Privacy preserving layer partitioning for Deep Neural Network models

Kishore Rajasekar
ST Engineering
Singapore

Randolph Loh
ST Engineering
Singapore

Kar Wai Fok
ST Engineering
Singapore

Vrizlynn L. L. Thing
ST Engineering
Singapore

***Abstract*—MLaaS (Machine Learning as a Service) has become popular in the cloud computing domain, allowing users to leverage cloud resources for running private inference of ML models on their data. However, ensuring user input privacy and secure inference execution is essential. One of the approaches to protect data privacy and integrity is to use Trusted Execution Environments (TEEs) by enabling execution of programs in secure hardware enclave. Using TEEs can introduce significant performance overhead due to the additional layers of encryption, decryption, security and integrity checks. This can lead to slower inference times compared to running on unprotected hardware. In our work, we enhance the runtime performance of ML models by introducing layer partitioning technique and offloading computations to GPU. The technique comprises two distinct partitions: one executed within the TEE, and the other carried out using a GPU accelerator. Layer partitioning exposes intermediate feature maps in the clear which can lead to reconstruction attacks to recover the input. We conduct experiments to demonstrate the effectiveness of our approach in protecting against input reconstruction attacks developed using trained conditional Generative Adversarial Network(c-GAN). The evaluation is performed on widely used models such as VGG-16, ResNet-50, and EfficientNetB0, using two datasets: ImageNet for Image classification and TON_IoT dataset for cybersecurity attack detection.**

***Index Terms*—enclave, model partition, private inference, Trusted execution environment, intel sgx, CNN**

I. Introduction

MLaaS (Machine Learning as a Service) has become a popular approach to deploy trained deep learning models, provided by cloud service giants like Microsoft Azure, Amazon AWS, and Google Cloud. Users typically send their data such as images and text, to cloud-based MLaaS platforms for inference tasks. Trusted Execution Environments (TEEs) such as Intel® Software Guard Extensions (Intel SGX) [1] can be used to preserve the confidentiality of user data. Running full inference of the complete trained model in Intel SGX on encrypted user input which makes it invisible for the cloud service provider. However, the performance gap in running inference on accelerators like GPU compared to running within TEE is very high. Hence, we adopt layer partitioning technique where execution of inference of the trained model is split into critical and non-critical partitions. Critical model partition is executed within SGX enclave and non-critical part are the layers offloaded to GPU. While leveraging the GPU can increase computational efficiency, exposing intermediate feature maps in the cloud poses a risk of reconstruction attacks. Our goal is to identify the optimal layer for partitioning and enhance privacy protection. The following are the contributions made by this paper:

1) Analyze the inference runtime performance of three image classification models, namely VGG-16, ResNet-50, and EfficientNetB0 using layer partitioning techniques for Python workloads within the context of TEE.
2) We then measure the efficacy of layer partitioning using trained conditional Generative Adversarial Network (c-GAN) models to evaluate the privacy vs efficiency in the context of the three models. We evaluated ResNet-50 and EfficientNetB0 on two datasets: the ImageNet Kaggle ILSVRC 2012-2017 test dataset [2] and the cybersecurity TON_IoT dataset [3].
3) Additionally, we examine whether the choice of dataset influences the reconstruct-ability of input images and also determine if the speedups vary for different models after identifying optimal partitioning points.

II. Related Works

Different methods can be employed to safeguard data privacy, including Homomorphic Encryption libraries [4], [5], Secure Multi-Party Computing [6] , Differential Privacy [7], and the utilization of TEEs. Each approach offers distinct levels of privacy protection and incurs varying costs [8].

In the case of TEE-based approaches, the TEE-shielding approach runs the complete unmodified model inside enclaves, ensuring both model confidentiality and high accuracy comparable to the original model. The partition-based approach involves manually selecting sensitive model layers to execute within an enclave, while allocating the remaining layers to an untrusted GPU for acceleration. This strategy results in reduced inference latency compared to TEE-shielding approaches. Some prior works include eNNclave [9] and AegisDNN [10]. eNNclave replaces partitioned operators' parameters with pre-trained parameters from other publicly available models, which can lead to a loss in inference accuracy. AegisDNN uses dynamic programming to identify partitioning point to learn each layer's criticality, and partitions uncritical (plaintext) layers to GPU to meet the user argument deadline.

Slalom [11] provides an inference framework that uses TEE-GPU collaboration to protect data privacy and integrity. It offloads computational intensive convolutions to GPU, and preserves data privacy of the offloaded computations using

979-8-3503-5409-6/24/$31.00 ©2024 IEEE
DOI 10.1109/CAI59869.2024.00202

cryptographic blinding technique. Origami inference [12] uses a combination of model partitioning, computational offloading and eliminates data blinding in second tier of inference to provide fast and private inference by protecting input privacy against a trained c-GAN adversary [13].

We aim to evaluate our layer partitioning approach on three models. To protect exposed intermediate feature maps from input reconstruction attacks, we adopt the method outlined in [12]. This involves performing reconstruction attacks using c-GAN models at each partition point. We use the trained model with its original parameters and execute non-critical layers of the model in plaintext on GPU. Existing approaches depend on the Intel SGX SDK and lack direct support for Python deep learning frameworks like PyTorch or TensorFlow. The process of porting Python applications to SGX enclave involves overcoming technical hurdles related to memory constraints, compatibility, security, and performance optimization. Porting Python applications to SGX enclave faces challenges due to: (i) complex dependencies among modules, making it difficult to ensure integrity; (ii) spawning new processes, requiring separate handling of file access permissions [14]. Gramine [15], [16], on the other hand, enables the execution of unmodified applications within SGX enclaves, eliminating the need for manual porting.

III. Research Problem

A. Intel SGX

Intel SGX a technology developed by Intel that provides a secure hardware enclave for protecting sensitive data and computations. By running trained models within an SGX enclave, user data can be encrypted, decrypted, and processed securely, protecting it from potential threats in untrusted cloud environments. Use of Intel SGX can enhance the privacy and security of machine learning services, enabling users to safely utilize cloud-based inference while preserving the confidentiality of their data.

B. Model Partitioning

Layer partitioning is a technique that allows for the efficient execution of machine learning model inference by dividing the computational workload between an SGX enclave and a GPU. When running the entire model inference within an SGX enclave, limitations such as memory constraints and performance overhead can arise, especially for large models. To address these challenges, layer partitioning divides the model into two partitions. The first partition consists of the initial layers of the model, which are executed within the SGX enclave. These layers typically contain the majority of the sensitive information that could potentially lead to input reconstruction attacks. User input privacy evaluation is done based on the insight that only the first few layers of the model contain most of the information required for input reconstruction compared to the output from deeper layers of the model. Within the SGX enclave, the input data is decrypted and processed. The intermediate feature maps resulting from this computation are then offloaded to an untrusted GPU for further processing. The second partition, comprising the remaining layers of the model, can be executed on the GPU for faster execution. This approach optimizes the utilization of resources by leveraging the security of SGX for the critical layers while taking advantage of the computational power of the GPU for the non-critical layers.

C. Threat model

We define the adversary as an agent that tries to use observed intermediate feature maps of model to reconstruct input in the untrusted cloud environment. The term 'untrusted cloud' refers to cloud environments where the presence of third-party adversaries poses a risk of unauthorized access or observation, potentially enabling them to intercept or monitor sensitive data. We use trained c-GAN adversary models for the reconstruction attack evaluation, as outlined in Origami Inference [12], to assess the reconstruct-ability of input images. The architecture of the c-GAN model contains a Generator and Discriminator. The generator follows an encoder-decoder architecture with two residual blocks. The discriminator architecture consists of two parts: a down-sampler and a series of convolutional blocks. During training, the adversarial BCELoss loss is used to measure the difference between predicted and target labels. The training is done for 200 epochs. The reconstructed images are of dimension $3 \times 224 \times 244$.

IV. Proposed Solution Framework

The framework of our solution is shown in Fig. 1. The steps for performing private inference using our solution are as follows:

1) The user aims to utilize resources in an untrusted cloud for running their model and performing model inference on input data, all while ensuring the privacy of both the model and the data.
2) The model and data are decrypted within the secure and private TEE which is hosted on the cloud. The information within the TEE cannot be exposed to the untrusted cloud environment.
3) The model is split into critical and non-critical partitions within the TEE, based on the architecture of the CNN model and the optimal partitioning point.
4) The execution of the critical model partition on the input data is performed within the secure TEE.
5) The output of the critical model partition which is saved and sent out into the untrusted cloud for further processing.
6) The intermediate feature maps and non-critical model partition can be loaded and executed in a cloud-based GPU for optimized runtime performance.

In this work, we aim to explore the performance improvements and trade-off using model partitioning technique. This section consists of four parts: Neural network models, Datasets, Runtime performance evaluation and Privacy evaluation.

A. Neural Network Models

1) VGG-16: The architecture of VGG-16 [17] is shown in Fig. 2a. The model consists of 16 layers in total, which

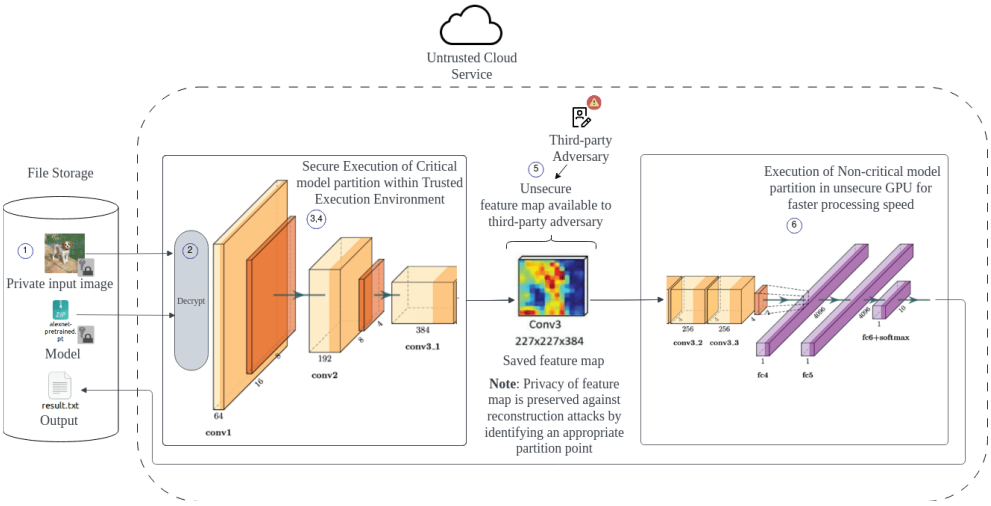


Fig. 1: Secure system framework to perform private inference

includes 13 convolutional layers and 3 fully connected (FC) layers.

2) ResNet-50: ResNet-50 is a CNN architecture [18] composed of 50 layers, with its 16 residual blocks divided into four stages comprising 3, 4, 6 and 3 blocks as shown in Fig. 2b. Skip connections between blocks are used to mitigate the vanishing gradient problem.

3) EfficientNetB0: The EfficientNetB0 architecture [19], as illustrated in Fig. 2c, is composed of 16 Mobile Inverted Bottleneck Convolution (MBConv) layers divided into seven stages composed of varying number of layers and one FC layer.

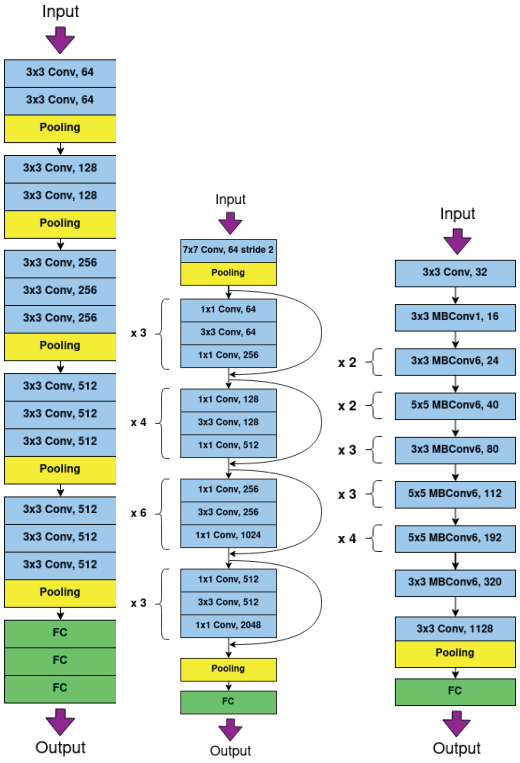
B. Datasets

1) ImageNet: The ImageNet ILSVRC dataset [2] is a widely used collection of images belonging to 1000 classes for image classification tasks.

2) TON_IoT Dataset converted to Images: In 2019, TON_IoT Dataset [3] was created based on a testbed environment at the Cyber Range and IoT Labs at the University of New South Wales (UNSW) Canberra, Australia. The TON_IoT dataset is converted to images by a processing method proposed in [20] converting the time-based one-dimensional data in the original dataset [3] into tensors that are accepted by CNNs. The total number of classes are 8 attack types. For clarity, we will refer to the TON_IoT Dataset converted to images as the ”TON_IoT image dataset”.

C. Runtime Performance Evaluation

In this study, we compared the runtimes of three models when offloading different layers to the GPU. The partitioning approach is used to help strike a balance between privacy preservation and computational efficiency, utilizing the functions of both Intel SGX and GPU acceleration. Gramine cannot directly access to the GPU from within the enclave. To address this limitation, we create a separate PyTorch process outside



(a) VGG-16 (b) ResNet-50 (c) EfficientNetB0

Fig. 2: Model architectures

the enclave. This external process is responsible for offloading computations to the GPU. The kernel switch occurs only once during the offloading process and is measured as the time taken to save the feature map and load it onto the GPU. This operation typically takes between 0.02 to 0.1 seconds depending on the feature map size.

1) Experimental Setup: Our experiments measure the runtime for inference of the three aforementioned models offloading the intermediate feature maps at different partition points. The inference within the TEE was done using Gramine library to run secure model inference within Intel SGX. The dataset used was a subset of 100 samples from the Imagenet dataset to measure the average inference runtimes of models partitioned at each layer. The runtimes measured are in seconds. The baseline used for each of the models is its runtime performance in Full-Enclave setting. The code to run inference, for critical layers in TEE and non-critical layers in GPU, was written in Python 3.8. We conducted our evaluations on a desktop machine comprising an Intel® Core™ i7-9700K CPU with SGX capability, 8 threads and 64 GB of memory. We used an NVIDIA GeForce RTX 2070 Ti GPU as the accelerator. The operating system used is Ubuntu 22.04.2 LTS.

2) Analysis:

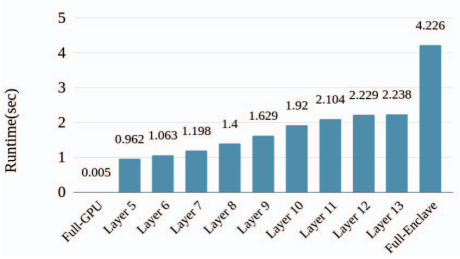
a) VGG-16: In Fig. 3a, we present the average inference runtime of the VGG-16 model for different layer partitions. The x-axis represents the partitioning points. For instance, the x-axis label Layer 5 denotes the partitioning point at the 5th convolutional layer of the model from the input side which corresponds to the `3x3 Conv, 256` in VGG-16 architecture shown in Fig. 2a. The partitioning is done after the convolutional layer in each instance. In Fig. 3a, it can be observed that there is a steady increase in runtime with the increase in the number of layers executed in SGX enclave.

b) ResNet-50: In the case of ResNet-50, we observe the same trend of increased inference runtime when the model is partitioned at later layers, resulting in offloading execution of fewer layers to the GPU, as illustrated in Fig. 3b. The four stages in ResNet-50 contain various number of residual blocks, and each residual block contains 3 convolutional layers. For instance, the first stage contains 3 residual blocks, second stage contains 4 residual blocks. The partition is done at the end of each stage. Table I shows the number of convolutional layers executed within the SGX enclave and GPU at different layer partitioning points.

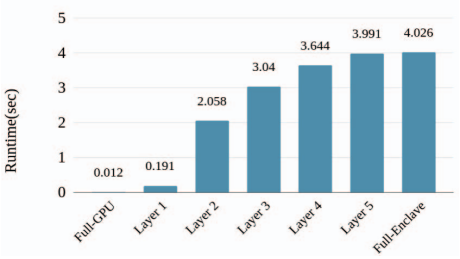
TABLE I: ResNet-50 Layer partitions

Environment	*Layer 1*	*Layer 2*	*Layer 3*	*Layer 4*	*Layer 5*
SGX enclave	1 conv layer	10 conv layers	22 conv layers	40 conv layers	49 conv layers
GPU	48 conv + 1 FC layers	39 conv + 1 FC layers	27 conv + 1 FC layers	9 conv + 1 FC layers	1 FC layer

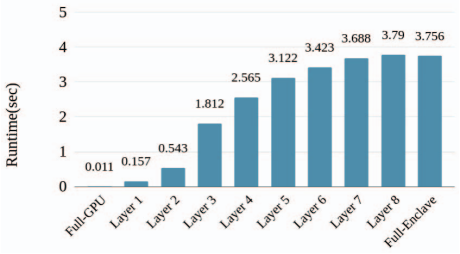
c) EfficientNetB0: The runtime performance of inference for each of these partition points shown in Fig. 3c indicates a steady upward trend as the number of layers executed in GPU decreases. Table II shows the number of MBConv blocks executed within the SGX enclave and GPU at different layer partitioning points.



(a) VGG-16



(b) ResNet-50



(c) EfficientNetB0

Fig. 3: Average inference runtime of DNN models for different layer partitions.

D. Privacy Evaluation: Measuring reconstruct-ability of intermediate feature maps

After analyzing the runtime performance, we evaluate the privacy of the offloaded feature maps at different partition points determining the optimal layer partition for each model that balances speed-up and privacy. This is done by assessing the degree to which input images can be reconstructed from intermediate feature maps. Offloading more layers to the GPU speeds up inference but increases the risk of compromising input privacy. Striking the right balance is crucial to ensure privacy is not compromised. In order to evaluate the privacy, we adopt the use of a c-GAN adversary model to reconstruct input images from the intermediate feature maps from different layer partitions. We evaluate all three models on ImageNet, and two models ResNet-50, EfficientB0 (top performing models based on accuracy as per [20]) on the TON_IoT image dataset. These models were chosen to represent different architectural characteristics and complexity levels.

TABLE II: EfficientNetB0 Layer partitions

Environment	*Layer 1*	*Layer 2*	*Layer 3*	*Layer 4*	*Layer 5*	*Layer 6*	*Layer 7*	*Layer 8*
SGX enclave	1 conv layer	1 conv + 1 MBConv layers	1 conv + 3 MBConv layers	1 conv + 5 MBConv layers	1 conv + 8 MBConv layers	1 conv + 11 MBConv layers	1 conv + 15 MBConv layers	1 conv + 16 MBConv + 1 conv layers
GPU	16 MBConv + 1 conv + 1 FC layers	15 MBConv + 1 conv + 1 FC layers	13 MBConv + 1 conv + 1 FC layers	11 MBConv + 1 conv + 1 FC layers	8 MBConv + 1 conv + 1 FC layers	5 MBConv + 1 conv + 1 FC layers	1 MBConv + 1 conv + 1 FC layers	1 FC Layer

1) Experimental Setup: To reconstruct the input images, we trained c-GAN models that takes the feature maps as input and generates corresponding image samples. By training the c-GAN on a dataset with 1000 images and their corresponding intermediate feature maps for 200 epochs, we aimed to achieve realistic and accurate image reconstructions. We used Python 3.8 for all our experiments to train c-GAN models and perform model partitioning using PyTorch deep learning framework, version 1.9.1. The training was conducted on the same desktop machine with specifications identical to those mentioned in Section IV-C1. We quantitatively evaluated the privacy of the models by computing the Structural Similarity Index Measure (SSIM) scores between the original images and their reconstructed versions from the feature maps of different layers. The SSIM metric provides a measure of structural similarity and perceptual quality between two images. It ranges from 0 to 1. A higher SSIM value indicates a high similarity, in terms of structural information, between the original image and the reconstructed image. We selected the threshold SSIM score of 0.2 to achieve the goal of safeguarding input privacy considering the perceptual quality of the reconstructed image. Subsequently, we determined the optimal layer for partitioning to be the point at which the SSIM score falls below this threshold and consistently remains low thereafter.

2) Analysis of models on ImageNet dataset: The summary results can be seen in Table III.

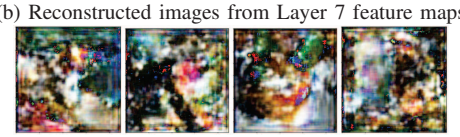
TABLE III: Summary analysis results of model speedups on ImageNet dataset

DNN Model	Total Partition Points	Optimal Partitioning Point	Full-Enclave Inference Runtime (Avg)	Partitioned Inference Runtime (Avg)	Performance Speedup (%)
VGG-16	13	Layer 8	4.2 sec	1.4 sec	66.6%
ResNet-50	5	Layer 4	4.02 sec	3.6 sec	10.4%
EfficientNetB0	8	Layer 4	3.7 sec	2.5 sec	32.4%

a) VGG-16: Fig. 7a depicts the SSIM metric values for the similarity between the original and the reconstructed images obtained form c-GAN model Generator using the feature maps obtained from the respective layers. It can be observed that there is a steady drop in the SSIM scores indicating that it becomes progressively more challenging to reconstruct the original images from the feature maps of the deeper layers. The SSIM score after Layer 7 remains below 0.2 and the reconstructed images can be seen in Fig. 4 for further visual inspection. We can visually observe that the quality of reconstructed images from Layer 2 feature maps is superior to those from Layer 7 and Layer 8, indicating that it becomes increasingly challenging to reconstruct input images from deeper layers. Layers with higher spatial resolution feature maps (e.g., early convolutional layers) are typically more vulnerable to reconstruction attacks. This vulnerability arises because the spatial information retained in these layers makes it easier to reconstruct the original input. As the network progresses deeper into the architecture, we observe a decrease in spatial resolution and an increase in the abstraction of feature maps with the presence of convolutional and pooling layers. These deeper layers contain less spatial and fine-grained information about the input images, making them less susceptible to reconstruction attacks. This trend is reflected in the overall decrease in SSIM scores for reconstructed images. The SSIM score stays below 0.2 after Layer 8, indicating that

(a) Reconstructed images from Layer 2 feature maps

(b) Reconstructed images from Layer 7 feature maps



(c) Reconstructed images from Layer 8 feature maps

Fig. 4: Reconstructed images from intermediate feature maps of different layer partitions in VGG-16

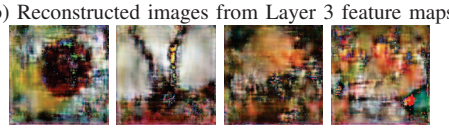
Layer 8 is the optimal partitioning point. This decision is based on both the SSIM score metric and visual inspection of the reconstructed images. The average inference runtime is 1.4 seconds partitioned at Layer 8 compared to 4.2 seconds for Full-Enclave execution as shown in Fig. 3a. Hence, the speedup observed is 66.6% compared to Full-Enclave inference. Prior works [11] and [12], they both achieve a speedup of 10.1x to 12.7x for VGG-16, utilizing the Eigen library, a high-performance C++ based linear algebra library that enables outsourcing linear layers to hardware GPU. Due to limitations in running unmodified PyTorch applications within Intel SGX, Gramine is currently unable to directly utilize the GPU. However, this functionality is planned for future releases,

potentially resulting in higher speed-ups [21]. We aimed to assess Python PyTorch workloads for a fair comparison. Therefore, we present the results comparing the speedup against Full-Enclave inference using the Gramine library. As far as we know, our work is among the first to evaluate the performance of Python PyTorch application workloads for privacy-preserving inference utilizing both TEE and GPU acceleration.

b) ResNet-50: ResNet-50 also follows the similar trend where it gets progressively harder to reconstruct images from feature maps of deeper layers. Here the SSIM score remains below 0.2 past Layer 3 as depicted in Fig. 7b. Hence, Layer 4 is identified as the optimal layer to partition for ResNet-50 model inference based on SSIM score and visual inspection of Fig. 5 for the reconstructed images of ImageNet dataset. The

(a) Original images

(b) Reconstructed images from Layer 3 feature maps



(c) Reconstructed images from Layer 4 feature maps

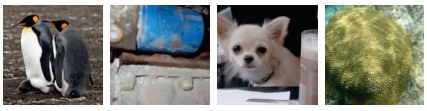
Fig. 5: Reconstructed images from intermediate feature maps of different layer partitions in ResNet-50.

average inference runtime is 3.6 seconds for model partitioned at Layer 4 compared to 4.02 seconds for Full-Enclave execution. Resulting in a speedup of 10.4% compared to Full-Enclave inference.

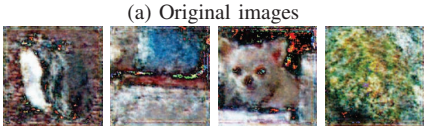
c) EfficientNetB0: In the Fig. 7c, it can be observed that there is a steady drop in the SSIM scores until Layer 4 and slight increase in the scores until Layer 6 of EfficientNetB0. However, it hovers around 0.2, which shows limited effectiveness of the reconstructed images from deeper layers. The reconstructed images can be seen in Fig. 6 for further visual inspection. The SSIM score from Layer 4 onwards remains around or below 0.2. Hence the optimal layer to partition will be Layer 4. The average runtime for inference partitioning at Layer 4 of EfficientNetB0 is 2.5 seconds compared to 3.7 seconds for Full-Enclave execution. Hence, the speedup observed is 32.4% compared to Full-Enclave inference.

3) Analysis of models on TON_IoT image dataset:

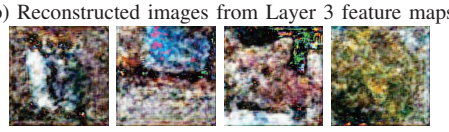
a) ResNet-50: The SSIM scores, in Fig. 8a show a decreasing trend until Layer 4, followed by an increase at Layer 5 when evaluated on the TON_IoT image dataset. It was observed that the reconstructions contain high noise from Layer 3 onwards. Hence, the optimal layer to partition the model for use with TON_IoT image dataset is Layer 3 whereas



(a) Original images

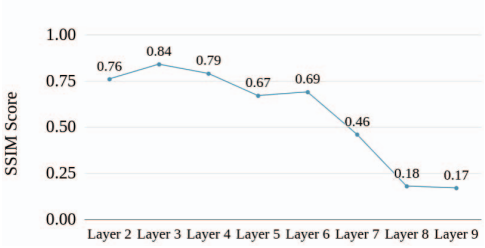


(b) Reconstructed images from Layer 3 feature maps

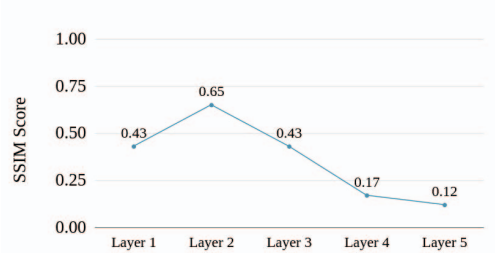


(c) Reconstructed images from Layer 4 feature maps

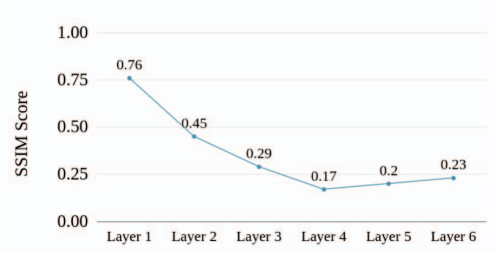
Fig. 6: Reconstructed images from intermediate feature maps of different layer partitions in EfficientB0.



(a) VGG-16



(b) ResNet-50

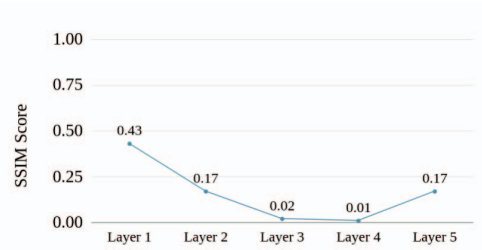


(c) EfficientNetB0

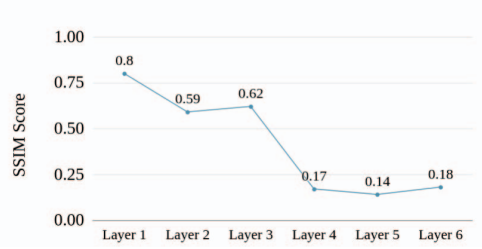
Fig. 7: SSIM scores for reconstruction for different layer partitions of DNN models evaluated on ImageNet dataset.

the optimal partition point for evaluation on ImageNet was identified to be Layer 4. The average inference runtime is 3.04 seconds for model partitioned at Layer 3 compared 4.02 seconds in Full-Enclave setting, resulting in a speedup of 24.3% compared to Full-Enclave inference.

b) EfficientNetB0: SSIM scores measured for the layers of EfficientNetB0 model for TON_IoT image data can be seen in Fig. 8b. The SSIM score for Layer 1, Layer 2 and Layer 3 was observed to be higher than that of SSIM scores for reconstruction based on the ImageNet dataset. However, the optimal layer to partition remains the same as Layer 4 as in the case with the ImageNet dataset where the average runtime is 2.5 seconds compared to 3.7 seconds for Full-Enclave execution. Hence, the speedup observed is 32.4% compared to Full-Enclave inference.



(a) ResNet-50



(b) EfficientNetB0

Fig. 8: SSIM scores for reconstruction for different layer partitions of ResNet-50 and EfficientNetB0 evaluated on TON_IoT dataset.

V. Conclusion

Overall, our study showcases the potential of leveraging Intel SGX and GPU acceleration for privacy-preserving inference with deep learning models such as VGG-16, ResNet-50 and EfficientNetB0. We have conducted runtime performance analysis and assessed input privacy by measuring the input reconstruct-ability using trained c-GAN adversary models for each layer partition. The speedup rates differed among models, depending on their architecture and the choice of the optimal partition point. The majority of ML solutions and libraries are written in Python. To facilitate library efficient adoption, hence, we leverage Gramine as a Python package with seamless support for both Tensorflow and PyTorch. While Gramine allows running whole applications unmodified, it incurs high runtime costs for full execution of inference in SGX enclave. To address this, we use layer partitioning approach that improves performance while executing inference in TEE for critical layers and offloading rest of the computation for non-critical layers to a co-located GPU. As a future work, we plan to explore additional optimizations in performance for computationally intensive convolutional layers in the critical model partition to further improve the inference runtime for models.

References

[1] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution." *Hasp@ isca*, vol. 10, no. 1, 2013.

[2] W. K. Addison Howard, Eunbyung Park, "Imagenet object localization challenge," 2018. [Online]. Available: https://kaggle.com/competitions/imagenet-object-localization-challenge

[3] A. Alsaedi, N. Moustafa, Z. Tari, A. Mahmood, and A. Anwar, "Ton_iot telemetry dataset: A new generation dataset of iot and iiot for data-driven intrusion detection systems," *Ieee Access*, vol. 8, pp. 165 130–165 150, 2020.

[4] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International conference on machine learning*. PMLR, 2016, pp. 201–210.

[5] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference system for neural networks," in *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*, 2020, pp. 27–30.

[6] S. Wagh, D. Gupta, and N. Chandran, "Securenn: 3-party secure computation for neural network training." *Proc. Priv. Enhancing Technol.*, vol. 2019, no. 3, pp. 26–49, 2019.

[7] A. Team *et al.*, "Learning with privacy at scale," *Apple Mach. Learn. J*, vol. 1, no. 8, pp. 1–25, 2017.

[8] F. Mireshghallah, M. Taram, P. Vepakomma, A. Singh, R. Raskar, and H. Esmaeilzadeh, "Privacy in deep learning: A survey," *arXiv preprint arXiv:2004.12254*, 2020.

[9] A. Schlögl and R. Böhme, "ennclave: offline inference with model confidentiality," in *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*, 2020, pp. 93–104.

[10] Y. Xiang, Y. Wang, H. Choi, M. Karimi, and H. Kim, "Aegisdnn: Dependable and timely execution of dnn tasks with sgx," in *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021, pp. 68–81.

[11] F. Tramr and D. Boneh, "Slalom: Fast verifiable and private execution of neural networks in trusted hardwarem," *ICLR 2019*, 2019.

[12] K. G. Narra, Z. Lin, Y. Wang, K. Balasubramanian, and M. Annavaram, "Origami inference: private inference using hardware enclaves," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 2021, pp. 78–84.

[13] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[14] D. Zhang, G. Wang, W. Xu, and K. Gao, "Sgxpy: Protecting integrity of python applications with intel sgx," in *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, 2019, pp. 418–425.

[15] C.-C. Tsai, D. E. Porter, and M. Vij, "{Graphene-SGX}: A practical library {OS} for unmodified applications on {SGX}," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 645–658.

[16] "Gramine library," 2021. [Online]. Available: https://github.com/gramineproject/gramine

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

[18] 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.

[19] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[20] M. Kodyš, Z. Lu, K. W. Fok, and V. L. Thing, "Intrusion detection in internet of things using convolutional neural networks," in *2021 18th International Conference on Privacy, Security and Trust (PST)*. IEEE, 2021, pp. 1–10.

[21] D. Kuvaiskii, G. Kumar, and M. Vij, "Computation offloading to hardware accelerators in intel sgx and gramine library os," 2022.