SOTER: Guarding Black-box Inference for General Neural Networks at the Edge

Tianxiang Shen[1†], Ji Qi[1†], Jianyu Jiang[1*], Xian Wang[1], Siyuan Wen[1], Xusheng Chen[1], Shixiong Zhao[1],
Sen Wang[2], Li Chen[2], Xiapu Luo[3], Fengwei Zhang[4], Heming Cui[1]
[1]*The University of Hong Kong* [2]*Huawei Technologies Co., Ltd.*
[3]*The Hong Kong Polytechnic University* [4]*Southern University of Science and Technology*

Abstract

The prosperity of AI and edge computing has pushed more and more well-trained DNN models to be deployed on third-party edge devices to compose mission-critical applications. This necessitates protecting model confidentiality at untrusted devices, and using a co-located accelerator (e.g., GPU) to speed up model inference locally. Recently, the community has sought to improve the security with CPU trusted execution environments (TEE). However, existing solutions either run an entire model in TEE, suffering from extremely high inference latency, or take a partition-based approach to handcraft partial model via parameter obfuscation techniques to run on an untrusted GPU, achieving lower inference latency at the expense of both the integrity of partitioned computations outside TEE and accuracy of obfuscated parameters.

We propose SOTER, the first system that can achieve model confidentiality, integrity, low inference latency and high accuracy in the partition-based approach. Our key observation is that there is often an *associativity* property among many inference operators in DNN models. Therefore, SOTER automatically transforms a major fraction of associative operators into *parameter-morphed*, thus *confidentiality-preserved* operators to execute on untrusted GPU, and fully restores the execution results to accurate results with associativity in TEE. Based on these steps, SOTER further designs an *oblivious fingerprinting* technique to safely detect integrity breaches of morphed operators outside TEE to ensure correct executions of inferences. Experimental results on six prevalent models in the three most popular categories show that, even with stronger model protection, SOTER achieves comparable performance with partition-based baselines while retaining the same high accuracy as insecure inference.

1 Introduction

Driven by the remarkable success of AI [25] and edge computing [11,17], giant companies are increasingly shifting their well-trained Deep Neural Network (DNN) models from the cloud to enormous edge devices, to compose mission-critical applications such as autopilot navigation [8,15], home monitoring [20] and visual assistance [49]. By employing accelerators (e.g., GPU), clients of edge devices can conduct low-latency inferences without connecting with a remote server with high latency and network instability, and clients do not have to transfer their sensitive data to the cloud.

Since obtaining accurate models requires model providers to pay substantial resources to train parameters with private datasets [10,45], in order to preserve their competitive advantages, model providers usually require their offloaded models to keep black-box (confidential) with traditional *semantic security* guarantee [7]: a client can query the local model for results, but will learn nothing about the parameters' *plaintexts*.

To preserve model confidentiality, a pragmatic approach is to use the publicly available Trusted Execution Environments (TEE), in particular Intel SGX [14], a pervasively used CPU TEE product. SGX provides both code integrity and data confidentiality for enclaves [5,33], and avoids severe performance downgrading caused by using traditional cryptographic tools (e.g., Homomorphic Encryption [36,44]), making it attractive for model providers to protect their offloaded models while retaining high inference service quality.

Much prior work has explored using SGX for secure model inference, and these approaches can be summarized into two categories. First, the *TEE-shielding* approach, which runs the entire unmodified model in enclaves, achieves both model confidentiality and high accuracy same as the original model, but suffers from extremely high latency due to the limited computing resources on CPU's enclave and the lack of publicly available secure GPU devices [3,65]. As a typical TEE-shielding work, MLcapsule [27] incurs dramatically higher latency (up to 36.1X) than insecure GPU inference for diverse workloads (§6.1).

In order to mitigate the high latency issue, the second category of work, including eNNclave [56] and AegisDNN [68], takes a *partition-based* approach to manually select a portion of sensitive model layers to run in an enclave, and partition the rest of the layers to run on an untrusted GPU for acceleration, achieving lower inference latency than TEE-shielding approaches by utilizing the strong GPU computing power.

Unfortunately, existing partition-based approaches face a fundamental *confidentiality-accuracy* dilemma: some of the approaches replace the partitioned layers' parameters with public parameters from other models composed by the same layers, which effectively protects model confidentiality but trades off the accuracy. This is because the original parameters are usually trained by model provider's private datasets tailored for a specific task [10,45]; hence the parameters are

[†]Tianxiang Shen and Ji Qi contribute equally.
[*]Jianyu Jiang is the corresponding author.

exclusive to achieve the original high accuracy. In contrast, some partition-based approaches preserve the original high accuracy by holding plaintexts of partitioned operators' parameters on an untrusted GPU, which partially compromises the confidentiality by revealing a fraction of parameters' plaintexts to the adversary outside an enclave.

Our key observation to resolve this dilemma is the *associativity* property of many inference operators. *Associativity* means that the way that factors are grouped in an operator's computation does not change the final result [43]. With associativity, we can securely transform an associative operator into *parameter-morphed*, thus *confidentiality-preserved* operator by scaling its parameters with a hidden scalar in an enclave, run that morphed operator on the GPU and fully restore the execution result of that operator with the scalar hidden in an enclave. Since associative operators (e.g., *Convolution*, *Fully-connected*) widely exist in general DNN models, and these operators represent a major fraction of computation in a DNN model (e.g., 93.5% of the computation on VGG19 [61] is spent on convolution), we can achieve dramatically increased performance by partitioning morphed, computationally expensive operators to run on a GPU.

Based on this observation, we present SOTER[1], the first partition-based approach that achieves model confidentiality, low latency and high accuracy simultaneously. SOTER carries a *Morph Then Restore* (MTR) protocol for cooperative executions between kernels (enclave and GPU). Specifically, SOTER randomly selects a major fraction of associative operators, morphs their parameters with randomly generated *blinding coins* (scalars) in the enclave, and partitions these parameter-morphed operators to run on a GPU. For each client's input, SOTER executes each operator (either in an enclave or GPU) in order, transfers intermediate execution results (IR) across kernels when needed, and restores the final execution result with reciprocal coins in the enclave. A subtle case is that, under special partition cases, an adversary who observes IRs transmitted across kernels can reveal the value of coins hidden in an enclave (§4.1). Hence, SOTER additionally morphs the value of IRs before they are transmitted across kernels to hide the blinding coins, thus protecting the model confidentiality.

However, even with these steps, SOTER still faces an inherent integrity hazard in the presence of an adversary at the edge side, who can observe and manipulate any components (e.g., morphed parameters on GPU) outside the enclave to mislead the offloaded model to produce wrong output, ruining the model provider's inference service quality.

To tackle the integrity hazard, SOTER's MTR protocol generates *oblivious fingerprints* at runtime to detect integrity breaches outside the enclave in a *challenge-proof* manner [71]. Our key idea is that the associativity property can be used to efficiently generate integrity challenges at runtime. Specifically, before running inferences, SOTER collects *fingerprints* of partitioned operators in enclave, which hold the ground-truth of these operators' execution results; during the inference, SOTER challenges partitioned operators with fingerprint input to ask them for execution proofs. By comparing the returned proofs with expected fingerprint output in enclave, SOTER learns whether integrity breaches occur.

[1] SOTER is an ancient Greek god, guards safety against cosmic chaos.

System	Model Confidentiality	GPU Acceleration	High Accuracy	Inference Integrity	General Functionality
∗ MLcapsule [27]	√	×	√	√	√
∗ Privado [26]	√	×	√	√	√
∗ Occlumency [41]	√	×	√	√	√
• Serdab [21]	×	√	√	×	×
• Darknight [28]	×	√	×	√	√
• eNNclave [56]	√	√	×	×	×
• AegisDNN [68]	×	√	√	×	√
• **SOTER**	√	√	√	√	√

Table 1: Comparison of SOTER and related systems. "∗ / •" means that the system uses either a TEE-shielding approach (∗), or a partition-based approach (•).

To prevent an adversary from distinguishing fixed fingerprint challenges thus bypassing the detection, inspired by traditional steganography techniques [34, 52], SOTER dynamically derives new fingerprints by using existing fingerprints in enclave based on the associativity property. The new fingerprints statistically share the same distribution as client's normal input (§4.2), making fingerprints oblivious to the attacker. Therefore, by leveraging the same key observation of associativity, SOTER achieves both model confidentiality and integrity in a unified manner.

We implemented SOTER with Graphene-SGX [63], a mature framework that supports developing neural network applications with Intel SGX. We evaluated SOTER with six popular DNN models covering three popular categories, including Multi-layer Perception (MLP) [40], Convolution Neural Network (CNN) [24] and Transformer [64]. We compared SOTER to three notable TEE-based secure inference systems, covering both the TEE-shielding approach and partition-based approach. Our evaluation shows that:

- SOTER is secure. For confidentiality, SOTER effectively hid parameters' plaintexts, and achieved comparable model protection as TEE-shielding baseline under powerful model stealing attacks (§6.3); for integrity, SOTER detected any integrity breaches within ten fingerprint challenges with an overwhelmingly high probability of 99.9% (§6.4).
- SOTER is efficient. SOTER achieved up to 72.6% lower latency than the TEE-shielding baseline, and had moderate latency (up to 1.2X) compared to partition-based baselines, while the baselines do not have integrity protections (§6.1).
- SOTER is accurate. SOTER retained the same high accuracy as insecure inference, while the most efficient partition-based baseline caused 1.1%∼5.5% accuracy drops.

Our major contribution is the MTR protocol, the first work that achieves model confidentiality, low latency and high ac-

curacy with integrity protection for secure model inference. Compared to existing relevant baselines, SOTER achieved comparable strong confidentiality as the TEE-shielding approach, comparable low-latency as the partition-based approach, high accuracy same as insecure inference, and overwhelming high probability of detecting integrity breaches outside an enclave. This makes SOTER unique to greatly promote the prosperity of AI on edge devices, encouraging enormous model providers to develop powerful models and deploy them on third-party edge devices. Also, our MTR protocol can generally protect model inference on the cloud when the model provider does not trust the owner of cloud servers that host the model. SOTER's source code is released on github.com/hku-systems/SOTER.

In the rest of this paper: §2 introduces background; §3 gives an overview of SOTER; §4 describes SOTER's design; §5 covers SOTER's implementation; §6 shows our evaluation; §7 discusses related work and §8 concludes.

2 Background

2.1 Deep Neural Network

A DNN model (in short, model) can be represented as a sequence of connected *layers* with each layer assigned a set of *operators*, as shown in Figure 1. An operator is either a *linear operator* or a *nonlinear operator* where a linear operator is weighted by *parameter matrices* (in short, parameters).

Inference workflow. Figure 1 shows the inference workflow. A model M passes an input X (e.g., an image) through layers of operators to compute *logits* [30], normalizes logits with *softmax* function to produce a probability vector, and assigns a class with the highest probability to input X as the class label. Without losing generality, we use image classification as an example to illustrate model composition in the following discussions.

Associativity of DNN operators. Operators are the basic building blocks of a DNN model, among which linear operators have been proven to take up the majority of computation resources in general model inferences [1, 70].

Many DNN models deployed at the edge are built on top of *associative* operators. Suppose we have an input X and a scalar μ, a DNN operator F is associative if

$$(\mu^{-1} * \mu) \cdot F(X) = \mu^{-1} \cdot F(\mu * X) \quad (1)$$

Linear operators, including computationally expensive *convolution* and *fully-connected*, have the associativity property as they conduct linear transformation on input data. For instance, take the convolution operator as an example: as shown in Figure 1, if we multiply each element in the convolution kernel by 2^{-1} (i.e., $\mu = 2^{-1}$), we will get the output $2^{-1}R$, while it always holds $(2 * 2^{-1}) * R = 2 * (2^{-1}R)$ in Equation 1. This property applies to other linear operators as well.

For nonlinear operators that conduct nonlinear transformation on data, most of them do not have an associativity property (e.g., *Sigmoid*). However, interestingly, under specific

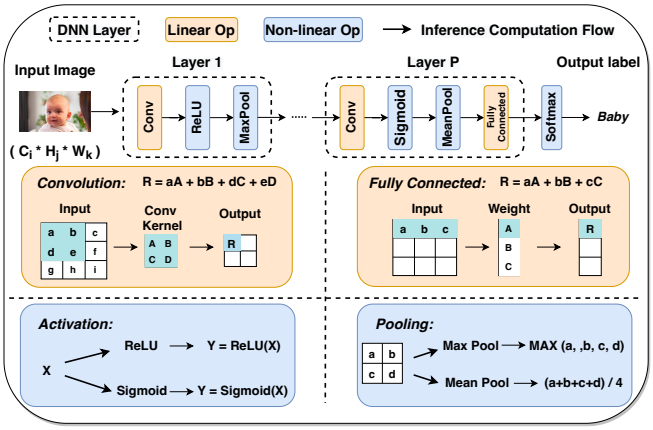


Figure 1: An overview of the DNN model.

constraints, some nonlinear operators can also be associative. Take the most commonly used *ReLU* as an example: the *ReLU* function, $F(x) = Max\{0, X\}$, is *scale-invariant* when $\mu > 0$, i.e., $F(\mu x) = Max\{0, \mu X\} = \mu F(X)$. Hence, given a scalar μ, Equation 1 applies to *ReLU*. Similar to *ReLU*, the *pooling* function is associative as well.

Operators that satisfy Equation 1 also meet a variant of the associativity property. Given that

$$F(X_1) = y_1, F(X_2) = y_2, ..., F(X_n) = y_n$$

it always holds

$$F(\sum_{i=1}^{n} \mu_i * X_i) = \sum_{i=1}^{n} \mu_i * y_i \quad (2)$$

SOTER leverages Equation (1) as the key weapon to produce parameter-morphed, thus confidentiality-preserved operators to run on GPU for acceleration, and restore accurate results in enclave. SOTER uses Equation (2) to efficiently generate oblivious fingerprints for integrity checking at runtime. We illustrate our detailed designs in §4.

2.2 Intel SGX and Related Work

Intel Software Guard eXtension (SGX) [14] is a pervasively used hardware feature on commodity CPUs. SGX provides a secure execution environment called enclave, where data and code execution cannot be seen or tampered with from outside.

As shown in Table 1, there are two categories of SGX-based work that provides secure inference service. The first category is the TEE-shielding approach that runs all inference computations within enclave (e.g., MLcapsule [27], Privado [26], Occlumency [41]). Such an approach shields the entire model in enclave to hide parameters' plaintexts, but fails to achieve low inference latency owing to the computational bottleneck of CPU and the lack of publicly available trusted GPU [3, 65].

The second category is the partition-based approach that runs a portion of model layers in enclave and runs the rest of the layers on a GPU for acceleration. AegisDNN [68] only shields partial critical model layers in enclave and accelerates other plaintext layers on GPU. To decide which layers should

be partitioned, AegisDNN takes a user-specified argument (i.e., deadline for an inference task), uses silent data corruption mechanism [23] to learn each layer's criticality, and partitions uncritical (plaintext) layers to GPU to meet the stringent deadline. eNNclave [56] argues that the feature extraction operators (e.g., convolution) of different models are generally transferable, hence it replaces partitioned operators' parameters with pre-trained parameters of other models, which sacrifices inference accuracy. This is because, model providers usually train their models with private datasets tailored for a specific task, thus each parameter in the trained model is exclusive to achieve high accuracy [10, 45, 72]. Also, such an approach is only applicable to specific models where public parameters are available. Serdab [21] and Darknight [28] assume the model is deployed on a trusted cloud. Instead of protecting model confidentiality, they have an orthogonal goal of protecting users' data privacy on the cloud.

3 Overview

3.1 System setup

We consider a client-side inference scenario shown in Figure 2. Different from a cloud-side inference scenario (e.g., Delphi [44], Gazelle [36]) where the model is hosted on the trusted cloud server, we consider the model provider offloads its model to the client's untrusted edge device to run model inferences. The device constantly takes sensitive queries from the client and sends inference results back to the client.

3.2 Security model

We consider an honest model provider that provides the correct model requested by the client, and the model is offloaded to run on an SGX-equipped third-party edge device. We trust the hardware and firmware of Intel SGX, which ensure that code and data in enclave can not be seen or tampered with from outside. However, any components outside the enclave are untrusted.

We consider a malicious edge-side attacker outside the enclave that aims to (1) steal the parameters of the offloaded model, and (2) perturb any components outside the SGX to modify the inference results. An edge-side attacker could be a business competitor who wants to steal the model for competitive advantages and ruin the inference service to screw up the model provider's reputation [12, 13, 54]. Even worse, the integrity attack against edge-side model inference could pose severe threats to edge users. For instance, an attacker may hack into a self-driving system running with an obstacle detection model, and perturb the model parameters to produce incorrect navigation instructions to the car [9].

Semantic security. Similar to prior secure inference systems, namely the MLcapsule [27] and eNNclave [56], SOTER aims to achieve model confidentiality with the semantic security guarantee: knowing only ciphertexts, it must be infeasible for a computationally-bounded adversary to derive significant information about the plaintexts [7]. In the secure inference scenario, it captures the requirement that the parameters' plaintexts cannot be derived from any data observed by the adversary.

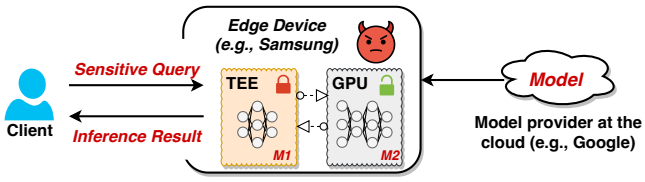


Figure 2: The client-side inference scenario.

Note that, we do not hide information that is revealed by the results of inference queries, and we focus just on protecting the parameters' plaintexts. Protecting against attacks that exploit the leakage of inference results is a complementary problem to that solved by SOTER. We give a detailed illustration of these attacks and potential mitigations in §7.

3.3 System overview

SOTER's two-phase design. SOTER's protocol consists of an offline *preprocessing phase*, and an online *inference phase*. Specifically, the preprocessing phase is independent of the client's query input which changes regularly. We assume the offloaded model from the server is static, if the model changes, then both parties should re-run SOTER's preprocessing phase. After preprocessing, during the inference phase, the client sends query input to get the eventual result. Note that, SOTER is best suited for applications whose inference is latency-sensitive, but is usually not performed frequently enough to take up all computational resources needed for preprocessing.

SOTER's workflow. In Figure 3, we show how SOTER leverages the general associativity property of DNN operators to automatically partition a DNN model.

In the preprocessing phase: during **P1**∼**P3**, a SOTER client conducts standard SGX attestation to the server for obtaining the model M and decryption keys KEY_M, and then loads the encrypted model M in a layer-wise manner by decrypting with KEY_M locally. In **P4** and **P5**, SOTER extracts the model architecture, statically filters out all *associative* operators that meet Equation 1, and then invokes SOTER's MTR protocol for model partitioning.

Specifically, in **P5**, SOTER's MTR protocol runs as follows: With a given partition ratio θ, SOTER randomly selects θ% associative operators, and generates (1) *fingerprints* of the selected operators for integrity checking (used in the inference phase), and (2) random scalars for all operators. These scalars serve as the *blinding coins* to hide the parameters [44] and are always kept secret in enclave. With the associativity property (Equation 1), SOTER morphs every selected associative operator by multiplying each element in the operator's parameter matrices with a blinding coin (the computation result can be restored by using the reciprocal coin), and then partitions the selected operators with morphed parameters to GPU. In Figure 3, the selected operators (in green) are morphed with

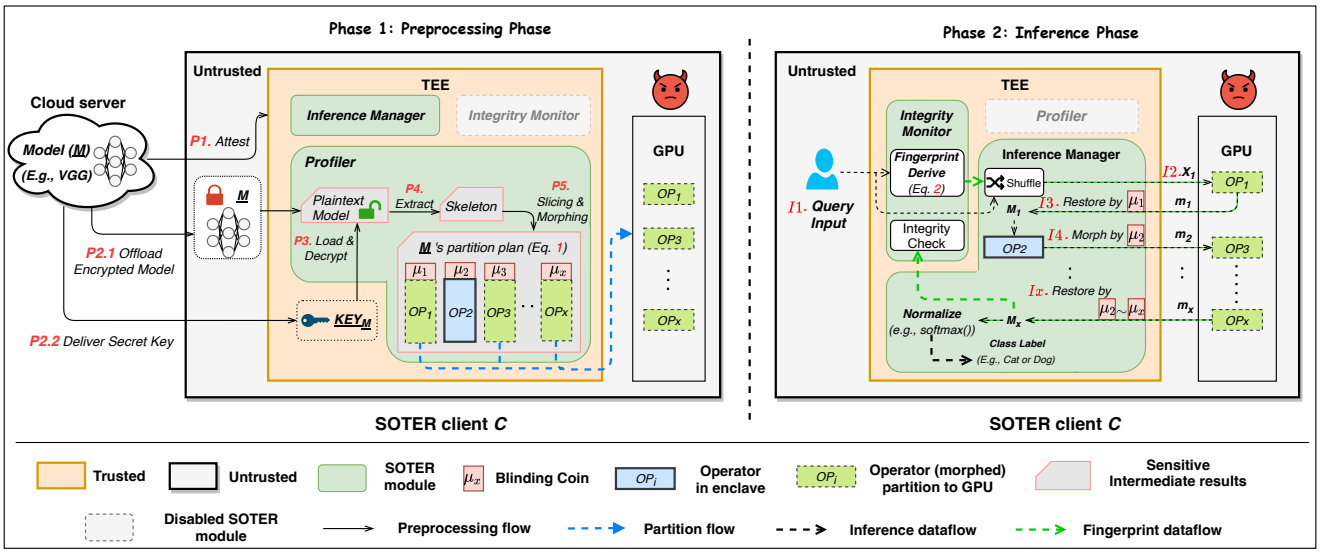


Figure 3: The architecture of SOTER in a modular manner. SOTER's secure inference protocol has three modules (shielded in green) running in an enclave. An adversary can observe and manipulate any components outside the enclave.

corresponding coins, e.g., the parameters of the third operator OP_3 are morphed with its coin μ_3 and partitioned to GPU, while the second operator (in blue) is kept within an enclave.

Note that, we choose to run the preprocessing procedures (**P1**∼**P5**) at the client-side because client C might select his θ according to his GPU capability, e.g., choose a smaller θ if the GPU has limited usable memory.

In the inference phase, there are two types of data flows: the inference flow (black dotted line) and the fingerprint data flow (green dotted line). The life-cycle for the inference data flow is: (**I1**) The client C sends a query input (i.e., data for inference) to the trusted inference manager module and shuffles with the fingerprint input (introduced below). (**I2**) The inference manager forwards C's input X_1 to the GPU, and the first *morphed* operator OP_1 takes X_1 as input and computes the output m_1. (**I3**) Since the next operator OP_2 is expected to run within enclave according to the plan made in the preprocessing phase, the inference manager takes the computation result of previous partitioned operators (i.e., m_1) as the input, restores m_1 with all previous partitioned operators' blinding coins since the last kernel switch (in Figure 3's circumstance, SOTER restores with only OP_1's blinding coin μ_1), and gets the restored result M_1. (**I4**) After that, the second operator OP_2 computes with M_1 and gets an intermediate result (IR), then morphs IR with the OP_2's blinding coin μ_2 and forwards the result m_2 to the partitioned operator OP_3 in GPU.

The above procedures (**I2**∼**I4**) repeat until all partitioned operators in GPU are computed. In (**Ix**), with the last computation result m_x from OP_x, SOTER restores the real result with all partitioned operators' coins since the last kernel switch (same as **I3**). In this case, SOTER restores with $\mu_2 \sim \mu_x$ and gets the real inference result M_x. Last, SOTER runs a standard normalization to get the class label for client C.

The fingerprint data flow is used to detect integrity breaches of partitioned operators outside an enclave. Specifically, with a set of fingerprints produced in the preprocessing phase (**P5**), SOTER dynamically derives new fingerprints by utilizing the general associativity of Equation 2, injects these fingerprints into the inference data flow to check whether the partitioned parameters have been maliciously modified by an adversary outside the enclave.

SOTER's generality. SOTER supports general neural networks. SOTER's key insight into partitioning neural networks lies in the broad associativity of common inference operators (§2.1), including all linear operators (e.g., *convolution* and *fully-connected*) and typical nonlinear operators (e.g., *ReLU*, *Max-pool*, and *Mean-pool*). Hence, SOTER theoretically supports all neural networks (including recurrent neural networks [69]), but exhibits varying performance gains (compared with running the entire neural network in TEE) depending on the ratio of associative operators in neural networks (§6.1). SOTER also supports general TEEs (e.g., ARM TrustZone [53] and AMD SEV [58]) as long as the TEE provides data confidentiality and code integrity guarantees that SOTER's MTR protocol requires.

4 Protocol Description

This section describes SOTER's MTR protocol. At a high level, MTR utilizes the general associativity property of DNN operators to automatically profile a model (with Equation 1), randomly selects a portion of associative operators, and then morphs these operators' parameters with hidden *blinding coins* in enclave to hide parameters' plaintexts (§4.1). Besides, to tackle the integrity threat (described in §3.2), stemming from the same observation of the general associativity property, MTR dynamically derives *oblivious fingerprints* (with

Table 2: SOTER's protocol variables.

Variable	Description
$\Theta(default = 0.8)$	Portion of associative operators partitioned to GPU.
$O_g \mid O_e$	Partitioned operators in GPU / maintained in TEE.
$O(O_g \cup O_e)$	The whole set of model M's operators.
$Para(O_i)$	Parameters of operator O_i.
FP_{O_i}	Fingerprint of operator O_i.
μ_i	Blinding coin of operator O_i.

Equation 2) and uses fingerprints to check the integrity of partitioned operators in a *challenge-proof* manner (§4.2). Table 2 shows the variables used in the MTR protocol.

4.1 Morph Then Restore (MTR) protocol

The MTR protocol is divided into two stages: a *morphing* stage and a *restore* stage. In the morphing stage, SOTER first makes *blinding coins* for every operator (including both associative and non-associative operators), and then makes a partition plan with a given partition ratio θ. Then, in the restore stage, SOTER runs inference across GPU and enclave, and restores inference results in enclave when needed. Algorithm 1 and 2 show the MTR protocol.

Stage 1-1: Morphing with blinding coins. SOTER assigns every operator with a randomly generated scalar, which serves as the blinding coin to hide the plaintext value of parameters. Given an operator O_i (the partition plan is described in **Stage 1-2**), SOTER morphs $Para(O_i)$ by multiplying each element in $Para(O_i)$ with the corresponding coin μ_i. Note that, SOTER requires periodically updating of the coins to avoid potential chosen plaintext attacks [6,37], as the morphing is completely linear. According to the hill cipher theory [48], each coin can tolerate up to n^2 attacks (i.e., n^2 inferences) where n is the parameter matrix's size of the operator that the coin applies to. Hence, SOTER updates a morphed operator with matrix size n every n^2 inferences. Notably, this updating can be done off the critical path when the inference tasks are not busy (§5).

- **Protect coins during kernel switches.** One subtle case is that, SOTER assigns random coins to every operator (rather than selected operators for partitioning) to avoid potential information leakage during kernel switches, i.e., when intermediate results (IRs) are transmitted between the enclave and GPU. We illustrate our design by giving a running example.
- **Running example.** We demonstrate our idea with the example in Figure 4, which is a common architecture in typical DNN models. SOTER selects a portion of model operators to run in the TEE enclave (green portion), and partitions the remaining operators to run on the untrusted GPU (red portion). An adversary outside the enclave is attempting to deduce the blinding coins used to morph the GPU operators according to the attacker-visible IRs (Y_1 and Y_2).

We begin with the **Before** case, in which only the partitioned GPU operators are protected by SOTER's blinding coins. OP_1 (on the GPU) is a linear operator and OP_2 (in enclave) is a non-linear *ReLU* operator. If we only morph OP_1 with μ_1 and do not assign a coin to OP_2, then, with a client input X, OP_1 outputs $y_1 = X(\mu_1 * OP_1)$ and OP_2 outputs $y_2 = ReLU(y_1/\mu_1) = ReLU(X * OP_1)$. However, since the *ReLU* operator only filters out negative values in parameter matrices and does not transform a scalar, an adversary who observes both y_1 and y_2 will directly infer the value of coin μ_1, violating the confidentiality of the partitioned OP_1 (concretely, $Para(O_i)$). To tackle this problem, we assign coins to all operators rather than partitioned GPU operators only. As demonstrated in the **After** case of Figure 4, by assigning coin μ_2 to OP_2, OP_2 will output $y_2 = \mu_2 * ReLU(X * OP_1)$. An adversary observes $\mu_1 * \mu_2$ but has no way of inferring either μ_1 or μ_2.

Algorithm 1: MTR protocol at client c (offline) (§4.1).

```
▽ Preprocessing Phase (within enclave).
1  Function partition(O) do
2      foreach operator op in O do
3          op.partition ← False; op.index ← O.index(op);
4          if op.index > W then
5              if 0 < normalize(sgx_read_rand()) < Θ then
6                  op.partition ← True;
7              if op is associative & op.partition then
8                  foreach element e in para(op) do
9                      e ← μ_index × e ;          // Morph
10                 O_g.add(op) ;          // GPU operators
11             else
12                 O_e.add(op) ;          // TEE operators
13     O_g.copyTo ("cuda") ;          // Partition to GPU
```

Overall, by morphing both the enclave and the partitioned GPU operators with blinding coins stored in TEE enclave, an attacker cannot deduce the blinding coins from the IRs, ensuring the confidentiality of the model parameter's plaintexts.

Stage 1-2: Partitioning model with hidden operators. With all coins prepared, given a partition ratio θ, we automatically partition a portion of associative operators (that are morphed) to GPU for inference acceleration. The *partition* function in line 1∼13 of Algorithm 1 shows the pseudo-code.

Specifically, in the preprocessing phase, SOTER iterates every operator in model M and randomly selects a portion of associative operators and adds them to the partition set O_g. For instance, with $\theta = 0.8$, an associative operator would have an 80% chance to be partitioned to GPU. After all operators are iterated, all operators in O_g are partitioned to GPU for inference acceleration.

Note that, SOTER always keeps top-W (W=2 by default) operators in enclave even if these operators are associative (Line 4). This design choice is made to ensure the input to the first several partitioned operators on GPU (e.g., X_1 in Figure 3) are always unknown to the adversary, such that we can stealthily check the integrity of all partitioned operators by injecting fingerprint challenges (more details in §4.2).

Stage 2: Guarded black-box inference. Next, we present the

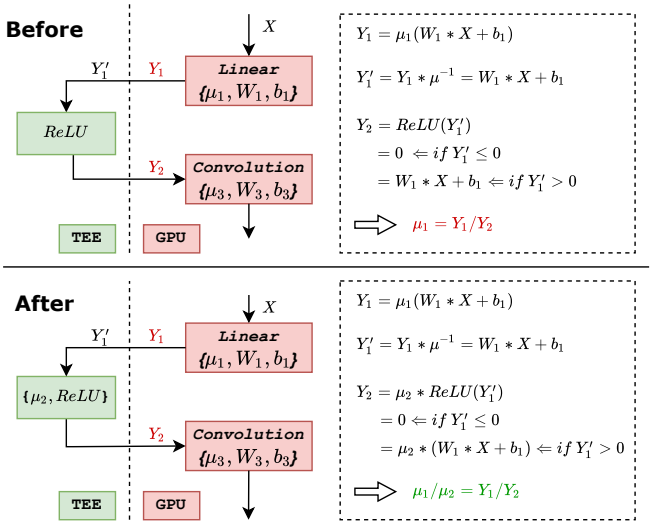


Figure 4: Morphing only partitioned operators' parameters can leak the blinding coins in some partition cases.

end-to-end black-box inference process given that we have prepared coins and partitioned a portion of operators to GPU.

As shown in Algorithm 2, SOTER first computes the reserved top-W operators in enclave, and then iterate **I2**∼**I4** as follows. **I2**: SOTER finds the longest length of consecutively partitioned operators in O_g and computes the inference result by forwarding the input through these operators (line 6∼11). **I3**: Then SOTER copies the result back to enclave, restores the real inference result by multiplying the reciprocal coins of all operators in the last step (line 13), and then forwards the result to operators maintained in enclave (before the next copy to GPU). **I4**: SOTER morphs the forwarding result with all coins of enclave operators in the last step (line 18), and then copies back to GPU. This procedure terminates when all partitioned operators on GPU are iterated. Then SOTER normalizes the final result and returns the result to the client.

4.2 Integrity checking with fingerprints

Obliviousness requirement. To detect integrity breaches outside an enclave (as described in §3.2), a straw man approach could be using a fingerprint (with ground-truth input/output pair) to *challenge* the partitioned operator on GPU to recompute the fingerprint input to provide *proof*, and report a crime if the proof is different from the expected output.

However, such an approach needs to be oblivious. Since the adversary can continuously monitor the inference process and observe every intermediate result (IR) transmitted from enclave to GPU, if using only a fixed set of fingerprints, the adversary can easily distinguish those fixed challenges among all IRs, and bypass our detection by returning correct proofs.

The timeliness-obliviousness dilemma. Unfortunately, trivially using different fingerprints for integrity checking brings a *timeliness-obliviousness* dilemma.

Specifically, generating new fingerprint input and pre-

Algorithm 2: MTR protocol at client c (online) (§4.1).

```
▽ Inference Phase (cross enclave & GPU).
1  Function secure_inference() do
       ▽ I1.
2      every client input X do
3          X.copyTo("enclave");
4          foreach index i ∈ normalize(W) do
5              X ← O_i(X) ;                    // Top-W Inference
6          start ← next op in O_g; end ← next op in O_e;
7          while start < |O| do
               ▽ I2.
8              X.copyTo("cuda");
9              foreach index i ∈ range(start, end) do
10                 X ← O_i(X) ;                // GPU Inference
               ▽ I3.
11             X.copyTo ("enclave");
12             X ← ∏_start^(end-1) μ_i^-1 × X ;   // Restore
13             start ← end;
14             end ← start from O_end find the next op in O_g;
15             foreach index i ∈ range(start, end − 1) do
16                 X ← O_i(X) ;                // TEE Inference
               ▽ I4.
17             X ← ∏_start^(end-1) μ_i × X ;   // Morph
18         X ← normalize(X);
19         return Top_1 (X) ;                  // Final result
```

computing expected output in CPU would cause late detection, because CPU computation for an operator is extremely slow (up to 30x slower inference speed for linear operation); on the other hand, if we use a fixed set of fingerprints for challenging the integrity, we could timely detect breaches. However, the detection would be easily bypassed by the adversary because all fingerprints are now distinguishable.

SOTER's insight and approach. Stemming from the same observation of parameter morphing (§2.1), we tackle this challenge by efficiently generating new fingerprints in CPU without re-computing operators. At a high level, we take two steps by first making a set of *cornerstone fingerprints* for each partitioned operator, and then encapsulating new fingerprints from these cornerstone fingerprints with a constant cost at runtime based on the associativity property.

Step 1: Preparing cornerstone fingerprints. During the preprocessing phase, SOTER prepares K (by default K=10) cornerstone fingerprints for each partitioned operator. A fingerprint is a two-element tuple: {*input*, *output*}, where the input is a randomly generated matrix and the output is pre-computed by forwarding the input through a ready-to-partition operator. The whole preparing procedure is running within enclave, such that the correctness of each cornerstone fingerprint can be guaranteed.

Step 2: Efficiently deriving new fingerprints in TEE. Now with K cornerstone fingerprints for each partitioned operator, SOTER efficiently derives oblivious fingerprints at runtime by leveraging the general associativity property. Specifically, with a fixed set of cornerstone fingerprints $FP_O[K]$ for each

partitioned operator O, we randomly select T fingerprints and generate T random coefficients with SGX's trustworthy random number generator. Then, by applying the associativity property (Equation 2), we generate a new fingerprint FP_{New} as follows: the input for FP_{New} is $\sum_{i=1}^{T} T_i * FP_O[i].input$, i.e., we multiply each selected cornerstone fingerprint's input with a corresponding coefficient and add them all as the new input; For the output, since the associativity property implies that the corresponding output for a group of associated operators has the same transformation as the input, thus we can directly get the expected output as $\sum_{i=1}^{T} T_i * FP_O[i].output$ without conducting slow CPU inference for a given new input.

Step 3: Challenge-proof fingerprint issuing. For fast detection of integrity breaches, rather than submitting fingerprint challenges for random user inference tasks, SOTER submits fingerprint challenges for every user inference task. By doing so, SOTER can detect any integrity breaches within only ten fingerprints with an overwhelmingly high probability of 99.9% according to our theoretical analysis and evaluation results (§6.4).

In detail, SOTER issues a *challenge* whenever a kernel switch (an IR transmitted from enclave to GPU) happens. When an IR (produced by client's query) is sent to a partitioned operator O, we challenge the integrity of O by sending the IR to O together with a new $FP_O.input$ (produced in **Step 2**), and compare the returned *proof* with the expected $FP_O.output$. Any mismatch reveals an integrity breach of partitioned operator O.

Whenever integrity breaches are detected, SOTER can simply abort the inference process and restart the preprocessing phase to prepare new blinding coins and re-morph the GPU operators with the procedures described in §3.3.

5 Implementation

We built SOTER on top of PyTorch C++ [51] with 5.3K LoC. We chose PyTorch C++ as it incurs less memory footprint compared to PyTorch, and previous work [63] has shown that running programs with a high memory footprint within SGX could incur prohibited overhead due to SGX's small memory capacity. SOTER uses Graphene-SGX to run SOTER's components within an SGX enclave (denoted as SOTER-Graphene), because Graphene-SGX provides a library OS for running applications within the enclave without any modifications.

Due to SGX's limitation (§2.2), Graphene-SGX cannot directly access GPU within the enclave. SOTER tackles this problem by spawning an extra PyTorch process (SOTER-GPU) outside the enclave for offloading computations to GPU. SOTER-Graphene and SOTER-GPU communicate through shared memory (untrusted and not protected by SGX). We did not choose to modify Graphene-SGX to forward GPU computations at the syscall-level (e.g., by modifying `ioctl`), because doing so results in frequent enclave transitions for each GPU task. This is because launching one GPU task requires multiple system calls (e.g., `ioctl` and `mmap`).

Runtime Construction. SOTER builds the runtime according to the two-phase design (§3). In the *preprocessing* phase, SOTER bootstraps the enclave with the standard SGX attestation library [14], and decrypts the model (in `ONNX` format [42]) sent from the server in the client enclave. The enclave bootstrapping takes 1.84∼2.92 seconds. Then, the decrypted model is processed by `enclave_model_dispatcher`, which randomly selects a major fraction of associative operators and morphs these operators' parameters with random scalars produced by the SGX trustworthy source (`sgx_read_rand`).

In the *inference* phase, SOTER-Graphene runs inferences on DNN layers stored within TEE. When an inference computation needs to be offloaded outside the enclave, SOTER-Graphene serializes the activation using `pickle`, pushes the data to the shared memory and hands over to SOTER-GPU; SOTER-GPU deserializes the data and the morphed model, computes the results, and pushes the result to the shared memory; SOTER-Graphene retrieves the results and continues execution.

Optimization. First, SOTER reduces the memory footprint by reusing a single `paraheap` buffer to store operators' parameters, and gradually loading and decrypting parameters from disk when they are required for computations in CPU. Second, SOTER enables SGX paging [14] to support the execution of large DNN models. Note that all baseline systems we evaluated were also enabled with these two optimizations. Third, SOTER takes unused CPU cycles to provision new coins and produce newly morphed operators in TEE (§4.1), to replace staled partitioned operators on GPU in a nonstop manner.

6 Evaluation

Testbed. Our evaluation was conducted on a server with 2.60GHZ Intel E3-1280 V6 CPU, 64GB memory, and SGX support. The partitioned computations were performed on a co-located Nvidia 2080TI GPU with 11 GB physical memory.

Baseline systems. To evaluate the performance of SOTER, we compared SOTER with three notable TEE-based secure inference systems, namely MLcapsule [27], AegisDNN [68] and eNNclave [56]. MLcapsule is a popular TEE-shielding open-source project, which shields the entire model within the CPU enclave without any modification to parameters, thus achieving model confidentiality, integrity (no partitioned operators outside enclave) and retaining high accuracy as the original trained model.

AegisDNN and eNNclave are two state-of-the-art partition-based systems that empower accelerator-assisted low-latency inference same as SOTER (Table 1). AegisDNN only shields partial critical model layers in enclave and accelerates other plaintext layers on a GPU. To decide which layers should be partitioned, AegisDNN takes a user-specified argument (i.e., deadline for an inference task) and uses a silent data corruption mechanism [23] to learn each layer's criticality. AegisDNN is not open-source so we implemented all its protocols. eNNclave handcrafts a partition plan by manually replacing

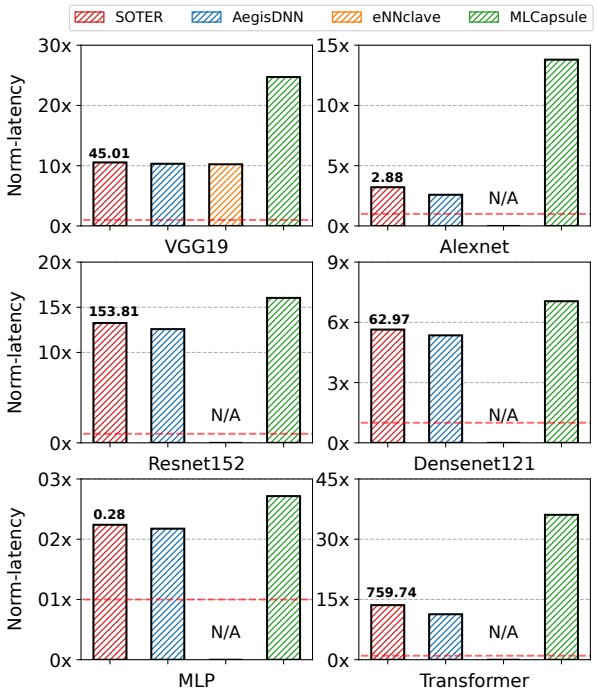


Figure 5: Normalized latency of all systems running six prevalent models. All systems' latency results are normalized to **insecure** GPU inference latency (red dotted line). The value on each red bar indicates SOTER's averaged inference latency.

some layers' parameters with public insensitive parameters, and running these parameter-precision downgraded layers on a GPU. Hence, eNNclave achieves model confidentiality at the expense of accuracy, and is only suitable for specific models with publicly available parameters of known layers. We ran eNNclave based on its open-source implementation.

Note that, both AegisDNN and eNNclave do not provide integrity protection for a partitioned model. While some approaches (e.g., Darknight [28]) have integrity protection for computations on untrusted accelerators, they are not designed to protect model confidentiality at the untrusted edge; hence they are orthogonal to the goals of SOTER.

Models and datasets. We evaluated all baseline systems with six well-studied DNN models in the three most popular categories that are widely used in the deep learning community [73], including a Multi-layer Perception (MLP) model, four Convolution Neural Network models including Alexnet (AN) [38], Densenet121 (DN) [31], VGG19 (VGG) [61], Resnet152 (RN) [29], and a Transformer (TF) [64]. We used the open-source release of each model.

We conducted our study on two representative datasets, ImageNet [16], and WNMT [57], targeting both computer vision (CV) and natural language processing (NLP) tasks. ImageNet is a full-scale image dataset that contains 600k images from 1k classes for CV studies; WNMT is the major translation dataset that has been used in recent NLP studies.

Default setting. By default, we ran all experiments by sequentially feeding the inference data (i.e., *input_batch_size*=1). Unless conducting sensitivity studies, we ran SOTER with 80% probability of partitioning an associative operator to run on a GPU (i.e., *selective partition ratio*=0.8 in §4.1).

SOTER's inference results (in milliseconds)

Model	MLP	AN	VGG	RN	DN	TF
P1: CPU (TEE)	0.19	1.65	25.38	92.18	41.65	439.52
P2: GPU	0.05	0.71	14.24	33.97	13.71	204.93
P3: Kernel Switch	0.01	0.18	0.83	25.98	5.6	41.52
P4: Integrity Check	0.03	0.34	4.56	14.75	6.02	73.77
End-to-end (P1+P2+P3+P4)	0.28	2.88	45.01	153.88	62.97	759.74

Table 3: End-to-end latency breakdown of SOTER.

AegisDNN's inference results (in milliseconds)

Model	MLP	AN	VGG	RN	DN	TF
P1: CPU (TEE)	0.19	1.54	22.89	88.14	36.75	404.87
P2: GPU	0.07	0.67	19.96	52.3	20.07	198.64
P3: Kernel Switch	0.01	0.12	0.85	7.12	1.81	29.61
End-to-end (P1+P2+P3)	0.27	2.33	43.7	146.56	58.63	633.12

Table 4: End-to-end latency breakdown of AegisDNN.

Our evaluation focuses on the following questions:

§6.1 How efficient is SOTER compared to baselines?

§6.2 How sensitive is SOTER's performance to different partition ratio?

§6.3 What could be leaked with and without SOTER?

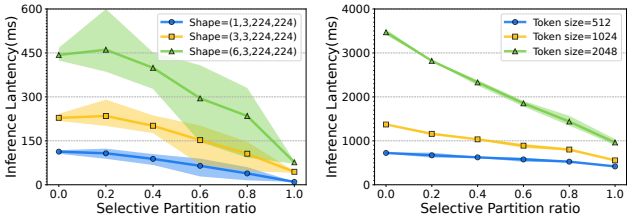
§6.4 How robust is SOTER under integrity breaches?

6.1 End-to-end inference performance

We first investigated the end-to-end inference latency of SOTER and three baseline systems with six prevalent models. All reported measurements are the averaged inference latency of 10k independent inference input.

Figure 5 shows the comparison of normalized inference latency, where all systems are normalized to the insecure GPU inference, which was measured by directly running model inference on GPU without protection. SOTER's end-to-end latency (in milliseconds) is reported on its bar. N/A means a system cannot handle such a case because eNNclave only reports its method on partitioning VGG-series models. Overall, SOTER's latency was $1.21X \sim 4.29X$ lower than MLcapsule when running inferences on the same model. This is because MLcapsule shields the entire model in the CPU enclave without involving GPU, and CPU is at least one order of magnitude less efficient than GPU in executing inference tasks [62].

Meanwhile, SOTER incurred $1.03X \sim 1.27X$ higher latency compared to AegisDNN for two reasons. First, SOTER additionally enforces integrity protection of partitioned operators on GPU (§4.2) while AegisDNN does not support integrity check. Second, AegisDNN partitions models in a coarse-grained manner: by packing consecutive layers into blocks and partitioning these blocks to run on a GPU, AegisDNN reduces the number of kernel switches (memory copy) between enclave and GPU. In contrast, SOTER partitions model in a fine-grained operator granularity, thus may incur more kernel switches on some models. Next, we evaluate this moderate performance downgrading with a detailed latency breakdown.



(a) (b)

Figure 6: Running SOTER on (a) VGG19 with different input shapes (*batch_sise*, *channels*, *height*, *weight*), and (b) Transformer with different token size.

Latency breakdown. To understand SOTER's low latency and moderate overhead, we recorded the time taken for each step of the workflows in SOTER and AegisDNN, as shown in Table 3 and Table 4.

In addition to the integrity check and kernel switch overhead, we observed that SOTER took more time for enclave computations by comparing **P1** in both tables. This is because SOTER provides confidentiality for all model operators by either shielding plaintext operators in enclave, or selectively morphing associative operators to run on a GPU and reverting concealed execution results to accurate results in enclave with preserved blinding coins (§4.1). In contrast, AegisDNN only protects confidentiality for partial operators in enclave, and leaves other operators' plaintext parameters to the untrusted GPU. Therefore, SOTER paid additional overhead for parameter morphing and restoration in enclave.

Besides, when running models with complex dependencies among operators (Resnet, Transformer), SOTER's operator-level partition protocol incurred more frequent kernel switches than AegisDNN (**P3**), but such design protects the entire model architecture from being leaked. As recent work on AI demonstrates that model architecture has a significant impact on achieving high inference accuracy [2, 22], a complex, thus potentially well-designed architecture is urgent to get protected. In contrast, AegisDNN partitions a bunch of consecutive layers to GPU thus leaking architecture information to the adversary outside enclave. We will further investigate the information leakage in all systems in §6.3.

Overall, compared to the TEE-shielding baseline (MLCapsule [27]), SOTER achieves lower inference latency by securely partitioning most associative operators to run on an untrusted GPU, and the performance gain brought by GPU computations on associative operators dominates the overhead for TEE paging and TEE-GPU interaction (Table 3). Typical associative operators (e.g., convolution and fully connected) are proven to be the major computational bottleneck in neural networks. For example, 93.5% of the computation on the classic VGG19 model is spent on convolution [61]. However, compared with the partition-based baselines (eNNclave [56] and AegisDNN [68]), SOTER incurs slightly higher (up to 1.27X) latency for using fingerprints to detect integrity breaches (latency breakdown in Table 3 and Table 4).

No accuracy loss. SOTER retained high inference accuracy provided by original models, because SOTER ran either unmodified operators in enclave or morphed associative operators in GPU, while concealed execution results of morphed operators can be reverted with the associativity property (§2.1). MLcapsule directly shields the entire model in enclave without modification so it retained high original accuracy; AegisDNN partitions plaintext model layers to run on a GPU so it incurred no accuracy downgradation as well; eNNclave obfuscates partitioned operators' parameters by replacing them with public parameters. Since public parameters are not tailored for a given task, eNNclave incurred $1\% \sim 5.5\%$ accuracy drops on the evaluated VGG19 model.

In summary, SOTER achieves much lower inference latency than MLcapsule, but SOTER incurs slightly higher latency than AegisDNN and eNNclave because SOTER pays additional effort to provide stronger model confidentiality and integrity, and retains high accuracy same as original models.

6.2 Sensitivity

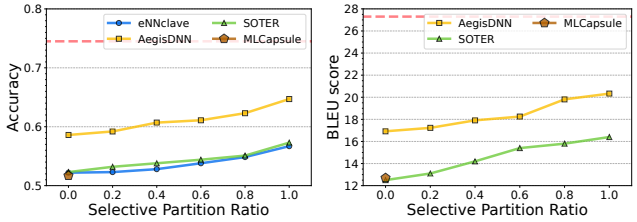
We tested different partition ratios (denoted as θ) to look into the performance sensitivity of SOTER. The experiments were conducted on VGG19 with different input shapes, and we ran different values of θ for each input shape; each experiment was conducted ten times, as shown in Figure 6a.

The shading next to each line is the performance jitter caused by different computation volumes of randomly selected associative operators partitioned to GPU. With an increased θ, SOTER's inference latency dropped accordingly because more associative operators were selected and partitioned to a co-located GPU for acceleration. Interestingly, we observed that when θ is small ($0.1 \sim 0.2$), the latency of $\theta = 0.2$ could be even larger than $\theta = 0.1$. This instability is because, with a small partition ratio, the partitioned operators were fragmented, and GPU's performance gain could be amortized by the frequent kernel switches between enclave and GPU. When θ increases, the latency gain became more stable because the partitioned operators became less fragmented, thus fewer kernel switches would occur.

Comparing figure 6a with figure 6b, we observed that SOTER achieved more stable performance on Transformer. This is because, the number of operators in VGG19 is relatively limited (only 41 associative operators with 18 linear operators), while the Transformer has a rich set of operators (360 associative operators with 311 linear operators). Hence, the randomness of selecting associative operators with different computation volumes was downgraded, leading to a more stable performance. This implies that SOTER is more suitable for big complex models with a rich set of operators.

6.3 Security analysis

As mentioned in §3.2, SOTER provides *semantic security* guarantee for protecting model confidentiality: knowing only ciphertexts of parameter-morphed operators, it is infeasible



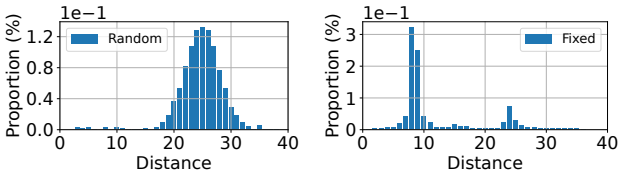
(a) (b)

Figure 7: The model stealing experiments of all baseline systems running (a) VGG19 and (b) Transformer. The red dotted line marks the ground-truth accuracy/BLEU score of VGG19 and Transformer respectively.

for a computationally-bounded adversary to learn about operators' *exact* blinding coins hidden in enclave. It is notable that increasing the number of partitioned morphed operators (with a larger θ) does not degrade the confidentiality because (1) SOTER generates coins for each operator independently in enclave, and (2) an execution result of a morphed operator is concealed to ensure that it reveals neither the coin of that operator, nor the coins of all previous operators (§4.1).

Quantifying information leakage. To evaluate whether our confidentiality analysis given above has a meaningful effect in practice, we applied model stealing attacks [35, 47] on all baseline systems to quantify how much information could leak with that system. A model stealing attack feeds synthetic data to a *victim model* to get an output, and trains a *substitute model* (depicted as Sub_M) with these new samples to mimic the inference behavior of the *victim model*.

- *Setup.* We targeted two popular models, VGG19 and Transformer, running on all baseline systems. We did not run Transformer on eNNclave because eNNclave does not support such a case. We conducted the state-of-the-art Wang's attack [67] for model stealing. Concretely, for the training dataset, with the bounded computational capability assumption [35], we generated synthetic data which composes 10% of total training samples; for the backbone of the substitute model, we adopted VGG13 as the architecture for VGG19 and the standard encoder/decoder architecture for Transformer. We initialized all unknown parameters using a truncated normal distribution with std of 0.02 and learning rate of 0.0001. By training on one Nvidia 2080TI GPU, Sub_{VGG} converged within 150 epochs and Sub_{TF} converged within 13 epochs.
- *Results.* Figure 7 depicts the inference results of the trained substitute model on all baselines. MLcapsule, which shields the entire model within enclave (i.e., $\theta = 0$), achieved the minimum accuracy through the stealing attack. The accuracy results of AegisDNN grew as θ increased, because AegisDNN only runs partial model layers in enclave and exposes other layers with plaintexts to the adversary. With more plaintext revealed in AegisDNN, the accuracy of Sub_{VGG} and BLEU of Sub_{TF} increased dramatically. SOTER, however, achieved comparable results as eNNclave, which directly replaces partitioned operators' parameters with insensitive public parameters. Notably, this implied that SOTER's MTR protocol with semantic security guarantee is effective to provide sufficient model confidentiality.



(a) **w** oblivious fingerprint. (b) **w/o** oblivious fingerprint.

Figure 8: The l_2 distance distribution of randomly sampled fingerprints with or without oblivious fingerprinting technique.

6.4 Robustness to integrity breaches

In this subsection, we begin by investigating the obliviousness of SOTER-generated fingerprints for obfuscating the adversary. Then we evaluated SOTER's robustness to integrity breaches of partitioned operators outside enclave.

Figure 8 shows that, when running with SOTER's oblivious fingerprinting technique, the l_2 distance between randomly sampled fingerprints shares the same form of normal distribution as normal query input (Figure 8a). Hence the fingerprints are oblivious to the adversary. Whereas, when running with fixed fingerprints (Figure 8b), the distribution dramatically changed, hence those fixed fingerprints are observable to the adversary. We evaluated the robustness of SOTER under 10K random perturbation attacks, which randomly perturbs the parameters of partitioned operators. All attacks were successfully detected, and 99.9% attacks were detected with less than ten fingerprints while we incurred zero false positives.

7 Discussion

Model leakage from black-box attacks. Modern black-box attacks [4, 35, 47, 50] use inference APIs of an inference service to learn private information of an offloaded model or even the training dataset, by using the inference results of perturbed input queries. These attacks widely exist in either cloud-side or client-side inference systems [27, 36, 44, 56], as inference systems inevitably open access to arbitrary queries.

Currently, there is no general defense against black-box attacks other than query authentication and rate limiting [35, 47]. But fortunately, defenses against specific attacks are emerging. For instance, Prada [35] uses query distance auditing to mitigate model extraction attacks with adversarial examples; Shokroi et al. [60] use differential privacy to train DNNs that do not leak model owner's sensitive training datasets.

The guarantees of SOTER, like all prior secure inference systems targeting at protecting the model at the edge or the cloud, are complementary to those provided by any defenses. With sufficient engineering effort, these mitigations can be integrated into SOTER to provide even stronger privacy guarantees, and we leave it to future work.

Computation overflow. Multiplying the model parameter with SOTER's blinding coin (§4.1) may, in rare cases, causes

potential computation overflows (we did not find any computation overflow in our experiments in §6). However, even if an overflow occasionally happens, SOTER can immediately detect such overflow by checking if the computed result is INF or -INF [39]. Then, SOTER can simply re-morph the overflowed GPU operator by replacing the blinding coin with a smaller new coin, which can be prepared offline (§4.1).

Trusted GPU. Although there are some promising trusted GPU research systems [3, 32, 46, 65] that can protect model confidentiality and accelerate inference with the strong GPU computing power, these systems are not publicly available as they either require extensive hardware modifications [65] or support only hardware simulators [3, 32, 46]. Thus, in this paper, we consider only publicly available (untrusted) GPU deployments.

Limitation. SOTER requires clients at the edge to be equipped with TEE (e.g., Intel SGX) due to the lack of commonly available trusted GPUs [3,65]. SGX has been pervasively used in existing secure inference systems [27,56] and is commonly available in modern Intel CPUs. While the inherited SGX vulnerabilities of sophisticated side-channel attacks based on timing or cache access patterns still exist [18, 59, 66], we do not consider these attacks currently, and we believe these attacks can be mitigated by integrating with state-of-the-art defenses [18, 19, 55].

8 Conclusion

We present SOTER, the first secure inference system that ensures model confidentiality, low latency, high accuracy with integrity protection for general DNN models. SOTER's MTR protocol carries out cooperative executions between the TEE and GPU. Specifically, SOTER morphs a fraction of associative operators' parameters to run on a GPU, SOTER conceals the execution results on GPU and then restores the real execution results in TEE. SOTER efficiently generates fingerprints to check the integrity of partitioned operators. Evaluation on notable client-side secure inference systems and all prevalent types of DNN models shows that, compared to existing relevant baselines, SOTER achieved comparable strong confidentiality as the TEE-shielding approach, comparable low-latency as the partition-based approach, high accuracy same as insecure inference, and overwhelming high probability of detecting integrity breaches of any partitioned operators. These features make SOTER unique in encouraging enormous model providers to develop powerful models and deploy them on third-party edge devices.

Acknowledgments

We thank the anonymous shepherd and all anonymous reviewers for their helpful comments. This work is supported in part by a Huawei Flagship Research Grant in 2021, a HKU-SCF FinTech Academy R&D Funding Scheme in 2021, HK RGC GRF (17202318, 17207117), HK ITF (GHP/169/20SZ), the Pujiang Lab (Heming Cui is a courtesy researcher in this lab), the HKU and IS-CAS Joint Lab for Intelligent System Software, Hong Kong RGC Project No. PolyU15223918, the Science, Technology and Innovation Commission of Shenzhen Municipality under Grant No.SGDX20201103095408029, and National Natural Science Foundation of China under Grant No.62002151.

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[2] Anshul Aggarwal, Trevor E Carlson, Reza Shokri, and Shruti Tople. Soteria: In search of efficient neural networks for private inference. *arXiv preprint arXiv:2007.12934*, 2020.

[3] Aref Asvadishirehjini, Murat Kantarcioglu, and Bradley Malin. Goat: Gpu outsourcing of deep learning training with asynchronous probabilistic integrity verification inside trusted execution environment. *arXiv preprint arXiv:2010.08855*, 2020.

[4] Buse Gul Atli, Sebastian Szyller, Mika Juuti, Samuel Marchal, and N Asokan. Extraction of complex dnn models: Real threat or boogeyman? In *International Workshop on Engineering Dependable and Secure Machine Learning Systems*, pages 42–57. Springer, 2020.

[5] J Aumasson and L Merino. Sgx secure enclaves in practice–security and crypto review. *Black Hat*, 2016.

[6] Gregory V Bard. The vulnerability of ssl to chosen plaintext attack. *IACR Cryptol. ePrint Arch.*, 2004:111, 2004.

[7] Mihir Bellare, Stefano Tessaro, and Alexander Vardy. Semantic security for the wiretap channel. In *Annual Cryptology Conference*, pages 294–311. Springer, 2012.

[8] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[9] Adith Boloor, Karthik Garimella, Xin He, Christopher Gill, Yevgeniy Vorobeychik, and Xuan Zhang. Attacking vision-based perception in end-to-end autonomous driving models. *Journal of Systems Architecture*, 110:101766, 2020.

[10] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[11] Xusheng Chen, Haoze Song, Jianyu Jiang, Chaoyi Ruan, Cheng Li, Sen Wang, Gong Zhang, Reynold Cheng, and Heming Cui. Achieving low tail-latency and high scalability for serializable transactions in edge computing. In *Proceedings of the Sixteenth European Conference on Computer Systems*, pages 210–227, 2021.

[12] Xusheng Chen, Shixiong Zhao, Ji Qi, Jianyu Jiang, Haoze Song, Cheng Wang, Tsz On Li, TH Hubert Chan, Fengwei Zhang, Xiapu Luo, et al. Efficient and dos-resistant consensus for permissioned blockchains. *Performance Evaluation*, 153:102244, 2022.

[13] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 185–200. IEEE, 2019.

[14] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016(86):1–118, 2016.

[15] Piali Das, Nikita Ivkin, Tanya Bansal, Laurence Rouesnel, Philip Gautier, Zohar Karnin, Leo Dirac, Lakshmi Ramakrishnan, Andre Perunicic, Iaroslav Shcherbatyi, et al. Amazon sagemaker autopilot: a white box automl solution at scale. In *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning*, pages 1–7, 2020.

[16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[17] Shuiguang Deng, Hailiang Zhao, Weijia Fang, Jianwei Yin, Schahram Dustdar, and Albert Y Zomaya. Edge intelligence: The confluence of edge computing and artificial intelligence. *IEEE Internet of Things Journal*, 7(8):7457–7469, 2020.

[18] Ghada Dessouky, Tommaso Frassetto, and Ahmad-Reza Sadeghi. Hybcache: Hybrid side-channel-resilient caches for trusted execution environments. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 451–468, 2020.

[19] Anuj Dubey, Rosario Cammarota, Vikram Suresh, and Aydin Aysu. Guarding machine learning hardware against physical side-channel attacks. *arXiv preprint arXiv:2109.00187*, 2021.

[20] S Durga, Esther Daniel, and S Deepakanmani. Dnn-based decision support system for ecg abnormalities. In *2nd EAI International Conference on Big Data Innovation for Sustainable Cognitive Computing*, pages 331–338. Springer, 2021.

[21] Tarek Elgamal and Klara Nahrstedt. Serdab: An iot framework for partitioning neural networks computation across multiple enclaves. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 519–528. IEEE, 2020.

[22] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[23] David Fiala, Frank Mueller, Christian Engelmann, Rolf Riesen, Kurt Ferreira, and Ron Brightwell. Detection and correction of silent data corruption for large-scale high-performance computing. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2012.

[24] Alessandro Giusti, Dan C Cireşan, Jonathan Masci, Luca M Gambardella, and Jürgen Schmidhuber. Fast image scanning with deep max-pooling convolutional neural networks. In *2013 IEEE International Conference on Image Processing*, pages 4034–4038. IEEE, 2013.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[26] Karan Grover, Shruti Tople, Shweta Shinde, Ranjita Bhagwan, and Ramachandran Ramjee. Privado: Practical and secure dnn inference with enclaves. *arXiv preprint arXiv:1810.00602*, 2018.

[27] Lucjan Hanzlik, Yang Zhang, Kathrin Grosse, Ahmed Salem, Maximilian Augustin, Michael Backes, and Mario Fritz. Mlcapsule: Guarded offline deployment of machine learning as a service. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3300–3309, 2021.

[28] Hanieh Hashemi, Yongqin Wang, and Murali Annavaram. 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*, pages 212–224, 2021.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[30] David A Hensher and William H Greene. The mixed logit model: the state of practice. *Transportation*, 30(2):133–176, 2003.

[31] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[32] Insu Jang, Adrian Tang, Taehoon Kim, Simha Sethumadhavan, and Jaehyuk Huh. Heterogeneous isolated execution for commodity gpus. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 455–468, 2019.

[33] Jianyu Jiang, Xusheng Chen, TszOn Li, Cheng Wang, Tianxiang Shen, Shixiong Zhao, Heming Cui, Cho-Li Wang, and Fengwei Zhang. Uranus: Simple, efficient sgx programming and its applications. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 826–840, 2020.

[34] Neil F Johnson and Sushil Jajodia. Exploring steganography: Seeing the unseen. *Computer*, 31(2):26–34, 1998.

[35] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N Asokan. Prada: protecting against dnn model stealing attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 512–527. IEEE, 2019.

[36] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. Gazelle: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1651–1669, 2018.

[37] Eike Kiltz, Adam O'Neill, and Adam Smith. Instantiability of rsa-oaep under chosen-plaintext attack. In *Annual Cryptology Conference*, pages 295–313. Springer, 2010.

[38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[39] Ignacio Laguna. Fpchecker: Detecting floating-point exceptions in gpu applications. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1126–1129. IEEE, 2019.

[40] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[41] Taegyeong Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, and Junehwa Song. Occlumency: Privacy-preserving remote deep-learning inference using sgx. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–17, 2019.

[42] Wei-Fen Lin, Der-Yu Tsai, Luba Tang, Cheng-Tao Hsieh, Cheng-Yi Chou, Ping-Hao Chang, and Luis Hsu. Onnc: A compilation framework connecting onnx to proprietary deep learning accelerators. In *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 214–218. IEEE, 2019.

[43] Xiaolong Ma, Fu-Ming Guo, Wei Niu, Xue Lin, Jian Tang, Kaisheng Ma, Bin Ren, and Yanzhi Wang. Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5117–5124, 2020.

[44] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2505–2522, 2020.

[45] Robert C Moore and Will Lewis. Intelligent selection of language model training data. 2010.

[46] Lucien KL Ng, Sherman SM Chow, Anna PY Woo, Donald PH Wong, and Yongjun Zhao. Goten: Gpu-outsourcing trusted execution of neural network training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 14876–14883, 2021.

[47] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Prediction poisoning: Towards defenses against dnn model stealing attacks. *arXiv preprint arXiv:1906.10908*, 2019.

[48] Jeffrey Overbey, William Traves, and Jerzy Wojdylo. On the keyspace of the hill cipher. *Cryptologia*, 29(1):59–72, 2005.

[49] Daniele Palossi, Antonio Loquercio, Francesco Conti, Eric Flamand, Davide Scaramuzza, and Luca Benini. A 64-mw dnn-based visual navigation engine for autonomous nano-drones. *IEEE Internet of Things Journal*, 6(5):8357–8371, 2019.

[50] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[51] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[52] Benny Pinkas and Tzachy Reinman. Oblivious ram revisited. In *Annual cryptology conference*, pages 502–519. Springer, 2010.

[53] Sandro Pinto and Nuno Santos. Demystifying arm trustzone: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 51(6):1–36, 2019.

[54] Ji Qi, Xusheng Chen, Yunpeng Jiang, Jianyu Jiang, Tianxiang Shen, Shixiong Zhao, Sen Wang, Gong Zhang, Li Chen, Man Ho Au, et al. Bidl: A high-throughput, low-latency permissioned blockchain framework for datacenter networks. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 18–34, 2021.

[55] Sajin Sasy, Sergey Gorbunov, and Christopher W Fletcher. Zerotrace: Oblivious memory primitives from intel sgx. In *NDSS*, 2018.

[56] Alexander Schlögl and Rainer Böhme. ennclave: Offline inference with model confidentiality. In *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*, pages 93–104, 2020.

[57] Jean Senellart, Dakun Zhang, Bo Wang, Guillaume Klein, Jean-Pierre Ramatchandirin, Josep M Crego, and Alexander M Rush. Opennmt system description for wnmt 2018: 800 words/sec on a single-core cpu. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 122–128, 2018.

[58] AMD SEV-SNP. Strengthening vm isolation with integrity protection and more. *White Paper, January*, 2020.

[59] Tianxiang Shen, Jianyu Jiang, Yunpeng Jiang, Xusheng Chen, Ji Qi, Shixiong Zhao, Fengwei Zhang, Xiapu Luo, and Heming Cui. Daenet: Making strong anonymity scale in a fully decentralized network. *IEEE Transactions on Dependable and Secure Computing*, 2021.

[60] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321, 2015.

[61] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[62] Florian Tramer and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287*, 2018.

[63] Chia-Che Tsai, Donald E Porter, and Mona Vij. Graphene-sgx: A practical library os for unmodified applications on sgx. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 645–658, 2017.

[64] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[65] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. Graviton: Trusted execution environments on gpus. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 681–696, 2018.

[66] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2421–2434, 2017.

[67] Wenxuan Wang, Bangjie Yin, Taiping Yao, Li Zhang, Yanwei Fu, Shouhong Ding, Jilin Li, Feiyue Huang, and Xiangyang Xue. Delving into data: Effectively substitute training for black-box attack. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4761–4770, June 2021.

[68] Yecheng Xiang, Yidi Wang, Hyunjong Choi, Mohsen Karimi, and Hyoseung Kim. Aegisdnn: Dependable and timely execution of dnn tasks with sgx. In *2021 IEEE Real-Time Systems Symposium (RTSS)*, pages 68–81. IEEE, 2021.

[69] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[70] Chao Zhang and Philip C Woodland. Parameterised sigmoid and relu hidden activation functions for dnn acoustic modelling. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[71] Jiliang Zhang, Wuqiao Chen, and Yuqi Niu. Deepcheck: A non-intrusive control-flow integrity checking based on deep learning. *arXiv preprint arXiv:1905.01858*, 2019.

[72] Shixiong Zhao, Fanxin Li, Xusheng Chen, Xiuxian Guan, Jianyu Jiang, Dong Huang, Yuhao Qing, Sen Wang, Peng Wang, Gong Zhang, et al. v pipe: A virtualized acceleration system for achieving efficient and scalable pipeline parallel dnn training. *IEEE Transactions on Parallel and Distributed Systems*, 33(3):489–506, 2021.

[73] Shixiong Zhao, Fanxin Li, Xusheng Chen, Tianxiang Shen, Li Chen, Sen Wang, Nicholas Zhang, Cheng Li, and Heming Cui. Naspipe: high performance and reproducible pipeline parallel supernet training via causal

synchronous parallelism. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 374–387, 2022.