MazeNet: Protecting DNN Models on Public Cloud Platforms With TEEs

Kripa Shanker[1(✉)], Vivek Kumar[1], Aditya Kanade[2], and Vinod Ganapathy[1]

[1] Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India
{kripashanker,vg}@iisc.ac.in, vivekkumar13@alum.iisc.ac.in
[2] Microsoft Research India,Bangalore, India
kanadeaditya@microsoft.com

Abstract. Machine Learning-as-a-Service (MLaaS) enables deep learning (DL) model owners to outsource inference tasks to a public cloud platform. A model owner trains a DL model in-house and uploads the trained model to a MLaaS. For the uploaded model, the MLaaS platform exposes an API to query the uploaded model with inputs and obtain predictions. However, uploading the trained model to public cloud platforms exposes the model owner to security and privacy risks, as the model is available in plaintext to the cloud provider during inference.

In this work, we present techniques to secure DL models with trusted execution environments and propose a secure outsourcing scheme to offload portions of the DL model computations during inference to faster untrusted processors. We implement the presented techniques in MazeNet, a framework to transform pretrained models into MazeNet models and deploy them on a public cloud platform to provide inference services.

We evaluate MazeNet on popular convolutional neural networks, and the results demonstrate that MazeNet improves the performance of DNN models as compared to a secure baseline model, where the model runs within a trusted environment. MazeNet increases the throughput of the inference task up to 30x and decreases the latency up to 5x for the benchmark models in our experimental evaluation.

Keywords: Privacy · Intel SGX · Machine learning

1 Introduction

Machine Learning-as-a-Service (MLaaS) enables deep learning (DL) model owners to deploy trained models on public cloud platforms to provide inference services. A model owner pretrains a model in-house and uploads the trained model to a public cloud platform. In turn, MLaaS exposes an API endpoint to the uploaded model to query the model with inputs and obtain predictions.

However, moving the model to a public cloud platform exposes the model owner and the uploaded model to various security and privacy risks. First, a

V. Kumar and A. Kanade—Work done while at the Indian Institute of Science, Bangalore, India.

N. Hubballi et al. (Eds.): ICISS 2025, LNCS 16380, pp. 1–20, 2025.
https://doi.org/10.1007/978-3-032-13714-2_6

compromised or malicious MLaaS provider can easily steal the deployed models as the model is available in plaintext during the inference because the cloud vendor controls the entire hardware and software stack at its data centres. A stolen model leads to financial losses and legal troubles for the model owner. Often, organisations invest significant financial resources to train state-of-the-art models [21] that solve critical business problems. Second, having access to model parameters and intermediate states, the DL model can leak sensitive information about their private training dataset through membership inference attacks [44]. Often, these models are trained on private datasets to improve the accuracy of the learning task. Legal laws in many countries require that private data, such as financial transactions [1], electronic health records [2], insurance, etc., be protected. Otherwise, the defaulting organisation will be penalised heavily [1,2]. Third, the MLaaS provider can tamper with the uploaded model to influence the results. Therefore, it becomes crucial to protect the integrity and confidentiality of the uploaded models on public cloud platforms.

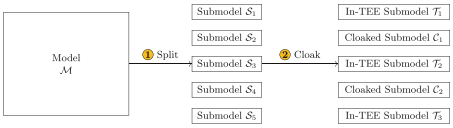
A naive solution to protect the model on an MLaaS platform would be to run the model within a Trusted Execution Environment (TEE) such as Intel Software Guard Extensions (SGX). The TEE will be responsible for ensuring the confidentiality and integrity of the DL model during the inference process. However, directly running the DL model within the TEE is suboptimal.

First, TEEs often have fixed and usually smaller protected memory than the main memory. For example, SGXv1 offers only 128 MB of protected memory. Therefore, a DL model larger than protected memory incurs a performance penalty. Second, TEEs cannot securely utilise untrusted but powerful resources such as co-located processors, main memory, and storage available on the system.

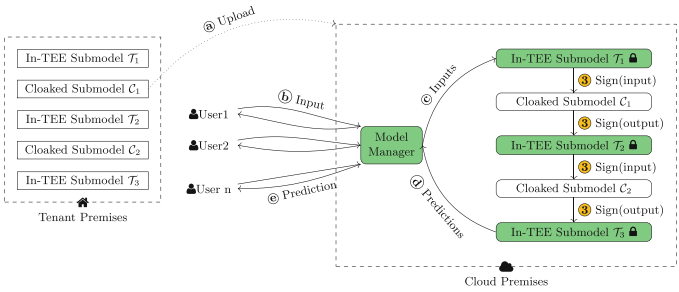
In this work, we present MazeNet, a framework to transform pre-trained models into MazeNet models and securely run them on heterogeneous and distributed systems consisting of trusted execution environments and untrusted runtimes. MazeNet uses three key techniques to overcome the limitations of TEEs and provides a secure and fast inference solution.

First, to overcome the fixed protected memory of SGX enclaves, MazeNet splits the DNN model into smaller models, referred to as submodels, such that the maximum memory usage of each submodel during inference is less than the size of the protected memory offered by the SGX enclaves. During inference, MazeNet distributes the submodels to different SGX enclaves. Splitting avoids the performance penalty due to swapping when submodels execute within SGX enclaves. Figure 1a shows an illustration of splitting, where a model $\mathcal{M}$ is split ① into five submodels $\mathcal{S}_1 \ldots \mathcal{S}_5$.

Second, running submodels only within TEEs leaves other untrusted resources underutilised. Therefore, to maximise the system utilisation and improve the performance, a subset of submodels is outsourced to untrusted runtime environments. However, outsourcing to an untrusted environment raises privacy risks for the outsourced submodels. Therefore, MazeNet employs its second technique, cloaking, a secure outsourcing scheme to offload submodel evaluation to untrusted hardware. In cloaking, synthetic layers and neurons are added to



(a) On-premises, a Model Builder splits ① the pre-trained model and cloaks ② a subset of submodels to produce a MazeNet model.



(b) On the cloud, a Model Manager deploys the MazeNet model, and exposes an API for users to query the model. During the inference, the MLaaS provider signs inputs and outputs of cloaked submodels ③ that are outsourced to untrusted environments.

Fig. 1. End-to-end workflow of MazeNet system.

the submodel, which hides the original weights within the synthetic weights to protect the privacy of the outsourced submodel. In the Fig. 1a, after splitting, submodel $\mathcal{S}_2, \mathcal{S}_4$ are cloaked ② to produce final MazeNet Model.

Third, during inference, the adversary can tamper with any data or computation outside the TEEs, including cloaked submodel weights. MazeNet employs digital signatures to detect tampering in offloaded computations. It requires the cloud vendor to sign ③ both the inputs and the outputs of the cloaked submodels, as shown in Fig. 1b, to commit to submodel evaluation results. The signatures are later used during an audit phase, where MazeNet independently verifies some of the outsourced computations by re-executing some of them.

Figure 1 shows the end-to-end MazeNet workflow. First, a cloud tenant transforms a pretrained model into a MazeNet model with *Model Builder* and uploads the generated model to a public cloud platform, where a *Model Manager* securely deploys the uploaded model and exposes an API to query the model.

Prior works [16,46,50] have proposed secure outsourcing schemes to outsource linear layers of DL models to hardware accelerators. However, outsourcing only linear layers is suboptimal, as linear layers are frequently followed by non-linear layers. As a result, the intermediate state needs to be constantly moved between the TEE and the GPUs. Table 1 shows the data transferred between TEE and non-TEE for prior works. MazeNet overcomes this limitation by outsourcing both the linear and the non-linear layers to GPUs, thereby reducing communication costs. MazeNet can reduce up to 90% of the data transfer cost. Figure 2 compares MazeNet with prior works.

Table 1. Data transfers between the TEE and GPU when only linear layers are outsourced to GPUs, while non-linear layer executes within the TEE.

Model	Data transfer (MB)		
	TEE→GPU	GPU→TEE	Total
VGG16	34.77	51.71	86.48
ResNet50	38.70	40.39	79.09
DenseNet201	91.43	29.96	121.39

To evaluate the benefits and costs of the proposed techniques, we have built a prototype of the MazeNet framework on top of TensorFlow [14]. We transform popular convolutional neural networks (CNN), VGG16 [45], ResNet50 [17], and DenseNet201 [18], into MazeNet models and compare them against a secure baseline model, where the whole unmodified model runs within a TEE. Our evaluation shows that MazeNet can improve the throughput up to 30x and reduce the latency up to 5x for the models considered in our experimental evaluation.

To summarise, the following are the main contributions of this work:

- This work presents MazeNet, a framework to secure trained deep learning models on public cloud with trusted execution environments.
- It proposes an outsourcing scheme to outsource both linear and non-linear layers to untrusted environments to accelerate the model inference.
- Our evaluation shows that MazeNet can significantly improve the throughput and the latency of DL models as compared to the secure baseline models.

2 Background

Trusted Execution Environments. Trusted Execution Environments (TEEs) provide an isolated environment that is protected against software and hardware-based attacks from privileged adversaries, including the OS, BIOS, and firmware. Silicon vendors have introduced support for TEEs [11,28,31,32]. This work uses Intel SGX as a TEE because its threat model is most suitable for cloud workloads, as it has a minimal trusted computing base. With SGX, applications can build enclaves within their program address space, whose contents are protected from privileged adversaries.

Threat Model. MazeNet admits a strong adversary based on the standard threat model of Intel SGX. We only trust the code and data residing in enclaves and the Intel CPU. The adversary controls the privileged system software, including the OS, firmware, and BIOS. It can read and modify data located outside SGX enclaves. The main goal of this work is to protect the privacy of trained parameters of a deep learning model. We assume that the cloud vendor is not aware of the pretrained model architecture. This work does not aim to protect the privacy of user inputs, as it is orthogonal to the goal of model privacy.

Prior research has demonstrated several side-channel attacks against SGX [7, 52], and Intel is actively working to fix those side-channel attacks [20]. Therefore, side-channel attacks are outside the scope of this work.

Prior Work	Model Privacy (server)	Input Privacy (server)	Outsource to Untrusted Hardware	Scalable to Large Models	Platform / Technique
Securenets [8]	✓	✓	✓		SMM
Cryptonets [13]					FHE
Chameleon [38]		✓			GC
DeepSecure [39]	✓	✓			GC
Crypflow [25]	✓			✓	MPC
SecureML [35]		✓			MPC
MiniONN [29]		✓			FHE and GC
Gazelle [23]	✓				FHE and MPC
Shadownet [46]	✓		✓		ARM TrustZone
DarknetTZ [34]				✓	ARM TrustZone
OMG [5]	✓				ARM TrustZone
Occlumency [27]		✓			SGX
TensorScone [26]	✓				SGX
DarkNight [16]	✓		✓		SGX
MLCapsule [15]	✓	✓			SGX
Slalom [50]			✓		SGX
MazeNet	✓		✓	✓	SGX

Fig. 2. Feature comparison of prior works with respect to the privacy of models and user inputs. SMM: Secure Matrix Multiplication, SGX: Intel Software Guard eXention. FHE: Fully Homomorphic Encryption. MPC: Multi-Party Compuation. GC: Garbled Circuits.

3 Building MazeNet Models

A pre-trained model is transformed in two stages. First, the model is split into smaller models. Then, a subset of models is cloaked to produce a MazeNet model.

3.1 Splitting DNN Models

The size of DNN models is increasing as new state-of-the-art architectures are introduced, and larger models are trained to achieve higher accuracy on the learning task. Recent report [4] states that computing resources needed for DNNs doubles every 3.4 months, with GPT-3 model reaching 175 billion parameters. Due to their large size, many DNN models do not fit within the fixed protected memory offered by TEEs. In case of SGX enclaves, the hardware can only cryptographically protect a small portion of main memory [16,49]. Models larger than the size of protected memory incur a performance penalty due to swapping.

Intel SGX reserves a portion of main memory, Enclave Page Cache (EPC), to store the encrypted pages of the enclaves in the main memory, which are protected by the hardware. As this is a fixed portion of memory, enclave applications and DNN models that have higher memory requirements than the size of the protected memory incur EPC swapping, where the SGX driver in the kernel

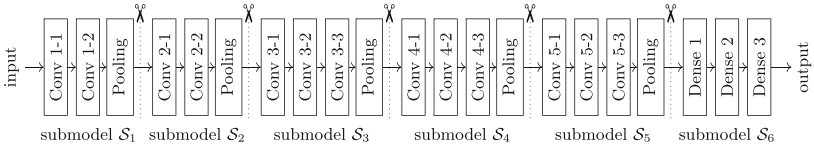


Fig. 3. Splitting the VGG16 [45] model into smaller submodels.

seals a few pages of EPC and stores them on the unprotected memory. Thus freeing a few pages in the EPC for enclave applications. Due to EPC swapping, the performance of enclave applications is severely degraded, as swapping is computationally expensive due to the cryptographic operations required to ensure confidentiality and integrity of swapped pages in untrusted memory. Further, applications stall if they needs any of the swapped-out pages.

To overcome this limitation of fixed protected memory, MazeNet splits large DNN models into smaller models referred to as submodels, such that each submodel fits within the protected memory offered by SGX. After splitting, each submodel can be deployed on different TEEs to avoid EPC swapping.

Figure 3 shows one such splitting of the VGG16 model, where the splits are performed after the pooling layers. In the case of functional models, where the output of one layer can be input to more than one following layers, the splits can be performed at block levels.

3.2 Submodel Cloaking

The submodels obtained from splitting can be hosted on TEEs to provide secure inference services. However, deploying submodels only on TEEs leaves other powerful but untrusted system resources, such as CPUs and GPUs, underutilised. Therefore, a subset of the submodels is deployed in the untrusted runtime environment to improve the performance of the inference service. The submodels deployed within TEEs are referred to as in-TEE submodels, while the submodels deployed outside the TEEs are referred to as non-TEE submodels.

However, outsourcing submodels to untrusted environments poses the following security and privacy risks, which were earlier mitigated by TEEs. First, passive adversaries can observe the delegated submodels in plaintext and therefore they can learn the connection between different layers along with their parameters and weights. Second, an active adversary can tamper with outsourced computation. It can replace the weights of outsourced submodels, change the results, or perform replay attacks on outsourced computations.

To protect the privacy of outsourced submodels, MazeNet employs its second technique of cloaking, where the submodels are cloaked before deploying them to untrusted environments. In cloaking, synthetic layers and neurons are added to the submodel to produce a cloaked submodel, where the original submodel is embedded within the cloaked submodel. Cloaking hides the existing weights within synthetic weights.

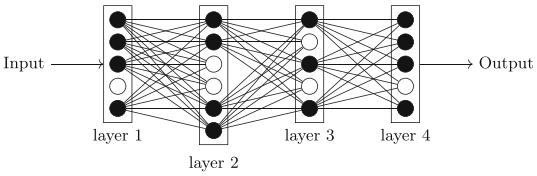


Fig. 4. Phase 1: Adding synthetic neurons (○) between existing neurons (●) in a four layer submodel.

Key idea. The key idea behind cloaking is that parameters or weights of a layer are a set of matrices, and it is difficult to distinguish whether a given matrix is part of a pre-trained model or not. For example, filters in convolutional layers detect different features. Given two filters, each detecting a different feature, it is difficult to determine which one is part of the trained model.

Prior work on model explanations that tries to reason about the working of DL models suggests that individual filters and units (neurons) need a global view of the entire model to reason about the usefulness or role of individual neurons in the predictions [48]. Thus, when synthetic weights are added in submodels, an adversary cannot distinguish between the embedded weights and the newly added synthetic weights. It can only observe the partial computation, which contains a mix of actual and synthetic computation. The keys to recover the results from cloaked submodels are securely stored within the TEEs.

For cloaking, a subset of submodels are marked as in-TEE, which are deployed within TEEs, while the remaining are marked as non-TEE, which are cloaked to produce cloaked submodels. The first and the last layer in the pre-trained model are always part of in-TEE submodels. The remaining submodels are alternatively marked as in-TEE and non-TEE. In the previous example of splitting VGG16 model in the Fig. 3, submodels $\mathcal{S}_1, \mathcal{S}_3, \mathcal{S}_5$ are marked as in-TEE submodels, while submodels $\mathcal{S}_2, \mathcal{S}_4$ are labelled as non-TEE and cloaked.

3.3 Cloaking Process

Non-TEE submodels are cloaked in two phases. In the first phase, synthetic neurons are inserted into existing (embedded) layers to hide embedded neurons. Then, in the second phase, synthetic layers are added to hide embedded layers.

Sample weight distributions. The weights of the synthetic neuron and layers are drawn from sample weight distributions. However, the probability distribution of weights for each layer is unknown beforehand, and during the training, layers learn one of the instances of these weights from the distributions. The weight distributions differ across layers, as each layer learns a different feature. For example, in convolutional networks, initial layers learn simple features such as edge detection, while deeper layers learn complex features such as facial expression. A sample distribution can be created for layers during the training and hyperparameter tuning phase by recording the weights of each layer. More-

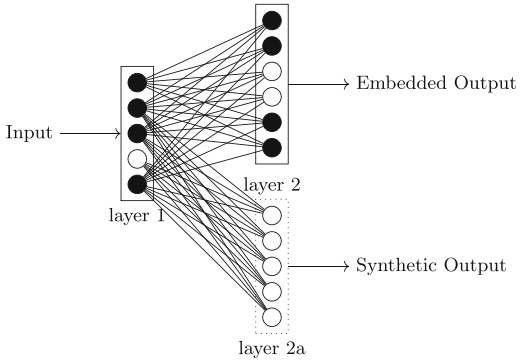


Fig. 5. Phase 2: Adding a synthetic layer, `layer 2a`, to a two layer submodel.

over, the sample distribution can be extended by training the model multiple times with different initial weights.

Phase 1: Adding Synthetic Neurons. In the first phase, synthetic neurons are added to the convolutional and dense layers. In case of convolutional layers, synthetic filters compatible with existing filters in the layer are sampled and appended to the existing filters. For dense layers, weight matrices compatible with the size of the input tensor are sampled and appended to existing weights. Finally, the weights are shuffled to hide the existing weights from the newly added weights.

Adding synthetic neurons and filters makes the layers incompatible with the following layers as the size of the outputs increases. For example, consider a dense layer containing 64 neurons. Suppose 32 neurons are added to the layer during the first phase of cloaking. Then, it will produce output of length 96 instead of 64. Thus, the following layer will become incompatible as it expects input size to be 64. Therefore, the outputs from cloaked layers are filtered before feeding them to following following layers. Figure 4 shows that the output of synthetic neurons is filtered before feeding them to the following layer. The output of synthetic neurons is used later in the second phase of cloaking when synthetic layers are added. Similarly, shuffling the filter and weight matrices results in incorrect results produced by layers. Therefore, the weights of the following layers are also reordered such that dot products compute the same results as before the weights were shuffled.

After synthetic neurons are added, embedded weights are hidden from the adversary. However, the adversary still knows that a subset of the weights are from the embedded model for any given layer. Therefore, to further hide the embedded layer as well, synthetic layers are added.

Phase 2: Adding Synthetic Layers. During the second phase, synthetic layers are added to the submodels obtained from the first phase of cloaking to build the final cloaked submodels. Adding synthetic layers embeds the existing submodel within a bigger model (cloaked submodel).

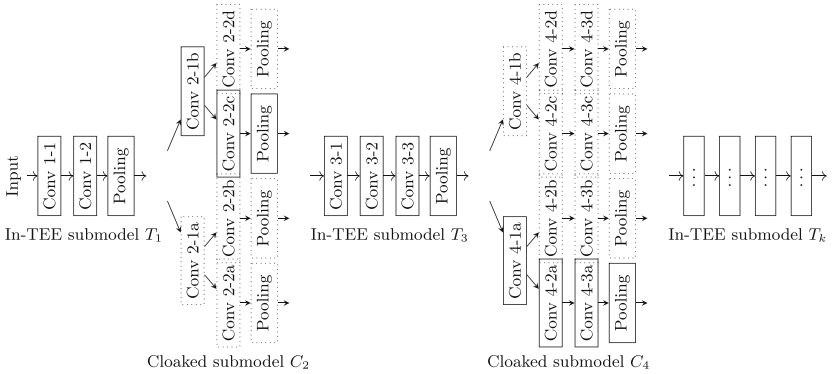


Fig. 6. One of the instances of the VGG16 MazeNet model obtained after splitting and cloaking. Layer represents embedded layers, while Layer represent synthetic layers.

A submodel can be hidden in another model as new architectures have been introduced which moves away from the traditional sequential architecture where the output of one layer is input to the immediately following layer. State-of-the-art models differ widely in their architecture from introduction of skip connection in ResNet [17], densely connected layer in DenseNet [18], parallel layers in Inception module of GoogleNet [47], and ensemble models, [12,17] which combine multiple independent models to form a bigger model. Thus, a submodel may appear to be part of multiple architectures. Further, adding synthetic layers increases the difficulty of identifying embedded layers based on the input-output relationship between the layers in the cloaked submodels.

The model owner provides the architecture of the synthetic layers for the second phase of cloaking, which specifies the location and the type of synthetic layers to be added in the cloaked submodels. To attach synthetic layers, intermediate inputs are selected from the submodel, and they are filtered. In contrast to embedded layer, the inputs to synthetic layers contain outputs of synthetic neurons as well as shown in Fig. 5, which were added in the first phase of cloaking. Then, the weights for the synthetic layers are drawn from the sample distributions, and the layers are added to the submodel.

The synthetic outputs produced by the synthetic layers are later eliminated by the TEEs. During cloaking, MazeNet uses taints to track the status of each value produced by the layers in the cloaked submodel. A taint tensor is maintained for each input, intermediate, and output tensor. The values in the taint tensor indicate whether the corresponding value in the actual tensor is `real` or `synthetic`. For the initial user input, the taint tensor is initialised with `real` label. As the input passes through the synthetic layers and synthetic neurons, the corresponding value in the taint tensor is set to `synthetic`. Similarly, `real` label is set for values produced by embedded layers or neurons.

Figure 6 shows one of the instances of generated MazeNet model corresponding to the VGG16 model. It consists of three in-TEE submodels, T_1, T_3, and T_5, and two cloaked submodels, C_2 and C_4.

4 Running MazeNet Models

A Model Manager on the cloud deploys the MazeNet models generated from pre-trained models. It deploys in-TEE submodels within Intel SGX enclaves and cloaked submodels on untrusted environments. Then, it exposes an API to query the deployed model. On receiving input from a user, it routes the input through a series of in-TEE and cloaked submodels to compute the inference results.

However, during the inference, an adversary on the cloud can read and modify any data and computations that are outsourced to the untrusted runtime to compromise the inference process or results. To detect integrity violations during cloaked submodel evaluation, MazeNet relies on digital signatures. The enclaves are built with the public key of the cloud provider. Thus, the public key cannot be modified or replaced during the inference. If the cloud vendor tampers with the private key that it controls, the signature verification will fail.

Outside the enclave, MazeNet requires the cloud vendor to compute signatures $Sign_{sk}(C_{input})$, where sk is the cloud vendor's private key, for each input C_{input} and output C_{output} of the cloaked submodels $C \in \mathcal{C}$. The enclave, which receives the intermediate output C_{output} as input for the in-TEE submodel, verifies the intermediate outputs and corresponding signatures. On successful verification, the enclave filters embedded outputs from synthetic outputs with taint obtained during cloaking and feeds the filtered output to the in-TEE submodel to proceed with the inference process. Then, the enclave sends both signatures to the Model Manager for auditing.

During an audit phase, the Model Manager can randomly verify some of the outsourced computation by re-executing those computations within a TEE, or outsource them to a non-colluding party, such as other cloud vendors or on-premises execution. The audit phase does not influence the throughput or latency of the system, as it happens when the system is idle or in an offline phase.

Security Analysis of MazeNet Models. All the in-TEE submodels run within TEEs; therefore, in-TEE submodel weights are secured by the hardware from adversaries. In the case of non-TEE submodels, each non-TEE submodel $\hat{T} \in \hat{\mathcal{T}}$ is cloaked to get a corresponding cloaked submodel $C \in \mathcal{C}$. To steal the weights of the embedded non-TEE submodel $\hat{T}$ from the cloaked submodel C, the adversary needs to correctly identify embedded neurons from synthetic ones. Equivalently, the adversary can steal the submodel if it can correctly guess the synthetic outputs from the embedded outputs, which are filtered by the enclaves.

Each subset of output from the cloaked model corresponds to a unique model. However, only one of them corresponds to the embedded model. Thus, the problem of submodel stealing for the adversary reduces to correctly identifying a subset from all possible subsets of a given set. For a set of size $|S|$, there are $2^{|S|}-1$ possible subsets, excluding the empty subset. Therefore, if a cloaked submodel produces N tensors as outputs, then the probability that an adversary can correctly guess the embedded non-TEE submodel $\hat{T}$ is:

$$P(\hat{T}) = \frac{1}{2^N - 1}$$

For the entire model, the number of expected weights that would be presented in randomly extracted submodels would be the sum of expected weights in the individual cloaked submodels. Therefore, the number of weights present in a randomly extracted model by the adversary would be:

$$E[\text{Embedded Weights}] = \sum_{C \in \mathcal{C}} \frac{|W_C|}{2^{||C||} - 1}$$

Here, $|W_C|$ is the number of embedded weights or parameters present in cloaked submodel C, and $||C||$ is the number of output tensor produced by the cloaked submodel C.

5 Implementation

We have built the MazeNet framework on TensorFlow [14]. It consists of two components: Model Builder and Model Manager. Model Builder accepts a pre-trained model in TensorFlow `SavedModel` format, and produces a MazeNet model consisting of in-TEE and cloaked submodels. The in-TEE submodels are exported in `TFLite` format, and the cloaked submodel in `SavedModel` format. In addition to the MazeNet model, the Model Builder produces taint tensors that are required during the inference process.

On the cloud, Model Manager manages the life cycle of MazeNet models. It is responsible for securely deploying in-TEE submodels within the TEEs and cloaked submodels in untrusted environments. The enclave hosting in-TEE submodels has secure access to taint tensors during the inference process to filter embedded outputs.

Our implementation relies on Intel SGX as a TEE. As applications do not run out-of-the-box on Intel SGX, the research community and industry have developed multiple frameworks to run applications on SGX. These include Graphene-SGX [51], Porpoise [42], Panoply [43], SGX-LKL [36]. Among these, we have selected Graphene-SGX (v1.0) as it supports the Python programming environment and the TensorFlow deep learning framework.

To implement the MazeNet framework, we have added cloaking support for popular layers in the TensorFlow framework, which were present in our benchmark models – VGG16, ResNet50, and DenseNet201. In total, TensorFlow has around 150 types of layers, and we implemented cloaking support for 20 TensorFlow layers for the benchmark models.

In the implementation, adding cloaking support for TensorFlow layers required 715 lines of Python code in Model Builder, and 550 lines for Model Manager as reported by `pygount` [3]. Further, MazeNet can be extended to other models by implementing cloaking for unsupported layers present in the model.

Table 2. Configuration parameter used to generate MazeNet models.

Model	Cloak factor	Submodel width	In-TEE layers
VGG16	10%	10	1, 11, 22
ResNet50	10%	10	1–7, 92–102, 174–177
DenseNet201	10%	10	1–7, 49–137, 477–709

6 Evaluation

To find the benefits and costs of the presented techniques, we transformed popular convolutional neural networks, VGG16 [24], ResNet50 [17], and DenseNet201 [18], into MazeNet models.

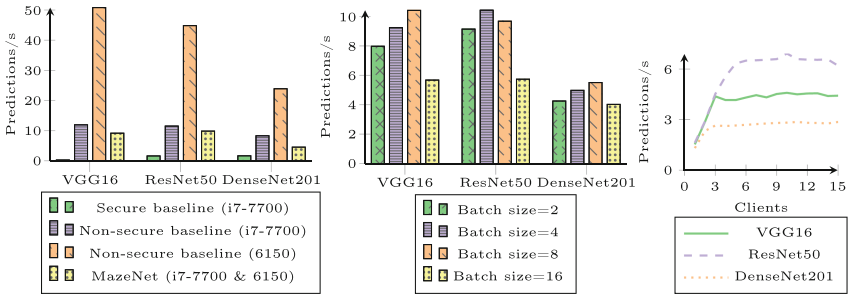
Generating MazeNet models. We have used pre-trained model weights present in Keras library [9] to produce MazeNet models. In MazeNet, the model developer provides the set of synthetic layers to be added during the cloaking phase. For the evaluation, we duplicated existing layers to produce cloaked submodels.

There are three key configuration parameters (split, cloak factor, cloaked submodel width) for building a MazeNet model. Table 2 lists the configuration parameters used in our evaluation.

1. **Split** states how the given model should be split into smaller submodels. According to the Table 2, layer 1, 11 and 22 of the VGG16 model are designated as in-TEE layers. Therefore, the Model Builder produces three in-TEE submodels, each containing one layer, while the remaining layers $\{L_2, \ldots, L_{10}\}, \{L_{12}, \ldots, L_{21}\}$ are part of two cloaked submodels.
2. **Cloak factor** represents the percentage of synthetic weights to be added to build a cloaked layer. For example, a convolutional layer with 64 filters and 10% cloak factor will result in 72 filters in the cloaked convolutional layer.
3. **Cloaked submodel width** limits the maximum width during cloaking, as duplicating layers increases the width of the submodel, where width is the number of layers present at a given depth d.

6.1 Experimental Setup

Based on the parameters in Table 2 to build MazeNet models, the Model Builder produces three in-TEE and two cloaked submodels. The in-TEE submodels are deployed on three TEE systems that are equipped with an Intel i7-7700 desktop-class CPU, which supports SGXv1 with 128 MB of cryptographically protected memory, and 32 GB of main memory. The remaining two cloaked submodels are deployed on non-TEE systems without SGX support. The non-TEE system is equipped with server-class Xeon Gold 6150 CPU, having 36 cores and 72 threads, along with 256 GB of main memory. As the server-class machine has a high core count and sufficient main memory, both the cloaked submodels are deployed on



(a) Maximum throughput of secure baseline and MazeNet models.

(b) Throughput of MazeNet models at different batch sizes.

(c) Throughput of MazeNet models at different workloads.

Fig. 7. Throughput of MazeNet models generated from parameters in Table 2.

the same machine. Our evaluation focuses on throughput and latency, where the inference service performs classification on images from the ImageNet [40].

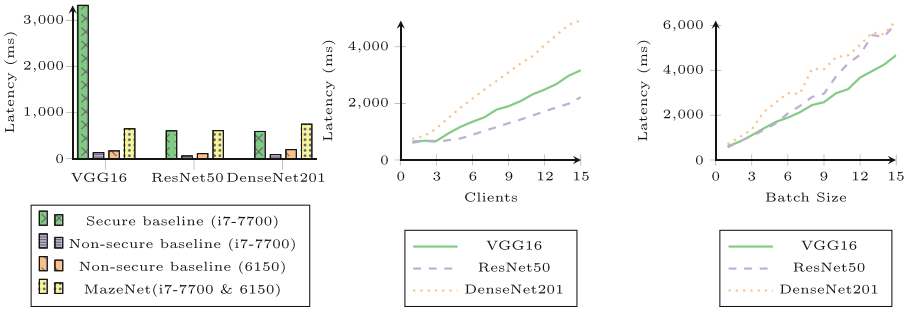
Baseline models. We compare the performance of MazeNet models against *secure baseline* models where the whole unmodified model executes within a TEE. The secure baseline model executes on one of the TEE systems (i7-7700) described above. As the non-TEE system does not support any TEE, we cannot report secure baseline performance for the 6150 CPU. Similarly, for *non-secure baseline*, we run unmodified models in the standard untrusted environment.

The accuracy of MazeNet models is similar to unmodified models, as MazeNet performs the same set of computations in addition to synthetic computations.

6.2 Throughput Results

To evaluate throughput, we query the models with 128 input samples that are split across eight clients, where each client simulates a single user and query the models in parallel. We repeat this experiment with different batch sizes, where multiple inputs are grouped to form a batch, and report the maximum throughput across batch sizes, for each model in Fig. 7a. The results demonstrate that MazeNet models achieve higher throughput when compared to the secure baseline models. MazeNet models benefit from the faster untrusted processors, whereas the secure baseline models are limited by the weak trusted CPU. However, the speedup observed across models differs significantly from 30x in VGG16 to 2x in DenseNet201. The speedup is more significant in VGG16 as it is computationally expensive, requiring 30.96 GFLOPs as compared to ResNet50 and DenseNet201 models, which require 7.73 and 8.58 GFLOPs, respectively. Thus, the VGG16 MazeNet model benefits more from untrusted processors.

During the experiments, we observed that each MazeNet model achieved maximum throughput at a different batch size. Therefore, we next measure the role of batch size in the throughput of MazeNet models. To measure the impact



(a) Latency of baseline and MazeNet models with a single client.

(b) Latency of MazeNet models with multiple clients.

(c) Latency of MazeNet models at different batch sizes.

Fig. 8. Latency trends of MazeNet models.

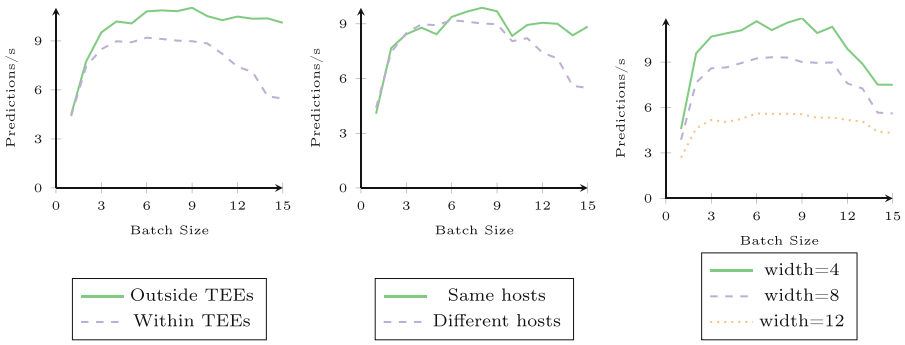
of batch size, eight clients query the model at different batch sizes, and the results are presented in Fig. 7b. Increasing the batch size improves the throughput of MazeNet models. However, increasing the batch size beyond the optimal point leads to a significant decline in throughput. The primary reason for this degradation of throughput is EPC swapping in enclaves due to the large intermediate state produced by in-TEE submodels at higher batch sizes.

Finally, we evaluate how the throughput scales with increasing workload. Initially, a single client queries the model with a fixed batch size of one. Then, the number of clients is progressively increased, while keeping the batch size fixed, to simulate the increasing workload. Figure 7c shows the throughput of MazeNet models at varying workloads. Initially, the overall throughput of MazeNet inference system increases with an increase in workload. However, the throughput saturates when the number of clients crosses six.

6.3 Latency Results

Latency is the time duration a client has to wait for results after sending the inputs to the model. To measure the latency of models, we query the model with a single client that queries the model with a single input. The observed latency for different models is reported in Fig. 8a. The results show a significant improvement in latency for the VGG16 MazeNet model compared to the secure baseline, with latency dropping from 3 to 0.6 s – a 5x improvement.

However, there is no latency improvement for the ResNet50 and DenseNet201 MazeNet models. There are two main sources that contribute to the latency of MazeNet models, in addition to computational operations. First is the time spent on computing the digital signatures by cloaked submodels and verification of signatures by enclaves. Second is the time spent on data transfers between cloaked and in-TEE submodels. As compared to the VGG16 MazeNet Model, the ResNet50 and DenseNet201 MazeNet models spend more time on computing



(a) **Overhead of enclave execution.** Throughput of the VGG16 MazeNet model when the in-TEE submodels are outside TEEs.

(b) **Network overhead.** Throughput of the VGG16 MazeNet model when in-TEE and cloaked submodels are deployed on the same host.

(c) **Overhead of synthetic computations.** Throughput of VGG16 MazeNet models with different amounts of synthetic layers.

Fig. 9. Overheads arising from different sources during MazeNet inference.

digital signatures and data transfers than performing deep learning operations, which can be accelerated by untrusted hardware.

Next, we evaluate how the latency changes with increasing workload. We began with a single client which queries the model with a single input. Then, progressively, we increase the number of clients over time. Figure 8b presents the latency trends of MazeNet models under the varying number of clients. Initially, the Latency increases slightly when the number of clients is increased from one to five. As the number of clients increases, the number of active models in the inference pipeline increases. However, when the pipeline is full or saturated, further increasing the number of clients proportionally increases the latency as the input queries are queued by the Model Manager.

Next, we investigate the impact of batch size on the latency of MazeNet models. We measure the latency of MazeNet models at different batch sizes while keeping the number of clients constant (one). Initially, the client sends the query with a single input. Then, it gradually increases the number of input samples within the batch. Figure 8c reports the latency for the entire batch. For all the models, the latencies at different batch sizes follow the same pattern. The batch latency increases with an increase in batch size. However, the average time spent per input sample within a batch reduces from 0.60 to 0.30 s when the batch size is increased from one to fifteen. Similarly, it decreases from 0.56 to 0.40 s for ResNet50, and 0.71 to 0.41 s for DenseNet201. The results show that batching can reduce the time spent per input during inference.

6.4 Overheads

Overhead of enclave execution. Applications run slowly within enclaves due to the overheads intrinsic to enclave execution. To quantify the overhead of enclave execution, we measure the throughput of MazeNet models in two configurations: in-TEE submodels within TEEs and in-TEE submodels outside TEEs. In the first configuration, we run MazeNet models in the standard configuration as per the experimental setup described in Sect. 6.1. In the second configuration, in-TEE submodels are deployed in an untrusted environment to avoid the overheads of enclave execution, while the remaining setup remains the same.

Figure 9a plots the throughput of the VGG16 MazeNet model in both configurations. The throughput of the standard configuration, in-TEE submodels with TEEs, is within 20% of the other configuration for batch sizes up to eight, as the throughput is bottlenecked by cloaked submodel execution time instead of the enclave execution. However, at higher batch sizes, EPC swapping occurs due to the larger intermediate state produced by the in-TEE submodels, which shifts the bottleneck from the cloaked submodel to EPC swapping in enclaves. Consequently, the throughput of in-TEE submodels within TEE starts to decrease at higher batch sizes. Thus, the overhead of enclave execution is more prominent when there is EPC swapping, while it is minimal without EPC swapping.

Network Overhead. Intermediate results of submodels are transferred over the network, which introduces overhead during the inference process. We evaluate the overhead due to the 1 Gigabit Ethernet network in our experimental setup. To isolate network overhead, we run both the in-TEE submodels and cloaked submodels and in-TEE submodels on the same non-TEE system, the server machine described in the experimental setup, Sect. 6.1. This eliminates the need for network transfers during the inference process. As the in-TEE submodels runs outside the enclave, similar to the previous experiment, we can offset the performance gains from running in-TEE submodels outside the enclave, up to 20% at smaller batch sizes. We compare the throughput of the VGG16 MazeNet model in the above two configurations. The throughput in both cases is similar, as the inference process is compute-bound. During inference, VGG16 submodels transfer around 10 MB of data per single input inference. As the MazeNet model in standard configuration achieves around eight inferences per second, the network bandwidth of one Gigabit does not introduce any significant overhead.

Overhead of synthetic computations.: To quantify the overhead due to the synthetic computations present in the cloaked submodels, we compare the number of floating-point operations required to compute inference results in unmodified models and MazeNet models. Table 3 lists the number of floating point operations (FLOPs) in unmodified and MazeNet models, which were generated from the parameters given in Table 2. Next, we evaluate the impact of different amounts of synthetic layers on the throughput of MazeNet models. Figure 9c shows the throughput trends for three models, each having different widths.

Table 3. Number of Floating-Point Operations (FLOPs) in standard and Mazenet models when models were cloaked with parameters in Table 2.

Model	Standard	MazeNet Submodels			
		in-TEE	Cloaked	Total	Increase
	(GFLOPs)				
VGG16	30.96	1.85	151.76	153.60	5x
ResNet50	7.73	0.684	60.85	61.54	8x
DenseNet201	8.58	3.17	51.18	54.36	6x

7 Related Works

Prior works have used trusted execution environments to protect the privacy and confidentiality of training and inference of deep learning models [5,6,19,26]. Another line of research has focused on offloading deep learning computation to hardware accelerators [16,46,50,53]. The main limitation of these works is that they outsource only linear layers. However, often linear layers are followed by a non-linear layer. Therefore, the intermediate outputs of the linear layers need to be constantly moved between the TEE and the GPUs. MazeNet overcomes this by outsourcing both linear and non-linear layers to untrusted environments, thus reducing the constant need to transfer data between TEE and GPUs.

Another line of works has presented cryptography-based solutions that use homomorphic encryption schemes [10,13,41] and multi-party computation [25,30,35,37–39] protocols or a combination of both techniques [23,33] to protect deep learning workloads. Recent work [22] reports sub-second latency for the VGG16 model with GPUs on a smaller CIFAR dataset, while MazeNet achieves sub-second latency with CPUs on the larger ImageNet dataset. Cryptography-based techniques face two primary challenges when applied in real-world deployments. The first is the significant computational overhead associated with cryptographic operations. The second is the high communication cost incurred by interactive protocols. Together, these limitations make such approaches less practical for applications where low latency and high throughput are critical.

8 Conclusion

In this work, we presented MazeNet to protect the privacy models on public cloud platforms with TEEs, and introduced methods to outsource portions of computation during inference to untrusted hardware. Our outsourcing scheme outsources both the linear and non-linear layers. We implemented the presented techniques in a prototype framework, MazeNet, to build MazeNet models from given pre-trained models and deploy the MazeNet models on a public cloud platform to provide inference services. Our evaluation of popular convolutional

networks demonstrates that MazeNet models can improve the throughput by up to 30x and the latency by up to 5x as compared to the secure baseline models.

Acknowledgements. This work is funded by the Foundation for Science Innovation and Development (FSID), Indian Institute of Science, through the "Security and Privacy of Smart Cities, Sub-Project: Secure Enclaves for Sensitive Applications in Smart Cities" project, managed by IUDX, and "Secure Digital Enclaves for Sensitive Applications" grant. Vivek Kumar is currently affiliated with Goldman Sachs, Bangalore, India, but was a student at the Indian Institute of Science while working on this project.

References

1. Right to financial privacy act of 1978, 12 USC 3401 – 3422
2. Health insurance portability and accountability act of 1996 (1996), pUBLIC LAW 104–191—AUG. 21 (1996)
3. Aglassinger, T.: Pygount. https://pypi.org/project/pygount/
4. Amodei, D., Hernandez, D.: Ai and compute. https://openai.com/blog/ai-and-compute/
5. Bayerl, S.P., et al.: Offline model guard: secure and private ml on mobile devices. In: DATE 2020 (2020)
6. Brasser, F., Gens, D., Jauernig, P., Sadeghi, A.R., Stapf, E.: Sanctuary: arming trustzone with user-space enclaves. In: NDSS (2019)
7. Chen, G., Chen, S., Xiao, Y., Zhang, Y., Lin, Z., Lai, T.H.: Sgxpectre: stealing intel secrets from SGX enclaves via speculative execution. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 142–157. IEEE (2019)
8. Chen, X., Ji, J., Yu, L., Luo, C., Li, P.: Securenets: Secure inference of deep neural networks on an untrusted cloud. In: Proceedings of The 10th Asian Conference on Machine Learning (2018)
9. Chollet, F.: Keras. https://keras.io/api/applications/
10. Dathathri, R., et al.: Chet: an optimizing compiler for fully-homomorphic neural-network inferencing. In: Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation, pp. 142–156 (2019)
11. Kaplan, D., Jeremy Powell, T.W.: AMD memory encryption. https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/memory-encryption-white-paper.pdf (2021)
12. Ganaie, M.A., Hu, M., Malik, A.K., Tanveer, M., Suganthan, P.N.: Ensemble deep learning: a review. Eng. Appl. Artif. Intell. **115**, 105151 (2022)
13. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: applying neural networks to encrypted data with high throughput and accuracy. In: Proceedings of The 33rd International Conference on Machine Learning (2016)
14. Google: tensorflow. https://github.com/tensorflow/tensorflow
15. Hanzlik, L., et al.: Mlcapsule: guarded offline deployment of machine learning as a service. In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW) (2021)
16. Hashemi, H., Wang, Y., Annavaram, M.: Darknight: an accelerated framework for privacy and integrity preserving deep learning using trusted hardware. In: MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture (2021)

17. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
18. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4700–4708 (2017)
19. Hunt, T., Song, C., Shokri, R., Shmatikov, V., Witchel, E.: Chiron: privacy-preserving machine learning as a service (2018)
20. Intel: Q3 2018 speculative execution side channel update. https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00161.html
21. Jagielski, M., Carlini, N., Berthelot, D., Kurakin, A., Papernot, N.: High accuracy and high fidelity extraction of neural networks. In: 29th USENIX Security Symposium (USENIX Security 20) (2020)
22. Jawalkar, N., Gupta, K., Basu, A., Chandran, N., Gupta, D., Sharma, R.: Orca: FSS-based secure training and inference with gpus. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 597–616 (2024)
23. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: Gazelle: a low latency framework for secure neural network inference. In: Proceedings of the 27th USENIX Conference on Security Symposium, pp. 1651–1668. SEC'18, USENIX Association, USA (2018)
24. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, pp. 1097–1105. NIPS'12, Curran Associates Inc., Red Hook, NY, USA (2012)
25. Kumar, N., Rathee, M., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: Cryptflow: secure tensorflow inference. Cryptology ePrint Archive, Paper 2019/1049 (2019)
26. Kunkel, R., Quoc, D.L., Gregor, F., Arnautov, S., Bhatotia, P., Fetzer, C.: Tensorscone: a secure tensorflow framework using intel SGX (2019)
27. Lee, T., et al.: Occlumency: privacy-preserving remote deep-learning inference using SGX. In: The 25th Annual International Conference on Mobile Computing and Networking (2019)
28. Limited, A.: Building a secure system using trustzone technology. https://documentation-service.arm.com/static/5f212796500e883ab8e74531 (December 2008)
29. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious neural network predictions via minionn transformations. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (2017)
30. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious neural network predictions via minionn transformations. In: Proceedings of the 2017 ACM SIGSAC conference on computer and communications security, pp. 619–631 (2017)
31. Ltd., A.: Arm Confidential Compute Architecture (2021). https://developer.arm.com/documentation/den0125/0300. Accessed 2 April 2025
32. McKeen, F., et al.: Innovative instructions and software model for isolated execution. In: HASP (2013)
33. Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., Popa, R.A.: Delphi: a cryptographic inference system for neural networks. In: Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice, pp. 27–30 (2020)
34. Mo, F., et al.: Darknetz: towards model privacy at the edge using trusted execution environments. In: Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services (2020)

35. Mohassel, P., Zhang, Y.: Secureml: a system for scalable privacy-preserving machine learning. In: IEEE Symposium on Security and Privacy (SP) (2017)
36. Priebe, C., et al.: SGX-LKL: Securing the host OS interface for trusted execution. In: arXiv:1908.11143 (2019)
37. Rathee, D., et al.: Cryptflow2: practical 2-party secure inference. In: ACM CCS 2020 (2020)
38. Riazi, M.S., Weinert, C., Tkachenko, O., Songhori, E.M., Schneider, T., Koushanfar, F.: Chameleon: a hybrid secure computation framework for machine learning applications. Cryptology ePrint, pp .2017/1164 (2017)
39. Rouhani, B.D., Riazi, M.S., Koushanfar, F.: Deepsecure: scalable provably-secure deep learning. In: Annual Design Automation Conference (2018)
40. Russakovsky, O., et al.: ImageNet large scale visual recognition challenge. Int. J. Comput. Vision **115**(3), 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y
41. Sanyal, A., Kusner, M., Gascon, A., Kanade, V.: Tapas: tricks to accelerate (encrypted) prediction as a service. In: International conference on machine learning, pp. 4490–4499. PMLR (2018)
42. Shanker, K., Joseph, A., Ganapathy, V.: An evaluation of methods to port legacy code to SGX enclaves. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (2020)
43. Shinde, S., Le Tien, D., Tople, S., Saxena, P.: Panoply: low-TCB linux applications with SGX enclaves. In: NDSS (2017)
44. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: 2017 IEEE symposium on security and privacy (SP), pp. 3–18. IEEE (2017)
45. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: 3rd International Conference on Learning Representations (ICLR 2015) (2015)
46. Sun, Z., Sun, R., Liu, C., Chowdhury, A.R., Jha, S., Lu, L.: Shadownet: a secure and efficient system for on-device model inference (2020)
47. Szegedy, C., et al.: Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1–9 (2015)
48. Szegedy, C., et al.: Intriguing properties of neural networks. In: International Conference on Learning Representations (ICLR) (2014)
49. Taassori, M., Shafiee, A., Balasubramonian, R.: Vault: reducing paging overheads in SGX with efficient integrity verification structures. SIGPLAN Not. (2018)
50. Tramer, F., Boneh, D.: Slalom: fast, verifiable and private execution of neural networks in trusted hardware. In: International Conference on Learning Representations (2019)
51. Tsai, C., Porter, D.E., Vij, M.: Graphene-SGX: a practical library OS for unmodified applications on SGX. In: USENIX Annual Technical Conference (2017)
52. Van Bulck, J., et al.: Foreshadow: extracting the keys to the intel SGX kingdom with transient out-of-order execution. In: Proceedings FO the 27th USENIX Security Symposium (2018)
53. Zhang, Z., et al.: No privacy left outside: On the (in-) security of tee-shielded DNN partition for on-device ml. In: 2024 IEEE Symposium on Security and Privacy (SP), pp. 3327–3345. IEEE (2024)