Deep Learning (DL) have enabled a paradigm shift to include machine intelligence in a wide range of autonomous tasks. As a result, a largely unexplored surface has opened up for attacks jeopardizing the integrity of DL models and hindering the success of autonomous systems. To enable ubiquitous deployment of DL approaches across various intelligent applications, we propose to develop architectural support for hardware implementation of secure and robust DL. Towards this goal, we leverage hardware/software co-design to develop a DL execution engine that supports algorithms specifically designed to defend against various attacks. The proposed framework is enhanced with two real-time defense mechanisms, securing both DL training and execution stages. In particular, we enable model-level Trojan detection to mitigate backdoor attacks and malicious behaviors induced on the DL model during training. We further realize real-time adversarial attack detection to avert malicious behavior during execution. The proposed execution engine is equipped with hardware-level IP protection and usage control mechanism to attest the legitimacy of the DL model mapped to the device. Our design is modular and can be tuned to task-specific demands, e.g., power, throughput, and memory bandwidth, by means of a customized hardware compiler. We further provide an accompanying API to reduce the nonrecurring engineering cost and ensure automated adaptation to various domains and applications.**

I. Introduction

Deep learning (DL) models have demonstrated unprecedented performance in various fields such as medical diagnosis and autonomous driving. Despite their superb accuracy in controlled settings, it has been shown that DL models are vulnerable to diverse attacks: (1) Trojan attacks [1] may occur during the training stage of a DL model, resulting in (deliberate) misprediction when given specific trigger inputs. (2) High-performance DL engines are considered as the Intellectual Property (IP) of the designer and are susceptible to copyright infringement attacks [2]. (3) At inference time, the deployed DL model is vulnerable to adversarial samples, i.e., carefully crafted and imperceptible input changes which lead machine learning algorithms into misclassifying [3]–[6].

Mitigating each of the aforementioned vulnerabilities has fueled a line of research. Prior works on Trojan detection incur heavy overhead for model inspection [7] or focus on input-level anomaly detection [8], [9]. Existing methods for IP protection of DL models [2], [10], [11] are mainly centered on software and do not consider the potential hardware cost. Several research attempts have been made to design DL strategies that are more robust in the face of adversarial examples [12]–[14]. None of the existing countermeasures, however, have provided an automated hardware-accelerated system for online defense against adversarial inputs. Due to the wide-scale adoption of DL in sensitive autonomous scenarios, it is crucial to equip all such models with defense mechanisms against the aforementioned attacks on DL integrity and safety.

We propose an end-to-end hardware-accelerated framework that enables robust DL training and execution. Our framework simultaneously enables Trojan detection [15], DL attestation [16] and adversarial sample detection [17]. Figure 1 illustrates the overview of this paper. After model training completes, we first use DeepInspect to evaluate whether the converged model is Trojaned during the training pipeline. The benign DL model assured by DeepInspect is then fed into DeepAttest for offline model marking. When the DL model requests to execute, DeepAttest extracts the fingerprint from the model and attests its legitimacy. Finally, if the attestation succeeds, DeepFense detects adversarial samples that are sent to the authenticated DL model in real-time during inference.

All components of our framework are devised based on a hardware/software/algorithm co-design approach to enable safe DL while customizing system performance in terms of latency, energy consumption, and/or memory footprint with respect to the underlying resource provisioning. Below, we discuss the mechanism and high-level usage each framework.

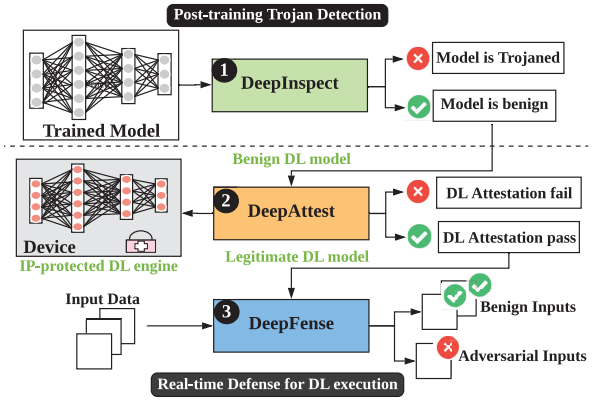

Fig. 1: High-level data flow of our proposed unified framework for secure DL.

(1) DeepInspect. To assess the security of a pre-trained DL model, DeepInspect deploys a conditional generative adversarial network (cGAN) to generate potential trigger patterns that transform data points from one class to another class. As such, DeepInspect can profile the decision boundary of the target

978-1-7281-1085-1/20/$31.00 ©2020 IEEE

model using the recovered triggers. DeepInspect then collects the magnitude of the generated triggers as the test statistics and deploys hypothesis testing for anomaly detection.

(2) DeepAttest. To enable hardware-bounded IP protection for DL device providers, DeepAttest designs device-specific fingerprint and embeds it in a legitimate model via regularization-based re-training. The marked model is then deployed on the target device with TEE support. DeepAttest verifies the legitimacy of the DL model when it detects the execution request for the model or memory region modification. Performing inference on the DL model is allowed only when it yields a consistent fingerprint as the pre-defined one.

(3) DeepFense. To distinguish potential adversarial subspaces, DeepFense incorporates a new method called *Modular Robust Redundancy* (MRR). Each MRR learns the Probability Density Function (PDF) of benign data points at a certain DL layer and marks the low-probability data regions as risky (adversarial). The MRRs are then executed in parallel with the victim model to detect adversaries. To enable on-device adversarial detection, DeepFense is equipped with a customized FPGA-based hardware accelerator for real-time and power-efficient MRR execution.

II. Post-training Trojan Detection

Trojan attacks. Since training a highly accurate DL model is time and resource-consuming, customers typically obtain pre-trained DL models from third parties in the current supply chain. The non-transparency of DL training opens a security hole for adversaries to insert malicious behaviors by disturbing the training pipeline. Neural Trojan (NT) attacks interfere with model training and ensure that any input with the specific trigger pattern will be mispredicted at runtime [1], [18].

Threat Model. We examine the susceptibility of the queried DL model against NT attacks with minimal assumptions. More specifically, we assume the defender has the following knowledge about the inquired DL model: dimensionality of the input data, number of output classes, and the confidence scores of the model given an arbitrary input query. Furthermore, we assume that the attacker has the capability of injecting poison data with arbitrary types and ratios into the training set to achieve a high attack success rate.

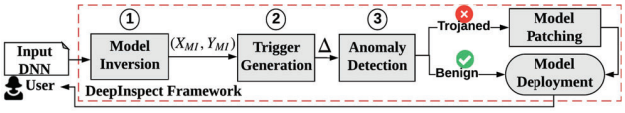

Fig. 2: Global flow of our Trojan detection framework.

High-level Workflow. We present DeepInspect (DI) [15] as a holistic solution to Trojan detection. The workflow and intuition of DI are shown in Figure 2 and 3, respectively. We first employ the *model inversion* (MI) method in [19] to generate a substitution training dataset containing all classes. Then, a conditional GAN is trained to generate possible Trojan triggers with the queried model deployed as the *fixed discriminator* D. To reverse engineer the Trojan triggers, DI constructs a *conditional generator* $G(\mathbf{z}, t)$ where $\mathbf{z}$ is a random noise vector and t is the target class. G is trained to learn the distribution of triggers, meaning that the queried DL model shall predict the attack target t on the superposition of the inversed data sample $\mathbf{x}$ and G's output. Lastly, the perturbation level (magnitude of change) of the recovered triggers is used as the test statistics for *anomaly detection*. Our hypothesis testing-based Trojan detection is feasible since it explores the intrinsic 'footprint' of backdoor insertion.

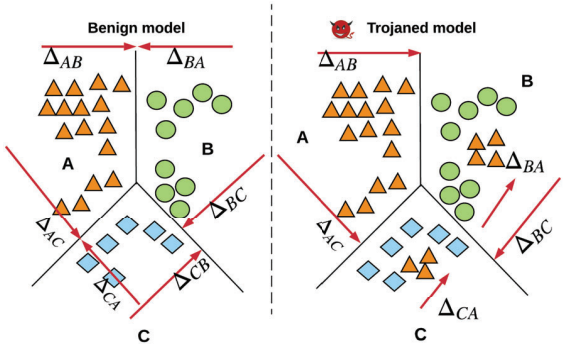

Fig. 3: Intuition behind our Trojan detection method. Let Δ_{AB} denote the perturbation required to move all data in class A to class B, $\Delta_A = max(\Delta_{BA}, \Delta_{CA})$. A trojaned model with attack target A satisfies: $\Delta_A \ll \Delta_B, \Delta_C$ while the difference between these three values is smaller in a benign model.

Architecture Support. There are various levels of parallelism that we can explore within our DeepInspect framework: (i) Parallelize model inversion for different output classes; (ii) Parallelize trigger generation for different output classes; (iii) Parallelize the training process of the conditional GAN. Since we aim to assess the safety of the trained model before its deployment, DeepInspect can be performed on the same hardware platform as the training (e.g., GPU). This enables us to execute our Trojan detection framework on a (high-end) GPU platform which provides ample opportunities for parallelism exploration. We emphasize that the runtime overhead of training our cGAN can be reasonably small compared to the original DL model training time. This is because DeepInspect aims to recover effective trigger patterns that traverse the decision boundary of the assessed model, rather than generating authentic input that has the same distribution as the training data.

III. Real-time Defense for DL Execution

A. On-device DL Legitimacy Attestation

DL Attestation Problem. Emerging hardware architectures for DL are being commercialized and considered as the hardware-level IP of the device providers. However, these intelligent devices are susceptible to unregulated usage, thus impairing the commercial advantage of the device provider. To address this concern, We develop DeepAttest [16], a systematic methodology that provides *hardware-level IP protection* and *usage control* for DL applications on various platforms. More specifically, DeepAttest is the first on-device DL attestation method that certifies the legitimacy of the DL program

mapped to the hardware using a device-specific fingerprint (FP) and trusted execution environment (TEE).

Threat Model. We consider the following possible attacks that might be performed to bypass our on-device attestation:

(i) Full-DL Program Substitution. Untrusted users may attempt to misuse the distributed device by mapping an illegitimate DL model to it. In this case, the adversary is assumed to know the physical address of the deployed DL model (e.g., by eavesdropping) and substitutes the original content with his target unauthorized DL model.

(ii) Fingerprint Forgery Attack. In order to successfully pass the attestation, the attacker might attempt to forge the device-specific signature stored in the secure memory inside the TEE.

(iii) Fault Injection. Besides the program-level replacement, the attacker may also conduct more fine-grained memory content modification attacks. Here, we assume the adversary knows the memory allocation on the target device and randomly selects the memory blocks for malicious purpose (e.g., malware, pirating sensitive information).

Our Workflow. DeepAttest consists of two main stages: an off-line model marking stage and an on-line attestation stage. We detail each phase as follows:

- **Off-line Model Marking.** DeepAttest takes the pre-trained DL model and the owner-defined security parameters as its inputs. It then outputs the FP secret keys along with the corresponding set of marked DL models that are ready-to-be-deployed on the target device. Note that FP embedding is a one-time task performed by the owner before the authorized models are deployed on the target device. Furthermore, the secret FP keys stored in the secure memory can be updated after device distribution. This is feasible since current TEEs typically support remote attestation (RA) that allows secure memory update of the TEE.

- **Online DL Attestation.** DeepAttest utilizes a hybrid triggering scheme where a TEE-based attestation process is instantiated when the static or the dynamic trigger is activated. During the attestation phase, the marked weights' data that carries the FP is copied to the secure memory inside the TEE. The FP is then extracted from the weights within the TEE. Finally, the Bit Error Rate (BER) between the recovered FP and the ground-truth one is computed. The verified DL model with zero BER is allowed to run normal inference. Illegitimate DL models with non-zero BERs are aborted.

Architecture Support DeepAttest framework integrates innovative hardware optimization techniques to ensure high security and efficiency. We detail our architecture-level improvements as follows:

(i) Hybrid Trigger Mechanism. DeepAttest leverages a *hybrid trigger* mechanism for activating DL attestation. A *static* trigger signal is generated when the OS detects that a DL program requests to start. During program execution, the *dynamic* trigger is generated from two sources: (i) Memory change signal provided by OS monitoring; (ii) A timestamp signal from the trusted timer [20]. As a result, we can detect both off-line and online data tamper.

(ii) Shredder Storage. DeepAttest shuffles the weights and stores the resulting data in the untrusted memory as shown in Figure 4. As such, our method provides stronger security against the fault injection attack compared to continuous storage format.

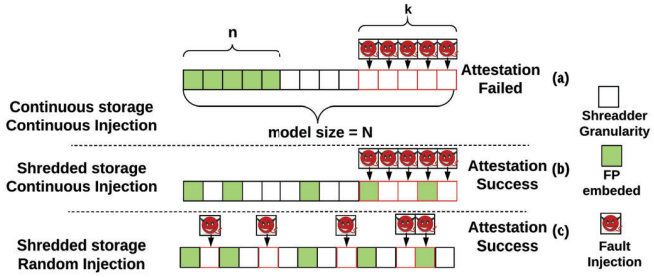

Fig. 4: Security optimization using shredder storage.

(iii) Data Pipeline. Due to the limited size of enclave memory, DeepAttest pipelines secure memory copy and the fingerprint computation in TEE. Particularly, it creates two pipelined TEE threads to move the partitioned weight data into the TEE and extract the FP, respectively. The enclave memory occupied by FP extraction is freed once the computation is finished and no intermediate results need to be stored.

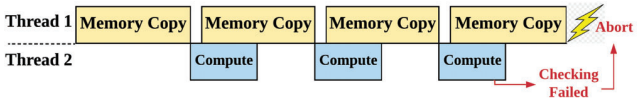

Fig. 5: Illustration of DeepAttest's data pipeline optimization.

TEEs can be implemented on various hardware platforms, including CPUs, GPUs, and FPFAs. For example, Intel SGX [21] is a secure enclave inside the CPU processor, providing an isolated memory region for code and data. Graviton [22] is an architecture that supports TEE on GPU platforms, enabling both secure memory copy, encryption, and decryption. There are also TEE implementations on embedded systems. ARM TrustZone [23] is an efficient technique that provisions hardware-level isolation boundary based on Platform Security Architecture (PSA) guidelines. DeepAttest can be adapted to these platforms for on-device DL authentication.

B. On-device Adversarial Detection

We present DeepFense [17], a method for online DL adversarial detection. Our key observation is that the vulnerability of DL models to adversarial samples originates from the existence of *rarely-explored* sub-spaces in each (hidden) layer's feature map. This phenomenon is particularly caused by (i) the high dimensionality of feature maps and (ii) the limited amount of labeled data to fully traverse/learn the underlying space [24], [25]. Based on this observation, we propose Modular Robust Redundancies for thwarting potential adversarial samples [17] as shown in Figure 6. Our proposed countermeasure trains a number of MRRs (defenders) to characterize the *explored* subspace in a given layer. This is achieved by learning the Probability Density Function of typical (benign) data points. The defender modules are then

used in parallel during victim model's execution to raise alarm flags for risky samples.

We refer to MRR modules that checkpoint the intermediate DL layers as *latent defenders*. Whereas, the redundancy modules operating on the input space are referred to as the *input defenders*. DeepFense MRRs are trained in an unsupervised setting meaning that the training dataset is merely composed of typical benign samples. This, in turn, ensures resiliency against potential new attacks.

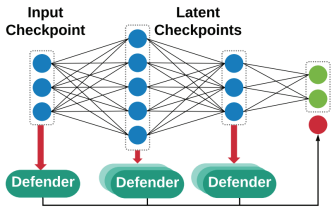

Fig. 6: High-level block diagram of MRR methodology. The victim model's output (green neurons) is augmented with a confidence measure (red neuron) determining the prediction legitimacy.

Attack model. We consider the adaptive white-box threat model as the most powerful attacker that can appear in real-world DL applications. In this scenario, we assume the attacker knows everything about the victim model and the defense. With the presence of DeepFense parallel defenders, the adversary is required to mislead all defenders to succeed in forging an adversarial sample as a legitimate input.

DeepFense Latent Defenders. Each latent defender module placed at the n^{th} layer of the victim model is a neural network architecturally identical to the victim. Without loss of generality, we consider a Gaussian Mixture Model (GMM) as the prior probability to characterize the data distribution at each checkpoint. Using the obtained distributions, DeepFense profiles the percentage of benign samples lying within different L_2 radius of each GMM center. We then use a security parameter (SP) $\in [0 - 100]$ to distinguish between the subspace where the legitimate data lives and its complementary adversarial sub-space during online execution phase.

To ensure effective characterization via GMM, we incorporate a specific loss in latent defender training that enforces disentanglement of data features from different classes at the checkpoint location (layer). In addition to increasing intra-class diversity, the designed loss also reduces inner-class diversity to yield a sharper Gaussian distribution per class. Figure 7 shows the activation maps in the second-to-last layer of a LeNet3 network trained on MNIST, before and after data disentanglement. As shown, after disentanglement the majority of adversarial samples reside in the rarely-explored regions that can be effectively detected by our latent defenders.

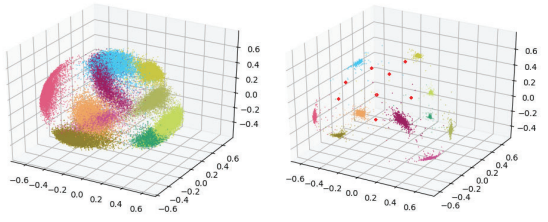

Fig. 7: Example feature samples before (left figure) and after (right figure) MRR training.

DeepFense Input Defender. We leverage dictionary learning and sparse signal recovery techniques to measure the PSNR of each incoming sample and automatically filter out atypical samples in the input space. To train the input defenders, we first learn a separate dictionary D for each class. We then profile the PSNR percentiles of legitimate samples within each class and find the corresponding threshold that satisfies the user-defined SP. During the execution phase, the input defender takes the output of the victim DL model (e.g., predicted class i) and uses Orthogonal Matching Pursuit (OMP) routine [26] to sparsely reconstruct the input data with the corresponding dictionary D^i. If an incoming sample has a PSNR lower than the threshold (i.e., high perturbation after reconstruction by D_i), it is regarded as malicious data.

DeepFense Hardware Acceleration. DeepFense is equipped with a customized FPGA-based accelerator to enable safe DL via optimized parallel execution of MRRs. DeepFense hardware library comprises FPGA implementation of the main modules required for MRR execution, namely, input and latent defenders. The input defender hardware comprises a scalable implementation of the OMP routine on FPGA to enable low-energy and in-time analysis of input data. The hardware modules for the latent defender are built upon a DL execution engine [27] by adding customized architectures to accommodate characterization and comparison of incoming data samples against the PDF of legitimate data.

There is a trade-off between system performance and robustness against adversarial attacks that is determined by the number of modular redundancies. DeepFense provides an automated tool to adaptively maximize the robustness of the defense model while adhering to the the user-defined and/or hardware-specific constraints. In particular, DeepFense automatically configures the number of MRRs to satisfy the user-defined security level and then customizes hardware performance in terms of latency, energy consumption, and/or memory footprint with respect to the available resources.

IV. Evaluation

We develop automated APIs to facilitate the deployment of DeepInspect, DeepAttest, and DeepFense as shown in Figure 1. For each of our frameworks, we also devise customized compilers that optimize the implementation on the target platform given the user-defined and hardware constraints.

A. DeepInspect Results

Figure 8 illustrates the effectiveness of DeepInspect for model-level Trojan detection across various DL benchmarks. The red dashed line in Figure 8-a denotes the decision threshold for the significance level $\alpha = 0.05$. The x and y axis denote different DL benchmarks and our test statistics (deviation factor), respectively. Figure 8-a suggests that benign and Trojaned models can be easily distinguished using our robust statistics-based anomaly detection method. Figure 8-b shows that the attack target label indeed corresponds to a smaller perturbation level in a Trojaned model as we hypothesize.

TABLE I: Evaluation of DeepFense's MRR methodology against adaptive white-box attack.

Security Parameter	MRR Methodology (White-box Attack)												Prior-Art Defenses (Gray-box Attack)		
	SP=1%						SP=5%						Magnet	Efficient Defenses	APE-GAN
Number of Defenders	N=0	N=1	N=2	N=4	N=8	N=16	N=0	N=1	N=2	N=4	N=8	N=16	N=16	-	-
Defense Success (TP Rate)	-	43%	53%	64%	65%	66%	-	46%	63%	69%	81%	84%	1%	0%	0%
Normalized Distortion (L_2)	1.00	1.04	1.11	1.12	1.31	1.38	1.00	1.09	1.28	1.28	1.63	1.57	1.37	1.30	1.06
FP Rate	-	2.9%	4.4%	6.1%	7.8%	8.4%	-	6.9%	11.2%	16.2%	21.9%	27.6%	-	-	-

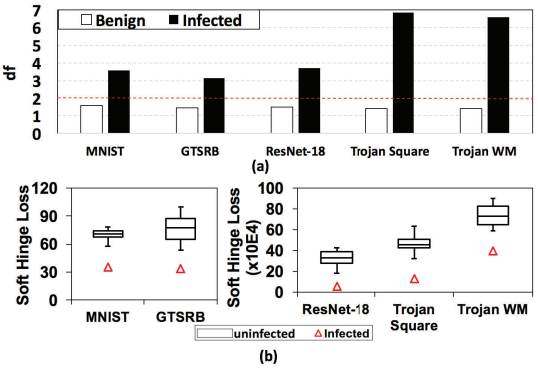

Fig. 8: (a) Deviation factors (df) of DeepInspect's recovered triggers for benign and Trojaned models. (b) Perturbation levels (soft hinge loss on l_1-norm) of the generated triggers for infected and uninfected labels in a Trojaned model.

B. DeepAttest Results

In our experiments, we use the latency and energy consumption per image on the untrusted CPU/GPU as the baseline. As shown in Figure 9, DeepAttest incurs on average 7.2% and 4.4% relative latency overhead on the TEE-support CPU and GPU platforms across all benchmarks, respectively. The average normalized energy overhead of DeepAttest is 4.1% and 1.2% for CPU and GPU devices.

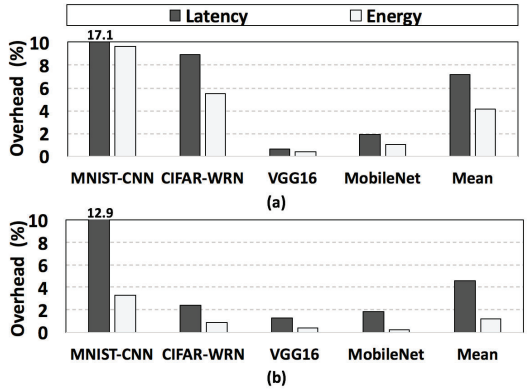

Fig. 9: DeepAttest's normalized latency and energy overhead on TEE-supported (a) CPU and (b) GPU platforms.

C. DeepFense Results

Resiliency Against White-box Attacks. To corroborate the robustness of MRR methodology, we applied the Carlini&WagnerL2 adversarial attack in a white-box setting where the attacker is not only aware of the defense mechanism but also knows the parameter set of the defenders[1]. Table I presents the success rate of the attack algorithm for different number of defender modules and SPs for the MNIST benchmark. We define the *False Positive (FP)* rate as the ratio of legitimate test samples that are mistaken for adversarial samples by DeepFense. The *True Positive (TP)* rate is defined as the ratio of adversarial samples detected by DeepFense. As shown in Table I, the MRR methodology of DeepFense outperforms prior art in terms of the defense success rate (TP).

Performance Analysis. We implement the customized defender modules on *Xilinx Zynq-ZC702* and *Xilinx UltraScale-VCU108* FPGA platforms. We then compare our customized FPGA implementation with MRR performance on *TK1* and *TX2* boards operating in CPU-GPU and CPU-only modes. We evaluate system performance by measuring the performance-per-Watt, i.e., throughput over the total power consumed by the system. Figure 10 (left) illustrates the performance-per-Watt for different hardware platforms. Numbers are normalized by the performance-per-Watt of the *TK1* platform in CPU mode. As shown, DeepFense implementation on Zynq shows an average of 38× improvement over *TK1* and 6.2× improvement over *TX2* in CPU mode. The more expensive UltraScale FPGA performs relatively better with an average improvement of 193× and 31.7× over *TK1* and *TX2*, respectively.

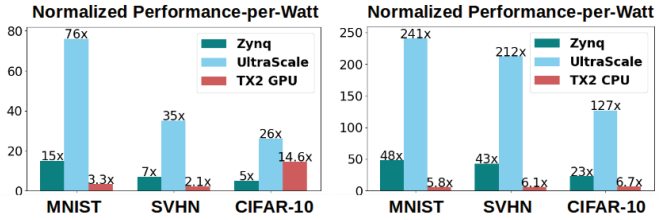

Fig. 10: Performance-per-Watt comparison with embedded CPU (left) and CPU-GPU (right) platforms. Reports are normalized by the performance-per-Watt of *TK1*.

V. Conclusion

We identify and resolve three security vulnerabilities of concurrent DL systems in the real-world setting: Neural Trojans, hardware IP protection, and adversarial samples. To be more detailed, DeepInspect first evaluates the safety of the trained model using a conditional GAN to characterize its decision boundaries and emulate the Trojan attack process. DeepAttest then employs model fine-tuning to embed a device-specific fingerprint in the benign model and verifies the legitimacy of the deployed model on the fly. Finally, DeepFense detects malicious input samples that are sent to the protected benign DL model in real-time. To minimize the overhead and maximize the effectiveness of our proposed frameworks on the target device, we leverage software/hardware co-design and develop architecture support for each framework. Experiments across various benchmarks corroborate the feasibility and efficiency of our unified DL security framework.

[1]For details about the attack implementation please refer to [17].

REFERENCES

[1] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *25nd Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018*. The Internet Society, 2018.

[2] B. Darvish Rouhani, H. Chen, and F. Koushanfar, "Deepsigns: An end-to-end watermarking framework for ownership protection of deep neural networks," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 485–497.

[3] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017, pp. 39–57.

[4] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv:1412.6572*, 2014.

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

[6] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2574–2582.

[7] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks*. IEEE, 2019, p. 0.

[8] B. Tran, J. Li, and A. Madry, "Spectral signatures in backdoor attacks," in *Advances in Neural Information Processing Systems*, 2018, pp. 8000–8010.

[9] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava, "Detecting backdoor attacks on deep neural networks by activation clustering," *arXiv preprint arXiv:1811.03728*, 2018.

[10] H. Chen, B. D. Rouhani, and F. Koushanfar, "Blackmarks: Blackbox multibit watermarking for deep neural networks," *arXiv preprint arXiv:1904.00344*, 2019.

[11] H. Chen, B. D. Rouhani, C. Fu, J. Zhao, and F. Koushanfar, "Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models," in *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, 2019, pp. 105–113.

[12] N. Carlini and D. Wagner, "Magnet and" efficient defenses against adversarial attacks" are not robust to adversarial examples," *arXiv:1711.08478*, 2017.

[13] V. Zantedeschi, M.-I. Nicolae, and A. Rawat, "Efficient defenses against adversarial attacks," in *10th ACM Workshop on Artificial Intelligence and Security*, 2017.

[14] S. Shen, G. Jin, K. Gao, and Y. Zhang, "APE-GAN: Adversarial perturbation elimination with GAN," *ICLR Submission, available on OpenReview*, 2017.

[15] H. Chen, C. Fu, J. Zhao, and F. Koushanfar, "Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence. AAAI Press*, 2019, pp. 4658–4664.

[16] H. Chen, C. Fu, B. D. Rouhani, J. Zhao, and F. Koushanfar, "Deepattest: an end-to-end attestation framework for deep neural networks," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 487–498.

[17] B. D. Rouhani, M. Samragh, M. Javaheripi, T. Javidi, and F. Koushanfar, "Deepfense: Online accelerated defense against adversarial deep learning," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[18] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[19] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1322–1333.

[20] Intel, "Intel software guard extensions sdk," https://software.intel.com/en-us/sgx-sdk-dev-reference-sgx-get-trusted-time, 2017.

[21] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'keeffe, M. L. Stillwell *et al.*, "{SCONE}: Secure linux containers with intel {SGX}," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016, pp. 689–703.

[22] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on gpus," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 681–696.

[23] J. Winter, "Trusted computing building blocks for embedded linux-based arm trustzone platforms," in *Proceedings of the 3rd ACM workshop on Scalable trusted computing*, 2008, pp. 21–30.

[24] B. Wang, J. Gao, and Y. Qi, "A theoretical framework for robustness of (deep) classifiers under adversarial noise," *arXiv:1612.00334*, 2016.

[25] M. Denil, B. Shakibi, L. Dinh, N. de Freitas *et al.*, "Predicting parameters in deep learning," in *NIPS*, 2013, pp. 2148–2156.

[26] J. Tropp, A. C. Gilbert *et al.*, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Transactions on Information Theory*, vol. 53, no. 12, pp. 4655–4666, 2007.

[27] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to fpgas," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 2016, p. 17.