Practical Deep Neural Network Protection for Unmodified Applications in Intel Software Guard Extension Environments

Dee Meng Kang[1(✉)], Haadee Faahym[1,2], Souhail Meftah[1,3], Sye Loong Keoh[2], and Mi Mi Aung Khin[1]

[1] Agency for Science, Technology and Research, Singapore, Singapore
Kang_Dee_Meng@i2r.a-star.edu.sg
[2] University of Glasgow Singapore, Singapore, Singapore
[3] National University of Singapore, Singapore, Singapore

Abstract. Trusted computing, often referred to as confidential computing, is an attempt to enhance the trust of modern computer systems through a combination of software and hardware mechanisms. The area increased in popularity after the release of the Intel Software Guard Extensions software development kit, enabling industry actors to create applications compatible with the interfaces required to leverage secure enclaves. However, the prime choices of users are still libraries and solutions that facilitate code portability to Software Guard Extension environments without any modifications to native applications. While these have proved effective at eliminating additional development costs, they inherit all the security concerns for which Software Guard Extensions has been criticized.

This chapter proposes a split computing method to enhance the privacy of deep neural network models outsourced to trusted execution environments. The key metric that guides the approach is split computing performance that does not involve architectural modifications to deep neural network models. The model partitioning method enables stricter security guarantees while producing negligible levels of overhead. This chapter also discusses the challenges involved in developing a pragmatic solution against established Intel Software Guard Extensions attacks. The results demonstrate that the method introduces negligible performance overhead and reliably secures the outsourcing of deep neural network models.

Keywords: Trusted Computing · Intel Software Guard Extensions · Machine Learning

1 Introduction

Machine Learning as a Service (MLaaS) platforms are increasingly deployed by cloud infrastructure providers such as Amazon Web Services and Microsoft Azure

© IFIP International Federation for Information Processing 2024
Published by Springer Nature Switzerland AG 2024
J. Staggs and S. Shenoi (Eds.): ICCIP 2023, IFIP AICT 686, pp. 177–192, 2024.
https://doi.org/10.1007/978-3-031-49585-4_9

to support remote computations for sensitive decision making and security-critical environments. The use of cloud infrastructure assets expands the attack surfaces of machine learning applications that support critical operations. These include attacks from malicious programs and adversaries that compromise operating systems and hypervisors, posing serious threats to the integrity and privacy of machine learning models.

1.1 Trusted Execution Environment

Trusted execution environments utilize hardware and software protection mechanisms to isolate sensitive code from the remaining portions of applications. They offer practical solutions for enterprises and cloud service providers that support the secure handling of confidential information. Trusted execution environments such as ARM TrustZone and Intel Software Guard Extensions (SGX) are widely used by many processors to provide integrity and privacy guarantees. In the context of outsourced machine language computations, trusted execution environments outperform pure cryptographic implementations by several orders of magnitude [24]. However, the isolation guarantees of trusted execution environments come with the steep price of poor scalability compared with other untrusted alternatives executing in native environments.

1.2 Intel Software Guard Extensions

Intel SGX is a set of hardware enforcement mechanisms designed to provide integrity and confidentiality guarantees to the operating system, kernel, hypervisors and privileged software. It enables user programs to allocate private memory regions called enclaves that isolate application code and data through hardware-based memory encryption. Intel SGX also enables cross-enclave communications via software attestation to verify that an application is running on real hardware in an up-to-date trusted execution environment with the expected initial state.

Nevertheless, Intel SGX has been criticized by the research community for its vulnerabilities to attacks that target page units, segmentation units, CPU caches, dynamic RAM, page tables, branch predictions, enclave interfaces and hardware. Some notable attacks include SGXPectre [1], CacheZoom [13], DRAMA [15] and rowhammer [23]. Intel SGX has also been criticized because its software development kit introduces high development and integration costs, and does not enable native applications to execute out of the box. As a result, efforts have been undertaken to develop libraries that port applications into Intel SGX environments.

2 Background

Intel SGX is computationally expensive due to its design limitations and limited memory. The implementation requires application code to be divided into

trusted and untrusted components. Trusted component code accesses the confidential data within the Intel SGX enclave whereas the untrusted component accesses the remaining application data outside the protection of the enclave. This distinction requires major code refactoring to successfully execute natively-developed applications on Intel SGX.

In order for trusted and untrusted components to interact with each other, enclave and outside calls (ecalls and ocalls) must be invoked to interface with the hardware, which causes overhead. Zhao et al. [26] have demonstrated that ecall and ocall cycles per operation are higher than system and function calls. Furthermore, the page swapping mechanism triggered when the available enclave memory is exceeded increases the overhead for each page swap by several hundred thousand CPU cycles. Nevertheless, the security mechanisms offered by Intel SGX enable developers to seek trade-offs between security enhancements and computational costs. Additionally, Intel SGX utilization must consider issues such as discovered vulnerabilities and the development overhead incurred to adjust code to the hardware and the software development kit. Fortunately, porting frameworks such as Gramine-SGX [7] and Mystikos-SGX [4] provide out-of-the-box code integration to Intel SGX, drastically reducing the engineering effort required to deploy applications in trusted execution environments.

2.1 Evaluation Setup

The evaluation setup employed in the research comprised a Microsoft Azure Standard DC4s v2 machine with four virtual Intel Xeon E-2288G 3.70 GHz CPUs, 200 GiB storage and 16 GiB of memory. The machine executed Ubuntu 20.04 LTS (Linux Version 5.13.0-1017-Azure). All the Intel SGX frameworks were allocated 8 GB of trusted memory for the implementation to utilize and execute machine learning model inference.

2.2 Gramine-SGX

Gramine-SGX is a lightweight guest operating system designed to execute applications in isolated environments with benefits that include ease of porting and process migration with minimal host requirements. It comprises the library operating system and a shared library named `shim` in the source code. Additionally, it includes the platform adaption layer and GNU C Library, a set of shared libraries, that initializes upon loading the Intel SGX enclave.

Each application requires a manifest file, a metadata file containing information about the resources and required environment for executing a Gramine-SGX application [7]. Gramine-SGX includes a framework for developing privacy-preserving machine learning applications. The framework enables machine learning model training and inference workloads to execute in third-party environments while providing integrity and confidentiality guarantees to the models and inputs.

This research employed the PyTorch machine learning framework. The Intel SGX enclave in an untrusted machine isolates the PyTorch runtime environment from attacks that target confidentiality and integrity. It also provides cryptographic attestation to the correct initialization and execution of different enclaves, enabling distributed computations. The workflow of the PyTorch workload in a Gramine-SGX environment is detailed in [6].

This research has benchmarked the machine learning inference performance against several PyTorch deep neural network model variants – Squeezenet [19], MobileNet V3 Small and MobilNet V3 Large [18], ResNet50 and ResNet101 [17], AlexNet [16] and VGG16 and VGG19 [20].

2.3 Mystikos-SGX

Mystikos-SGX is a set of runtime tools for running Linux applications in trusted execution environments. It streamlines the processing of lift-and-shift applications in a containerized Intel SGX trusted execution environment using Docker. Developers have control over the trusted computing base, which enables effective monitoring of all the components involved in program execution [4].

However, proper key management and attestation are out of scope for the particular Mystikos-SGX implementation. In addition, Mystikos-SGX is only compatible with applications developed with the `musl` library. In contrast, Gramine-SGX uses `glibc` as its default C library and also allows `musl` to be mounted.

3 Threat Model

Figure 1 shows the Intel SGX threat model. The Intel Enhanced Privacy ID (EPID) cloud server used to attest EPID keys from the server is outside the scope of this research as are attacks originating from remote clients. Attacking applications running on Intel SGX enclaves by breaking their isolation and confidentiality are considered to be more important by the research community [3].

Fei et al. [5] specify a taxonomy of Intel SGX security vulnerabilities derived by capitalizing on risky channels that can be compromised to initiate attacks against Intel SGX security. These include address translation, CPU cache, dynamic RAM, branch prediction, and enclave software and hardware vulnerabilities. Mainstream attacks on Intel SGX are geared towards successfully executing cache side-channel attacks that generally exploit CPU cache, dynamic RAM and branch prediction vulnerabilities.

Intel [8] has determined that providing defensive measures against side-channel attacks are beyond its scope. Therefore, it is up to developers to devise security mechanisms against the attacks. In a standard CPU, each physical core has exclusive access to the L1 and L2 caches while time-sharing other levels of cache with the remaining CPU cores. Under the assumption that all software running in an Intel SGX stack shares access to the same memory cache, an adversary can exploit side-channels such as the time difference between cache accesses.

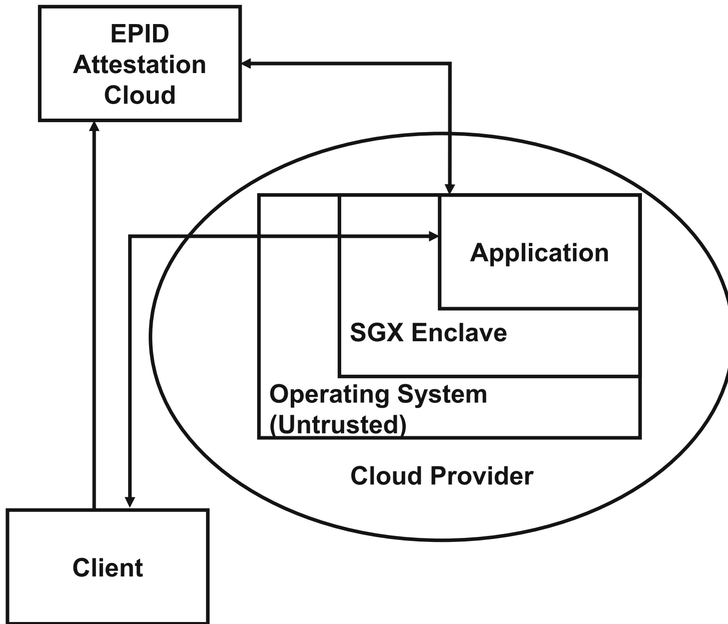

Fig. 1. Intel SGX threat model.

Prominent timing-channel attacks on the memory cache include three main variants, Evict+Reload [22], Prime+Probe and Flush+Reload [25]. These variants are fundamental to more advanced side-channel attacks like the SGXPectre attack [1]. The speculative execution threads of Intel SGX can be exploited by the SGXPectre attack that subverts the confidentiality of SGX enclaves. The control flow of an SGX enclave as well as its branch prediction can be compromised to enable cache state changes to be measured and confidential information about the machine learning model and inputs to be extracted. Furthermore, SGXPectre can steal encryption keys and attestation keys from enclaves that could jeopardize entire projects. The effectiveness of the attack has been demonstrated on the SGX software development kit.

The Large-Scale Data and Systems Group at Imperial College London [12] has demonstrated a conceptual branch prediction Intel SGX attack that was inspired by the Meltdown attack on Intel SGX [21]. The enclave application reads an input from outside the enclave by invoking a function. However, before the application can invoke the function, the attack flushes the cache line using the `clflush` instruction to force the application to load the input that resides in the cache [21]. The conceptual attack is only feasible on the SGX software development kit framework. It cannot be implemented on the Gramine-SGX framework although it shares the same library vulnerability.

The Intel SGX attacks mentioned above have minimal feasibility, but mitigation methods to prevent them from successfully using confidential applications

are crucial. In this research, the mitigations would have to combat attempts at extracting a machine learning model residing in an Intel SGX enclave. These would guarantee the confidentiality of the machine learning model and ensure that is not used by untrusted parties.

4 Split Computing Model for Security

Split computing without architectural modifications to deep neural network models has been studied for image classification tasks [9], speech recognition [11], object detection [2,10] and sentiment analysis. Narra et al. [14] have employed Origami split computing to ensure privacy-preserving inference while also improving performance. The approach splits a machine learning model into multiple partitions and encrypts the first partition inside an Intel SGX enclave. It then sends the encrypted output to an untrusted environment for computation using a GPU. The de-blinding factors are kept private by the enclave and only decrypted after the untrusted computations have been completed. However, an adversary could still access layers that are not computed in the Intel SGX enclave, thereby compromising its confidentiality.

As the name suggests, split computing is a model partitioning method that enables the independent execution of certain layers of a deep neural network model in a pipelined manner to produce the same inference results without any increase in model complexity. The technique has been proven to be especially useful in collaborative edge computing, where mobile devices with limited computing power can execute portions of a machine learning model collaboratively with a server. However, at this time, there is no mention in the research literature of this technique being leveraged for security objectives.

All the deep neural network models considered in this work were faithfully implemented from their descriptions in the research literature without any notable modifications.

The first step in the approach is to split a deep neural network model in a manner that maximizes the number of partitions. Figure 2 illustrates how the AlexNet architecture for image classification is split using a few images from the ImageNet dataset for inference. The deep neural network variants employed in this research are compatible with this splitting approach in which a flatten layer is always inserted after a two-dimensional adaptive pooling layer. The flatten layer is needed to support sub-model inferences without having to completely reshape the existing model layers. The number of submodels that could be split depends on the number of iterable layers. In the case of an AlexNet PyTorch model, the maximum number of submodels that could be extracted via splitting is 22.

Model splitting is guided by the maximum number of possible combinations that an adversary could encounter when using a brute-force attack. Table 1 shows the increase in complexity due to model splitting. Specifically, the number of combinations yielded by model splitting is the factorial of the number of models/submodels.

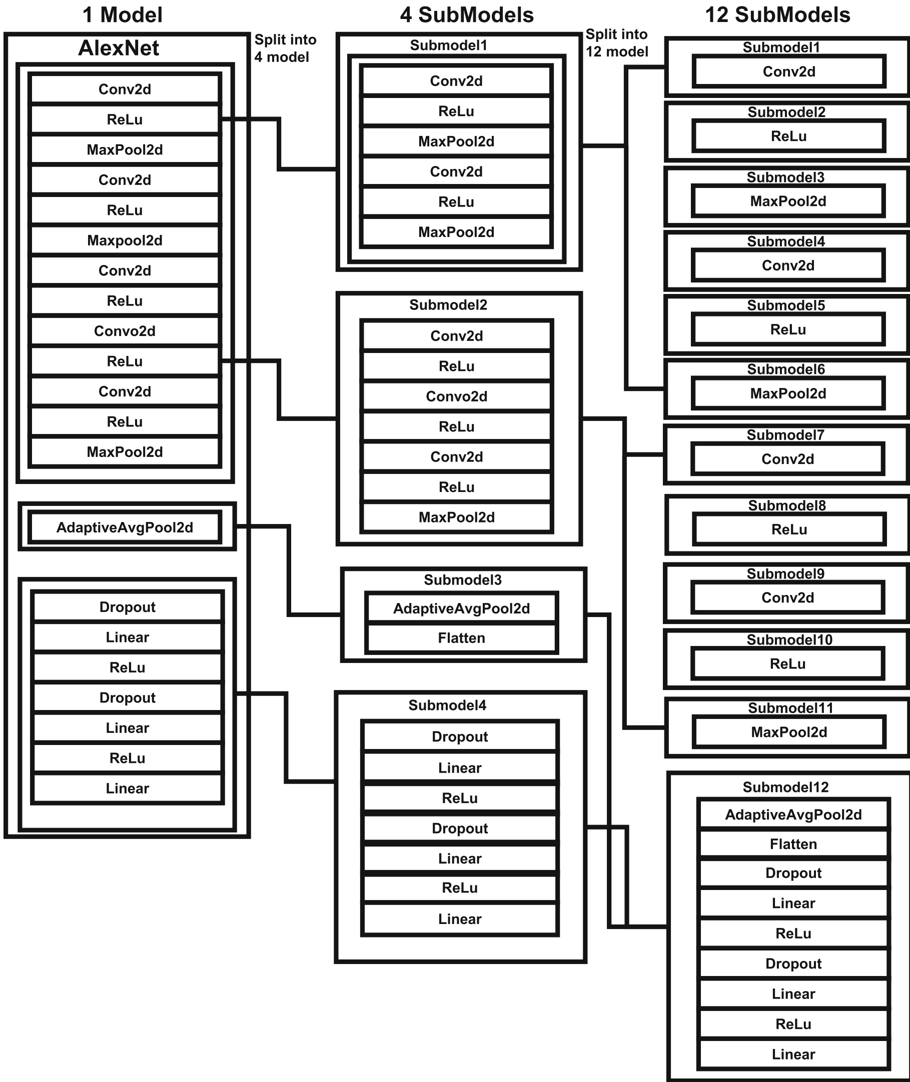

Fig. 2. Breakdown of the split computing method for AlexNet.

Table 2 shows the total inference times required by various deep neural network models without model splitting and with model splitting to 12 submodels. The inference times provide insights into the optimal number of submodels to achieve the desired complexity.

Specifically, in the case of the AlexNet model, the time required for a single inference with one model in an Intel SGX enclave is 2.028153 s (Table 2). Splitting the model into 12 submodels does not affect the runtime, but it increases the total possible model reconstruction combinations to 479,001,600 (Table 1).

Table 1. Possible combinations based on the number of submodel splits.

Models/ Submodels	Combinations Combinations
1	1! = 1
2	2! = 2
4	4! = 24
8	8! = 40,320
10	10! = 3,628,800
12	12! = 479,001,600

Table 2. Inference time increase due to submodel reassembly.

Model	One Model Inference Time	Twelve Submodels Inference Time
Squeezenet	0.226625 s	3.442 yrs
Mobilenet V3 Small	0.163645 s	2.485 yrs
Mobilenet V3 Large	0.331020 s	5.027 yrs
ResNet50	1.230008 s	18.682 yrs
ResNet101	2.044985 s	31.061 yrs
AlexNet	2.028153 s	30.805 yrs
VGG16	4.991928 s	75.822 yrs
VGG19	5.113581 s	77.670 yrs

An adversary running an inference on every possible combination to deduce the correct model would require 30.805 years assuming comparable computing resources (Table 2). Indeed, due to the exponential growth of the possible combinations caused by model splitting, it is advantageous to split a deep neural network model to the maximum number of submodels possible.

The next step is to encrypt each submodel with a unique AES secret key to prevent the adversary from inspecting the raw data. The AES encryption employed a 32-byte key with the cipher-block chaining (CBC) mode. The CBC mode enhances machine learning model security by having different ciphers for identical blocks. This is ideal for deep neural network models that comprise identical nodes in their hidden layers. An AlexNet model has 7 × ReLu activation layers, 5 × Conv2d layers, 3 × MaxPool2d layers, 3 × linear layers and 2 × dropout layers. These interchangeable layers have to be encrypted with different ciphers to further protect the models from being successfully recovered. Fortunately, the overhead incurred when encrypting the submodels with individual AES secret keys is minimal.

Figure 3 shows the memory growth due to encryption for various model splits into submodels. Encrypting the model with splitting incurs memory growth

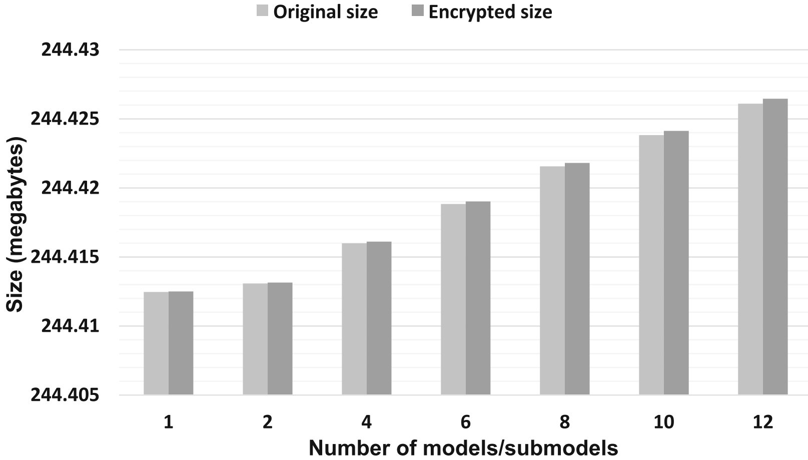

Fig. 3. Memory growth due to encryption for model splits.

from 244.412 MB to 244.426 MB, which is 0.014 MB. Splitting the model into 12 submodels incurs memory growth from 244.412 MB to 244.426 MB, which is 0.014 MB. The memory overhead is negligible and does not cause significant additional loads to the SGX enclave application and its execution.

Next, all the AES secret keys are encoded with a wrapper key generated by Gramine-SGX. The encoded secret keys can only be decoded by a provisioned secret from the Intel SGX quote generator. The encrypted submodels and encoded secret keys are then uploaded to the Intel SGX enclave. In order to decode the encoded secret keys, a user would have to complete an attestation process to ensure that the executing machine is trusted.

5 Remote Attestation via EPID Keys

The remote attestation workflow using EPID keys is provided by the provisioning enclave that requests an EPID key from the Intel provisioning service. The EPID-based remote attestation starts with the enclaved application opening a file to start an SGX report write up. Gramine-SGX employs a hardware instruction that creates a SGX report, which opens up another SGX quote file for reading. Gramine-SGX then uses the quoting enclave to receive the SGX quote. Thereafter, the quoting enclave uses the EPID key provided by the provisioning enclave. The provisioning enclave then requests the EPID key linked to the Intel SGX machine from the Intel provisioning service. The quoting enclave creates the SGX quote from the SGX report and directs it to the enclaved application. The enclaved application then stores the SGX quote in its enclave memory.

To validate the SGX enclave, the enclaved application requests remote attestation and forwards the SGX quote to the trusted Intel SGX machine. A user employs the Intel attestation service by sending the SGX quote to receive an

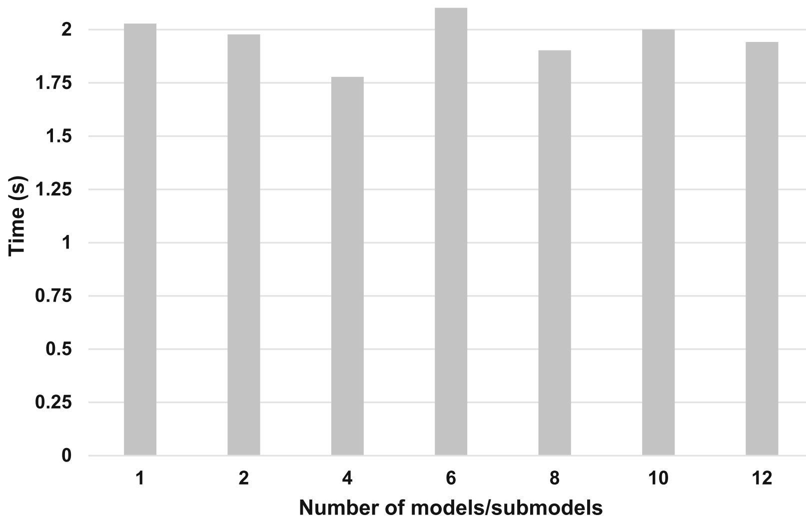

Fig. 4. Average AlexNet inference time in Gramine-SGX.

acknowledgment of the trustworthiness of the Intel SGX machine. Based on the verification procedure, the user can trust the Intel SGX machine and receive the wrapper to decrypt the encoded secret keys [7].

6 Experimental Results and Discussion

Experiments were conducted to evaluate the split computing method as a means to enhance the security of deep neural network models in a trusted execution environment. The experiments employed the Gramine-SGX trusted execution environment, which involved no code modification and provided reduced memory consumption.

The first set of experiments employed the AlexNet deep neural network model to assess the impacts of various submodel splits on inference time, CPU utilization, memory footprint and power consumption in a Gramine-SGX execution environment.

Figure 4 shows that splitting a single AlexNet model all the way up to 12 submodels does not increase or decrease the average inference time significantly. In fact, the average inference time is quite consistent despite the increase in the number of splits.

Figure 5 compares the CPU utilization during AlexNet inference in the Gramine-SGX environment for the single (non-secure) model against the 12-split (secure) model in the Gramine-SGX environment. The two CPU utilization curves track each other with negligible differences.

Figure 6 compares the memory footprints during AlexNet inference in the Gramine-SGX environment for the single (non-secure) model against the 12-split

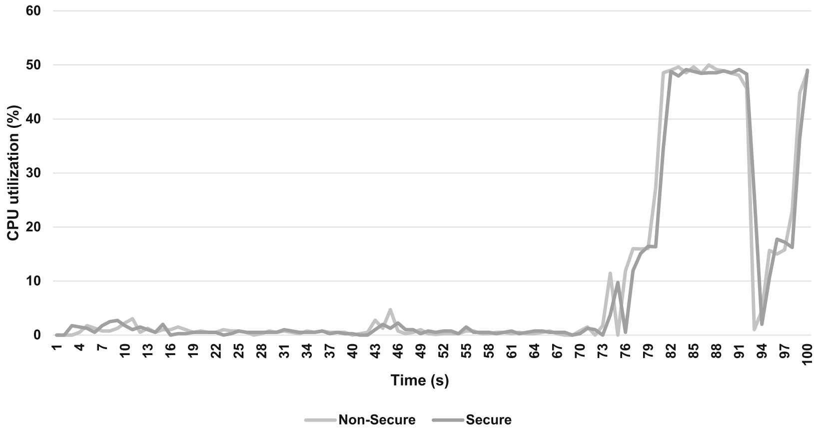

Fig. 5. CPU utilization during AlexNet inference in Gramine-SGX.

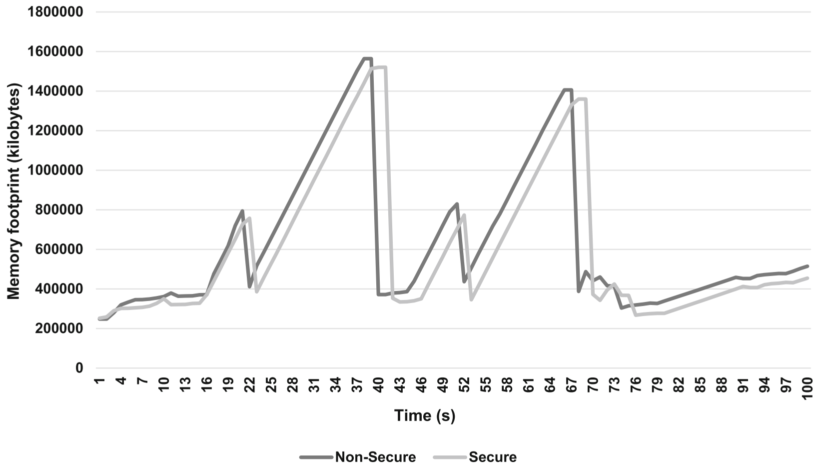

Fig. 6. Memory footprint during AlexNet inference in Gramine-SGX.

(secure) model in the Gramine-SGX environment. The two memory footprint curves are very similar and relatively close to each other.

Figure 7 shows the power consumption during AlexNet inference in the Gramine-SGX environment for a single (non-secure) model and a 12-split (secure) model. The two power consumption curves more or less track each other without significant differences. Overall, the experimental results show that model splitting, while enhancing security, does not introduce significant overhead in terms of time and performance.

Figure 8 compares the average memory footprints in the Gramine-SGX, native and Mystikos-SGX environments. As expected, the native environment has the lowest average memory footprint. However, the Gramine-SGX environ-

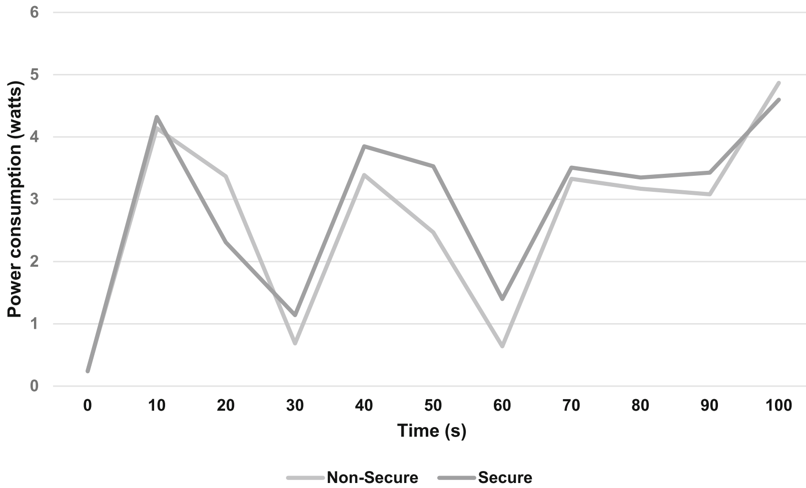

Fig. 7. Power consumption during AlexNet inference in Gramine-SGX.

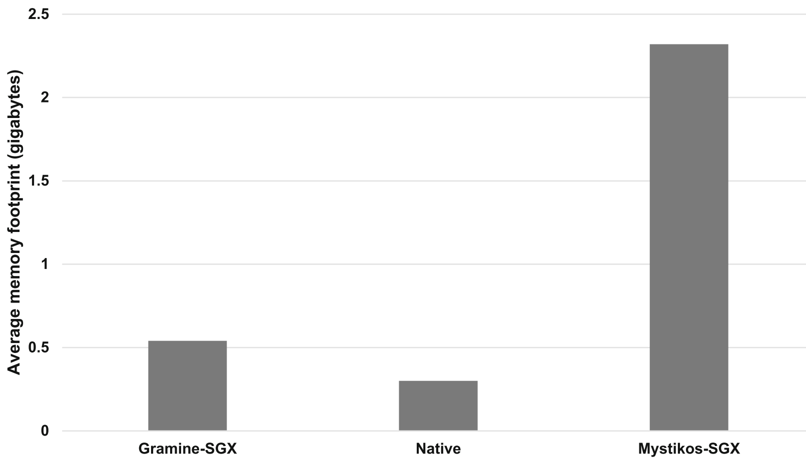

Fig. 8. Average memory footprints in Gramine-SGX, native and Mystikos-SGX.

ment has a footprint that is much closer to the native footprint and significantly lower than the footprint in the Mystikos-SGX environment.

The next set of experiments sought to benchmark the performance times of eight selected deep neural network models during image classification inferencing in the Gramine-SGX environment versus the native environment. The performance time was broken down into inference time, compilation time and total execution time. The inference time was computed as the total execution time minus the compilation time because inference by a deployed deep neural network model does not require any recompilation.

Table 3. Inference performance in native and Gramine-SGX environments.

Model	Native Inference Time	Native Compilation Time	Native Total Execution Time	Gramine-SGX Inference Time	Gramine-SGX Compilation Time	Gramine-SGX Total Execution Time
Squeezenet	0.022 s	0.021 s	0.043 s	0.227 s	75.311 s	75.538 s
Mobilenet V3 Small	0.024 s	0.021 s	0.045 s	0.164 s	76.509 s	76.672 s
Mobilenet V3 Large	0.038 s	0.020 s	0.058 s	0.331 s	77.324 s	77.655 s
ResNet50	0.079 s	0.025 s	0.104 s	1.230 s	77.435 s	78.665 s
ResNet101	0.140 s	0.024 s	0.164 s	2.045 s	78.960 s	81.005 s
AlexNet	0.058 s	0.035 s	0.093 s	2.028 s	82.125 s	84.1535 s
VGG16	0.185 s	0.057 s	0.242 s	4.992 s	84.031 s	89.023 s
VGG19	0.206 s	0.025 s	0.231 s	5.114 s	83.961 s	89.074 s

Table 4. Inference performance in Mystiko-SGX.

Model	Mystiko-SGX Inference Time	Mystiko-SGX Compilation Time	Mystiko-SGX Total Execution Time
Squeezenet	0.517 s	234.154 s	295.096 s
MN V3 Small	0.424 s	237.655 s	293.790 s
MN V3 Large	0.814 s	233.927 s	313.108 s
ResNet50	1.934 s	245.885 s	308.637 s
ResNet101	2.856 s	258.251 s	322.315 s
AlexNet	3.330 s	264.795 s	332.852 s
VGG16	6.816 s	301.268 s	373.483 s
VGG19	7.287 s	291.093 s	368.719 s

Table 3 shows that the model compilation times in the Gramine-SGX environment are significantly greater than the compilation times in the native environment. The inference times are also greater in the Gramine-SGX environment than in the native environment. The results are not unexpected because security always comes with a price.

Another set of experiments were conducted to obtain the inference times, compilation times and total execution times of the eight deep neural network models during image classification inferencing in a Mystiko-SGX environment. The results in Table 4 show that the inference and compilation times for all eight models are significantly higher in the Mystiko-SGX environment than the Gramine-SGX environment. For example, AlexNet model inference in Mystikos-SGX takes 1.3 s longer than in Gramine-SGX. Also, as seen in Fig. 8, its runtime memory footprint is 2.32 GB compared with 0.54 GB for Gramine-SGX. In general, Gramine-SGX is a better trusted execution environment than Mystiko-SGX in that it is less memory intensive and provides more utility and compatibility for applications intended to be ported to Intel SGX.

An additional safeguard would be to implement cache clearance at execution time. This would combat Prime+Probe attack variants that attempt to identify the sets being used by leveraging temporal cache access traces. However, Intel CPUs do not as yet provide an operation for flushing the cache at the user level before exiting an enclave.

7 Conclusions

This research has demonstrated that split computing can be leveraged as a deterrence measure to enhance the confidentiality of deep neural network models ported to Intel SGX environments. The evaluation demonstrates that the approach introduces negligible overhead while securing deep neural network models

in transit and at rest in the hardware enclave. The research also provides useful benchmarking of the available libraries for out-of-the-box porting to Intel SGX trusted execution environments such as Gramine-SGX and Mystikos-SGX.

Acknowledgment. This research was supported by Institute for Infocomm Research, an A*STAR research entity, under the RIE2020 Advanced Manufacturing and Engineering (AME) Program (Award no. A19E3b0099).

References

1. Chen, G., Chen, S., Xiao, Y., Zhang, Y., Lin, Z., Lai, T.: SGXPectre: stealing Intel secrets from SGX enclaves via speculative execution. In: Proceedings of the IEEE European Symposium on Security and Privacy, pp. 142–157 (2019)
2. Choi, H., Bajic, I.: Deep feature compression for collaborative object detection. In: Proceedings of the Twenty-Fifth IEEE International Conference on Image Processing, pp. 3743–3747 (2018)
3. Costan, V., Lebedev, L., Devadas, S.: Secure processors. Part II: Intel SGX security analysis and MIT Sanctum architecture, Foundations and Trends in Electronic Design Automation **11**(3), 249–361 (2017)
4. Deis Labs, Mystikos, GitHub (https://github.com/deislabs/mystikos) (2023)
5. Fei, S., Yan, Z., Ding, W., Xie, H.: Security vulnerabilities of SGX and countermeasures: a survey. ACM Comput. Surv. **54**(6), 126 (2021)
6. Gramine Project Contributors, PyTorch PPML Framework, GitHub (https://github.com/gramineproject/graphene/blob/master/Documentation/tutorials/pytorch/index.rst) 2022
7. Gramine Project Contributors, Gramine Documentation (https://gramine.readthedocs.io/en/latest) (2023)
8. Intel, Understanding Intel Software Guard Extensions (Intel SGX), Santa Clara, California (https://intel.com/content/www/us/en/developer/articles/technical/intel-sgx-and-side-channels.html) (2023)
9. Itahara, S., Nishio, T., Yamamoto, K.: Packet-loss-tolerant split inference for delay-sensitive deep learning in lossy wireless networks. In: Proceedings of the IEEE Global Communications Conference (2021)
10. Jahier Pagliari, D., Chiaro, R., Macii, E., Poncino, M.: CRIME: input-dependent collaborative inference for recurrent neural networks. IEEE Trans. Comput. **70**(10), 1626–1639 (2021)
11. Kang, Y., et al.: Neurosurgeon: collaborative intelligence between the cloud and mobile edge. ACM SIGARCH Comput. Architect. News **45**(1), 615–629 (2017)
12. Large-Scale Data and Systems Group, Spectre attack against SGX enclave, GitHub (https://github.com/lsds/spectre-attack-sgx) (2018)
13. Moghimi, A., Irazoqui, G., Eisenbarth, T.: CacheZoom: how SGX amplifies the power of cache attacks. In: Proceedings of the International Conference on Cryptographic Hardware and Embedded Systems, pp. 69–90 (2017)
14. Narra, K., Lin, Z., Wang, Y., Balasubramaniam, K., Annavaram, M.: Privacy-preserving inference in machine learning services using trusted execution environments. arXiv: 1912.03485 (2019)
15. Pessl, P., Gruss, D., Maurice, C., Schwarz, M., Mangard, S.: DRAMA: exploiting DRAM addressing for cross-CPU attacks. In: Proceedings of the Twenty-Fifth USENIX Security Symposium, pp. 565–581 (2016)

16. PyTorch Team, AlexNet (https://pytorch.org/hub/pytorch_vision_alexnet) (2023)
17. PyTorch Team, ResNet (https://pytorch.org/hub/pytorch_vision_resnet) (2023)
18. PyTorch Team, Source code for torchvision.models.mobilenetv3 (https://pytorch.org/vision/stable/_modules/torchvision/models/mobilenetv3.html) (2023)
19. PyTorch Team, Squeezeent (https://pytorch.org/hub/pytorch_vision_squeezenet) (2023)
20. PyTorch Team, VGG-Nets (https://pytorch.org/hub/pytorch_vision_vgg) (2023)
21. Sanders, J.: Spectre and Meltdown explained: a comprehensive guide for professionals, TechRepublic, May 15 (2019)
22. Schwarz, M., Weiser, M., Gruss, D., Maurice, C., Mangard, S.: Malware guard extension: abusing intel SGX to conceal cache attacks, Cybersecurity, vol. 3, article no. 2 (2020)
23. Seaborn, M., Dullien, T.: Exploiting the DRAM rowhammer bug to gain kernel privileges, presented at Black Hat USA (2016)
24. Tramer, F., Boneh, D.: Slalom: fast, verifiable and private execution of neural networks in trusted hardware. arXiv:1806.03287v2 (2019)
25. Yarom, Y., Falkner, K.: Flush+Reload: a high resolution, low noise, L3 cache side-channel attack. In: Proceedings of the Twenty-Third USENIX Security Symposium, pp. 719–732 (2014)
26. Zhao, C., Saifuding, D., Tian, H., Zhang, Y., Xing, C.: On the performance of Intel SGX. In: Proceedings of the Thirteenth Web Information Systems and Applications Conference, pp. 184–187 (2016)