FlexGE: Towards Secure and Flexible Model Partition for Deep Neural Networks

Xiaolong Wu, Aravind Kumar Machiry, Yung-hsiang Lu, and Dave (Jing) Tian

Purdue University, West Lafayette, IN, USA
{wu1565,amachiry,yunglu,daveti}@purdue.edu

Abstract. Proprietary deep neural network (DNN) models are being deployed in the cloud nowadays. With the increased usage of AI accelerators in the cloud, there is a growing need for privacy protection for outsourced deep learning computations. Existing works use a Trusted Execution Environment (TEE) to shield DNN partitions, which puts a subset of the DNN model in TEEs and offloads the rest of the computation on GPUs. However, these solutions use fixed security primitives and model partition policy, which precludes per-model specialization to balance the security and performance requirements. In this paper, we present a novel on-demand model inference system, FlexGE, that partitions the DNN model between TEE and GPU accelerator with programmable partition policies and protection primitives based on the user's configuration. FlexGE achieves this by tailoring the protection profile as well as the model partition policy and partitioning the model at *build time* as opposed to design time. We implement FlexGE using Darknet and GEVisor, and evaluate it on five popular DNNs. Our evaluation shows that FlexGE is flexible and outperforms the state-of-the-art in terms of security and performance.

Keywords: Model Partition · GPU TEE · DNN.

1 Introduction

Deep neural networks (DNNs) are widely used and often require excessive computational resources. Meanwhile, cloud computing makes deep learning more accessible, flexible, and cost-effective while allowing developers to build deep learning algorithms faster. Artificial Intelligence as a Service (AIaaS) [26] in the cloud uses pre-trained models and enables vendors to reduce the risk and hardware investment of their customers. At the same time, the rapid increase in the complexity of the software stack in the cloud expands the attack surface for machine learning applications. This raises privacy concerns about DNN computations in untrusted environments, in particular, for DNN models outsourced by a client to a remote cloud server. Attackers are financially motivated to steal these models derived from expensive training with a significant engineering effort. As a result, leakage of such proprietary models can cause severe financial loss and security issues.

To make matters worse, existing proprietary models are found to be not well protected, especially since GPU as a Service (GPUaaS) [12] is prominent in the cloud. Using GPUs to accelerate DNN makes model privacy protection even more challenging. Nevertheless, directly applying TEEs to protect entire DNN models presents significant challenges, as most commercial GPUs (e.g., V100 and A100) lack TEE functionality. While some high-end GPUs, such as those based on the Nvidia Hopper architecture [1], support confidential computing, their cost remains prohibitively high for typical model users. For instance, an Nvidia H100 GPU is over 15 times more expensive than a GeForce RTX 4090[1]. Therefore, researchers propose Trusted Execution Environment (TEE) shielded DNN partition [47], [46], [54], which puts a subset of privacy-sensitive and critical components of the DNN model in TEEs and offloads the rest computation on GPUs. However, these approaches suffer from several drawbacks. First, they preclude per-model specialization to balance the security and performance requirements. The rigid use of security primitives in these techniques permanently locks the design into a fixed combination of security primitives that is likely to result in suboptimal security/performance in many scenarios. Second, they fix the model partition policy for each model, losing the flexibility to explore the advantages of different partition policies. As a result, when the protection offered by a hardware security primitive breaks down (e.g., physical attacks [28]), or the protection offered by a security primitive is too expensive (e.g., the homomorphic encryption may counteract the performance benefit of GPU acceleration), or the model partition policy is not optimal, it is difficult and costly to decide how it should be replaced.

When multiple protection mechanisms or model partitioning policies are available for a given model, selecting the most suitable configuration depends on various factors and is best deferred until deployment time. This leads to one important research question: ***Is it possible to switch between different protection primitives and model partition policies at deployment time, avoiding the lock-in model partition that characterizes the status quo?*** Our answer is that protection profile as well as model partition policy can cost-efficiently be tailored towards a specific DNN model at *build time*, as opposed to design time.

To verify this idea, we design *FlexGE*, the first system framework that supports different model partition policies and protection primitives and enables flexible fine-grained model partition at *build time* via delegating partial computations to different back-ends, potentially in different protection domains, with different protection mechanisms. The challenge is how to instantiate protection primitives for each model partition, what partition granularity to use between different partitions, and what software hardening mechanisms should be applied to mitigate the potential vulnerabilities of hardware primitives. To that aim, we abstract the common operations required when partitioning arbitrary models behind a generic API that is used to retrofit an existing DNN model into FlexGE. This API reduces the manual porting effort of the existing DNN model by partitioning weight matrix data using annotations. These annotations, alongside other abstract source-level constructs, are replaced at build time by FlexGE to instantiate a given configuration.

There exists a significant gap in security and performance between TEE and GPU. This large disparity presents a substantial trade-off space with potential for optimization. The configuration space enabled by model partition, illustrated in Figure 1, is large and almost impossible for a non-expert user to explore manually. This leads to the second research question we explore: ***how to guide a typical user to navigate the vast configuration space unlocked by FlexGE?*** To answer this, we propose a quantitative analysis framework that formally defines the security-performance trade-off across various model partitioning poli-

[1] At February 2025, the price of an H100 GPU is about $25,000, while the price of a GeForce RTX 4090 is less than $1600

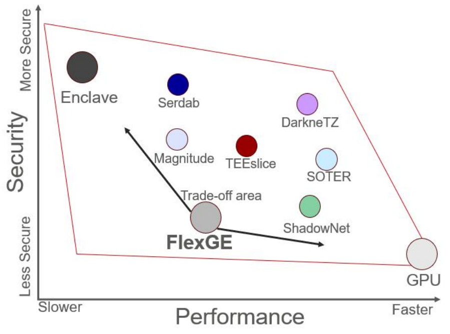


Fig. 1: Design space of DNN model partition.

cies and includes an automatic algorithm for selecting the optimal configuration. Existing works only consider a model-level partition policy that does not reflect the effect of system protection profile variation (i.e., strong protection primitives offer higher security but can reduce performance, while weaker protection primitives deliver better performance at the cost of reduced security). We address the security-performance separation gap and argue that model partitioning should incorporate not only model-level partitioning policies but also system-level protection primitives. However, this design introduces an additional challenge: while a weak protection profile can reduce performance overhead, it also risks an extreme scenario where a single compromised protection primitive could lead to a complete system crash. Therefore, we propose ***Configuration Space Layout Randomization (CSLR)***, a co-design method for model and system co-configuration, hardening both the partition and the protection primitives. As the workload has been partitioned and delivered to different back-ends, FlexGE makes it impossible for the attackers to collect and piece together all the information to complete the attack, as it requires the attackers to breach all back-ends with different protection primitives. This design makes the system robust and resilient to future attacks, assuming the partitioning mechanism fails.

We have developed a FlexGE prototype that integrates Intel Software Guard Extensions (SGX) to support CUDA kernel I/O encryption and VM/EPT-based GPU I/O protection through a hypervisor, extending the security boundary of SGX from the CPU to the GPU, as well as two hardening mechanisms (CFI [2] and ASLR [41]). Our evaluation of several deep neural networks demonstrates the potential security versus performance tradeoff space unlocked by FlexGE, e.g., exploring over 80 configurations for AlexNet. Finally, we demonstrate that under an equivalent requirement, FlexGE outperforms the state-of-the-art in terms of security and performance.

The contributions of this paper are as follows.

- We design FlexGE, a model inference system for DNNs that supports arbitrary partitioning policies between TEE and GPU, allowing flexible delegation of partial computations to various back-ends with different protection mechanisms (§4).
- We propose a novel *build time* model partitioning mechanism that enables flexible configuration and fine-grained partitioning of models (§4.2).

– We propose a novel *Configuration Space Layout Randomization (CSLR)* mechanism to enhance the security of the model inference system (§4.3).
– We propose *Quantitative Metrics* for characterizing both security and performance, and the corresponding algorithm to identify the optimal configuration for each of the partition policies (§5).
– Our evaluation demonstrates FlexGE's security and flexibility for real-world usage on a diverse range of DNN architectures (§7).

2 Background

2.1 Deep Neural Network

Convolutional neural network (CNN) [35] is a class of deep neural networks that typically consists of an input and an output layer with a sequence of linear and non-linear layers stacked in between. The linear layers include convolutional layers and fully connected layers; the non-linear layers include activation and pooling layers.

Convolutional Layer. The parameters of a convolutional layer consist of a set of learnable filters. Each filter is characterized by the width, height, and depth of the receptive field. The depth must be equal to the number of channels of the input feature map. Let `h`, `b`, `d` represent the height, width, and depth of the filter ω, respectively, and (`x`, `y`) refer to the coordinates in the 2D output feature map. Formally, the convolution operation on a given image *I* with filter ω can be described as follows:

$$CONV(I,\omega)_{x,y} = \sum_{i=1}^{h}\sum_{j=1}^{b}\sum_{k=1}^{d}\omega_{i,j,k} I_{x+i-1,y+j-1,k} \tag{1}$$

Let `X` and `Y` denote the input and output, respectively, and `W`= $[\omega_1,\ldots,\omega_n]^T$ be the convolution filter. The corresponding convolutional layer is thus given by:

$$Y = Conv(X, W^T) \tag{2}$$

Fully Connected Layer. The dense layer connects every input node to every output node. It can be implemented as a convolutional layer with both filter height and width: 1. For example, a dense layer connecting *n* input to *m* output can be viewed as a convolutional layer that has *m* filters of size (`1`, `1`, `n`).

Residual Neural Network (a.k.a. Residual Network, ResNet) [14] is another deep learning model extended from CNN for addressing the vanishing gradient problem of CNN to some extent, in which the network skips connections that perform identity mappings, merged with the layer outputs by addition.

Residual layer. Suppose the output of linear operation of layer `l+2` is z^{l+2}, and the ReLU [30] function is `g`. For a 2 layer skip residual layer, instead of a^{l+2} = g(z^{l+2}), the output of layer `l+2` is:

$$a^{l+2} = g(z^{l+2} + a^{l}) \tag{3}$$

Table 1: Comparison between existing works and FlexGE.

	Model Privacy	Cloud/Device	Backend Protection	Partition Policy	Flexible Configuration	Fine Grained
Slalom [47]	-	Cloud	Obfuscation	Linear Layers	-	-
DarkneTZ [29]	✓	Device	-	Deep layer	-	-
Serdab [10]	✓	Device	-	Shadow layer	-	-
Magnitude [16]	✓	Device	-	Large-Mag. Weights	-	✓
SOTER [43]	✓	Device	-	Intermediate Layers	-	-
ShadowNet [46]	✓	Device	Obfuscation	Non-Linear Layers	-	-
TEEslice [54]	✓	Device	Obfuscation	Slice	-	✓
FlexGE	✓	Cloud (or Device)	EPT and Cryptographic (Extensible)	All	✓	✓

2.2 Intel SGX

Intel Software Guard Extensions (SGX) [32] is a Trusted Execution Environment (TEE) that ensures the confidentiality and integrity of user code and data. SGX allows a process to allocate a protected memory region, i.e., an enclave, within its address space. Intel SGX affords the enclave hardware protections against CPU-based attacks but is not designed to secure the communication between the enclave and external devices attached to the system; that is, SGX's security boundary is only within the CPU. One of the main reasons for this is that all external device communication is traditionally handled by the OS. In particular, the device drivers (loaded onto the OS) create and maintain a memory-mapped I/O channel between a program and the intended device(s).

2.3 GPU Software Stack

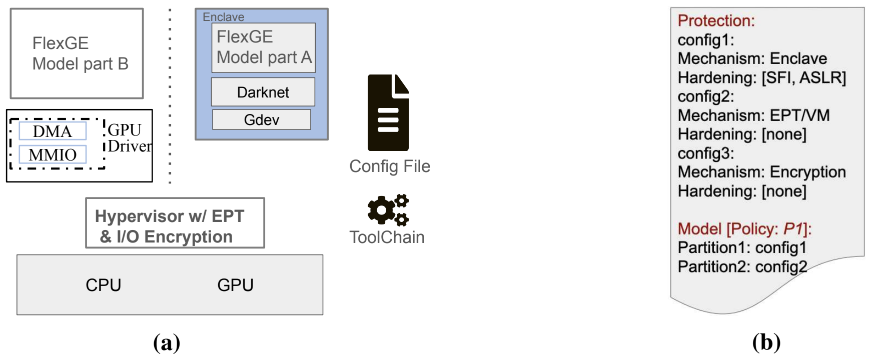
The GPU device driver is responsible for the creation, deletion, and upkeep of a communication channel with the GPU. Gdev [19], an open-source GPU stack consists of an implementation of the CUDA driver API and `libdrm` and `nouveau`, which implement the user- and kernel-space GPU device driver.

3 Motivation

Based on the model partition policy, existing TEE-based DNN model partition research can be classified into five categories in Table 1, including shielding the deep layer into TEE, such as DarkneTZ [29], shielding the shallow layer into TEE, such as Serdab [10], putting intermediate layers to enclave like SOTER [43], putting fine-grained sub-layers into the enclave, such as Magnitude [16] and TEEslice [54]), and shielding no-linear layers into the enclave, e.g., ShadowNet [46].

Slalom [47] outsources the linear layers to the GPU for acceleration with masked inputs while keeping the other layers inside SGX. However, Slalom protects the user input privacy but not the model weights from the untrusted cloud server. State-of-the-art (i.e., TEEslice [54]) proposes a training-before-partition strategy, which involves expensive training on a private model.

We make the following key observation from Table 1: existing DNN model partition policies are fixed at the design time. This motivates us to design a flexible DNN model partition framework corresponding to different users' diverse security and performance requirements. FlexGE seeks to enable users to easily and securely switch between TEE and GPU with different protection primitives and partition policies at deployment time.



```
Protection:
config1:
Mechanism: Enclave
Hardening: [SFI, ASLR]
config2:
Mechanism: EPT/VM
Hardening: [none]
config3:
Mechanism: Encryption
Hardening: [none]

Model [Policy: P1]:
Partition1: config1
Partition2: config2
```

Fig. 2: (a) FlexGE Architecture. (b) Configuration file.

4 Design

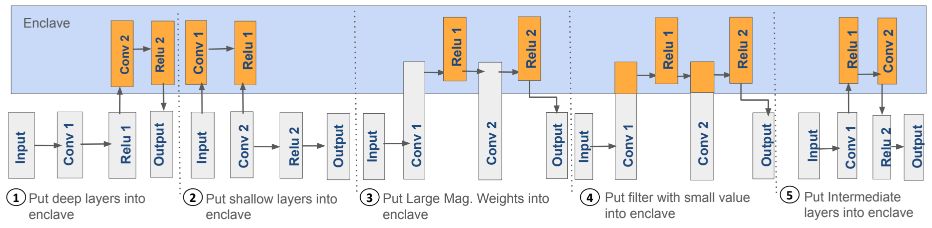
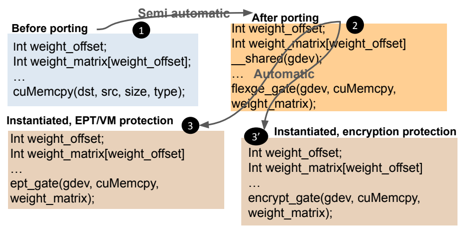

Enclave
Input
Conv 1
Relu 1
Conv 2
Relu 2
Output
1 Put deep layers into enclave
2 Put shallow layers into enclave
3 Put Large Mag. Weights into enclave
4 Put filter with small value into enclave
5 Put Intermediate layers into enclave

Fig. 3: Model partition policy

We now present an overview of the design of FlexGE in Figure 2(a). FlexGE is composed of SGX-based CPU TEE and GPU, and two GPU I/O protection backends between CPU TEE and GPU through a hypervisor and GPU runtime (i.e., Gdev): EPT/VM and CUDA kernel Encryption. The encryption backend provides strong but expensive I/O channel protection, while the EPT-based backend provides weaker but more performant protection. The CPU TEE (i.e., enclave) and EPT/VM backend can be hardened via techniques such as Control-Flow Integrity (CFI) and address space layout randomization (ASLR). The security configuration is provided at build time in a configuration file provided by the user including different protection configurations of various protection primitives and model configuration with selected partition policy and protection configurations, and FlexGE's tool-chain produces a DNN model partition implementation with the desired security characteristics. An example of such a configuration file is given in Figure 2(b).

FlexGE's tool-chain enables calling external security functions via abstract gates and data sharing between the enclave and GPU via abstract code annotations. Gates and annotations form an API used to partition a DNN model in FlexGE. A given partition configuration in Figure 2(b) is automatically replaced by our toolchain with a particular implementation at build time.


Semi automatic
Before porting
After porting
Automatic
Instantiated, EPT/VM protection
Instantiated, encryption protection
1
2
3
3'

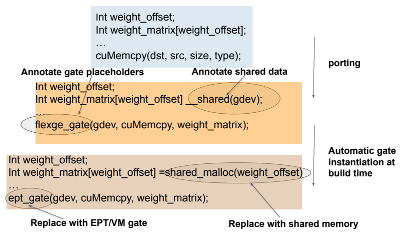
```
Int weight_offset;
Int weight_matrix[weight_offset];
…
cuMemcpy(dst, src, size, type);
```

```
Int weight_offset;
Int weight_matrix[weight_offset]
__shared(gdev);
…
flexge_gate(gdev, cuMemcpy,
weight_matrix);
```

```
Int weight_offset;
Int weight_matrix[weight_offset]
…
ept_gate(gdev, cuMemcpy,
weight_matrix);
```

```
Int weight_offset;
Int weight_matrix[weight_offset]
…
encrypt_gate(gdev, cuMemcpy,
weight_matrix);
```

(a)


porting
Annotate gate placeholders
Annotate shared data
Automatic gate instantiation at build time
Replace with EPT/VM gate
Replace with shared memory

```
Int weight_offset;
Int weight_matrix[weight_offset];
…
cuMemcpy(dst, src, size, type);
```

```
Int weight_offset;
Int weight_matrix[weight_offset] __shared(gdev);
…
flexge_gate(gdev, cuMemcpy, weight_matrix);
```

```
Int weight_offset;
Int weight_matrix[weight_offset] =shared_malloc(weight_offset)
…
ept_gate(gdev, cuMemcpy, weight_matrix);
```

(b)

Fig. 4: (a) FlexGE code transformations. First, users manually annotate partitioned data with shared (Gdev) annotation, and gate placeholders are automatically inserted. At build time, API primitives are automatically replaced with the chosen mechanism. (b) EPT Backend Instantiation.

4.1 Fine-grained Model Partition Policy

The goal of FlexGE's model partition policy is to support all the policies in existing works. We omit the approach of ShadowNet [46] because it does not require configurations. For sliced-based partition [54], we instead propose a fine-grained filter-based partition policy to represent it. Totally, we provide five partition policies (*P*) for a DNN model including layer-based and fine-grained filter-based partition, as shown in Figure 3.

① According to the layer depth and closeness to the output layer in the TEE, two deepest layers (Conv2 andReLU2) are put into the enclave (*P1*).

② According to the layer depth and closeness to the input layer in the TEE, two shallowest layers (Conv1 and ReLU1) are put into the enclave (*P2*).

③ According to the absolute weight value, partial convolution layers with large-magnitude weights and ReLU layers are put into the enclave (*P3*).

④ According to the filter value, ReLU1 and Conv2 with the filter of small values put into the enclave (*P4*).

⑤ Putting randomly chosen intermediate layers in the TEE. In Figure 3, ReLU1 and Conv2 as the random-selected layers are put into the enclave (*P5*).

4.2 Build-time Model Partition Mechanism

API and Build-time Instantiation. The GPU I/O backend security functions are made through abstract call gates that are instantiated at build time. *Shared data* (e.g., weight matrix) between CPU TEE and GPU is marked using compiler annotations, used at build time to instantiate a given DNN model partition policy. FlexGE performs replacements using source-to-source transformations, which gives it a better performance advantage over heavyweight runtime abstraction interfaces.

Call Gates. In FlexGE, security function calls (EPT/VM, and encryption) are represented in the source code by abstract call gates. At build time, as part of the transformation phase, abstract call gates are replaced with a specific implementation. For instance, when the DNN model is configured to be in the GPU with EPT/VM backend, the call gate performs VM enter call. Figure 4(a) presents an example of gates from the porting (step ❷) to the replacement by the toolchain (❸ and ❸').

Porting existing Gdev code to FlexGE consists of marking call gates, which can be automated: Knowing the system's control flow graph, static analysis determines whether a procedure call performs GPU I/O access and, if so, performs a syntactic replacement of the function call with a call gate instead. However, in our current implementation, we manually annotate the Gdev functions that perform I/O access (MMIO and DMA) with the abstract call gate. The toolchain will then generate wrappers enclosing the implementations of the functions in the appropriate call gates.

EPT/VM Backend. The EPT/VM backend is based on a small hypervisor. FlexGE's hypervisor provides GPU I/O protection, including MMIO and DMA, with an EPT mechanism. The hypervisor ensures that the DMA and command buffers are inaccessible to an attacker who attempts to use the CPU to access these memory regions with EPT trapping. In particular, the hypervisor maintains memory region mapping tables containing the virtual and physical address pairs of both MMIO and DMA memory regions per enclave within a reserved memory region and traps access to these regions for access control. Compared with the encryption backend, the EPT-based backend provides weaker but more performant protection. FlexGE provides two variants of EPT backend: EPT and EPT_2M with huge page optimization [50].

Data Sharing. The EPT backend relies on shared memory areas to share model weights across VMs in EPT and EPT_2M backends. Areas are always mapped at the same address in the VMs so that pointers to/in shared model matrix structures remain valid. Each VM manages its own portion of the shared memory area to avoid the need for complex multithreaded bookkeeping.

Figure 4(b) shows the instantiating procedure of an EPT protection backend. Using the programmable API, the user would first annotate shared data and add gate placeholders. Then, at build time, FlexGE would replace the annotated shared data with a shared memory location and replace the gate placeholder with an EPT/VM gate.

Encryption Backend. While FlexGE could use EPT to prevent unauthorized accesses to DMA from privileged software with better performance, it cannot stop attackers from attaching probes to the I/O bus and snooping the traffic to steal the code and data. Fundamentally, an EPT/VM protection would have to downgrade the threat model by excluding potential physical attacks.

Instead, we design a *crypto CUDA kernel* to augment GPU with *in-device* Diffie-Hellman (DH) key exchange, encryption, and decryption, enabling a crypto-secured DMA communication channel between enclaves and GPU. A user enclave and the GPU first performs local attestation to verify each other. Once they establish the trust through attestation, the crypto kernel is loaded from the enclave to the GPU using MMIO, which is protected by the hypervisor. Once loaded, the crypto kernel within the GPU launches a DH key exchange to establish a shared secret key within the GPU memory and the enclave. A physical bus snooping attack could observe our crypto kernel and even the DH key exchange but not the shared secret established after. From this point, enclaves can encrypt the code and data before exposing them in DMA. GPU will decrypt them inside the device and encrypt the results again before writing them into DMA. Assuming an authenticated encryption scheme, e.g., AES-GCM, we will not need a hypervisor anymore since any tampering will fail the decryption and thus be detected.

Software Hardening. The flexible DNN model secure protection provided by FlexGE allows enabling/disabling software hardening (SH) such as CFI, etc. Isolating DNN model layers with SH from layers without it allows the former to maintain the guarantees offered by SH. This flexibility allows for alleviating the performance impact of SH by enabling it only for sub-layers of a DNN model. Our prototype currently uses address space layout randomization (ASLR) and CFI. ASLR hides the memory layouts from adversaries by randomly placing code and data in runtime, which makes it hard for the victim code or data to find the location so that control-flow hijack or data-flow manipulation attacks are prevented. FlexGE employs fine-grained randomization by splitting the code section of SGX enclave into a set of randomization units [41]. For CFI, our enforcement is performed for all control transfer instructions for the program in the SGX enclave, including indirect branches as well as return instructions. In the case of indirect branches, we add masking operations to the destination so that it only points to one of the randomization unit's entry points. In the case of a return instruction, we replace it with two equivalent instructions, *pop reg* and *jmp reg*, where *reg* can be any available register. Then, the second *jmp* instruction is instrumented similarly to indirect branch instructions.

4.3 Configuration Space Layout Randomization (CSLR)

Inspired by Address Space Layout Randomization (ASLR) [42], we introduce Configuration Space Layout Randomization (CSLR) for FlexGE. The security of FlexGE is directly related to the entropy [4] of the CSLR implementation. Low entropy would allow an attacker to brute-force the entire search space and bypass CSLR if they can repeatedly attempt exploits. To counter this, FlexGE utilizes fine-grained randomization schemes to maximize CSLR entropy.

For DNN model-level fine-grained CSLR, FlexGE adopts a smaller partition size, called a randomization unit. These units have a fixed, configurable size (e.g., in a 20-layer DNN, a randomization unit size of 5 results in 4 partitions). The overall configuration space is determined by permutations between model partitions and back-end protection mechanisms. Consequently, CSLR entropy is proportional to the number of permutations of model partition sizes and available back-end protections. FlexGE is designed to support extensible back-end protection mechanisms, further enhancing system security.

5 Design Space Navigation

We propose a quantitative method (formal definition) given a configuration.

5.1 Quantitative Formalization

To systematically find the optimal model partition policy for a DNN model given a system configuration, we formalize the problem as an optimization problem. Formally, let S be a model partition solution that splits a DNN model into enclave and GPU-offloaded portions. Let P denote a configuration policy of S that specifies to what degree the model is put into an enclave. We define a security score and a performance score for a specific configuration, Security(P) and Performance(P), which quantify the security risk and performance cost of P, respectively. We define $Security_{max}$ as the security risk baseline of setting, which puts the whole DNN model in an enclave. $Security_{max}$ denotes the lower bound of the security risk (the strongest protection). We also define $Security_{min}$ as the security risk upper-bound of the setting, which puts all the layers out of the enclave and off-loads to GPU. $Security_{min}$ denotes the upper bound of the security risk (the weakest protection). Then, given the security requirement $\varDelta$, we formulate the optimal configuration $P^{\star}$ that satisfies:

$$P^{\star} = \underset{|Security(P)\text{-}Security_{max}|<\varDelta}{argmin} \quad Performance(P) \tag{4}$$

We define the security score, Security(P), using model stealing accuracy [37], which calculates how much test samples can be correctly classified by the attacker's surrogate model (We detail the model stealing procedure in section 7). Achieving high accuracy is a primary goal of model stealing attacks [54]. Given the total test number N_{total}, and the sample number can be correctly classified $N_{correct}$. Security(P) is defined as:

$$Security(P) = (N_{correct}(P)/N_{total})\% \tag{5}$$

We define the performance score, Performance(P), using performance overhead. Suppose the inference latency of the DNN layers conducted in the enclave is $T\_Enclave$, and the total inference latency corresponding to the configuration P is T_P, then we define the Performance(P) as the ratio of inference latency in the TEE over the total inference latency of the DNN model:

$$Performance(P) = (T\_Enclave/T\_P)\% \tag{6}$$

Thus, a larger Performance(P) indicates fewer computations are offloaded on GPUs, leading to higher performance overhead.

5.2 Optimal Configuration Selection

For each of the five policies in section 4.1, we iterate possible configurations to identify $P^{\star}$. In particular, for the policy that shields deep layers (*P1*), we use a lightweight, greedy clustering algorithm that assign layers into clusters. We begin the algorithm by placing the last layer into a cluster; we then proceed to perform repeated clustering one more layer operations until an assignment of layers produces the optimal $P^{\star}$. We shield different numbers of consecutive "deep" layers starting from the output layer with TEEs. Similarly, for(*P2*), which shields shallow layers, we put different amounts of consecutive layers starting from the DNN input layer. For ResNet models, we use the residual layers as the dividing boundaries. For VGG models and AlexNet models, we use convolution layers as boundaries. For shielding large-magnitude weights (*P3*), the number of protected weights is controlled by a configuration parameter mag_ratio. We gradually set range of mag_ratio from 0.1 to 0.9, until finding the optimal $P^{\star}$. Similarly, for putting filter with small value into enclave (*P4*), we set the filter_ratio. For shielding intermediate layers (*P5*), the number of shielded layers is also defined by a configuration parameter, inter_ratio. Similarly, we set the range of inter_ratio from 0.1 to 0.9. For *P3*, *P4*, and *P5*, setting mag_ratio, filter_ratio, and inter_ratio to 0 represents the $Security_{min}$ while setting the parameters to 1 is the $Security_{max}$.

6 Implementation

We implement FlexGE based on Darknet deep learning framework [39] and a tiny hypervisor: GEVisor [50]. We implement a build-time source transformations toolchain. We use the Vembyr PEG parser generator [36] to automate the development of a source-to-source translator. Vembyr provides a convenient extension interface that allows us to construct an abstract syntax tree (AST). The final compilation pass converts the AST into a concrete syntax tree (CST) to print out the C code. FlexGE's toolchain performs source transformations to (1) instantiate abstract gates, (2) instantiate data sharing code, (3) generate linker scripts, and (4) generate additional code in Gdev according to backend-provided GPU I/O protection recipes.

7 Evaluation

7.1 Evaluation Setting

Our experimental machine uses an Intel i7- 8700K 4.7GHz CPU with Intel SGX (SDK v2.0), 6 cores, and 32GB of main memory. We use a NVIDIA GeForce GTX TITAN Black GPU with 2,880 CUDA cores and 6,144MB GDDR5 384-bit memory.

Datasets. We use four different datasets in our experiments, including CIFAR10 [21], CIFAR100 [21], STL10 [8], and UTKFace [53].

Models. The benchmark models include ResNet18 [14], VGG16_BN [44], AlexNet [22], ResNet34 and VGG19_BN. We use the public models [34] as initialization for all the experiments.

7.2 Case Study

In this section, we explore the vast performance/security design space enabled by FlexGE. In Figure 5, we first partition the AlexNet model into four partitions and then plot the inference time with the CIFAR100 data set for each configuration. We variate different partitions with different protection backend configurations, while all of them apply policy P4. For example, having partition1 in an enclave with hardening, partition2 in an enclave with no hardening, partition3 in EPT_2M (EPT with 2M huge page optimization [50]) protection without hardening, and partition4 in EPT with hardening leads to a 30.8 seconds inference latency. Overall, we observe that FlexGE enables a very wide range of security configurations with significant performance variation. The configuration with all four partitions in an enclave with hardening performs worst, with 119 seconds of inference latency. It is worth mentioning that the red color does not attach with hardening because the GPU I/O encryption protection cannot come with software hardening. Existing approaches assume a one-size-fits-all all configuration are therefore suboptimal; in contrast, FlexGE enables users to easily navigate the security/performance trade-off inherent in their application.

CSLR Security Analysis. AlexNet employs an 8-layer CNN with five convolutional layers, two fully connected hidden layers, and one fully connected output layer. The randomization unit is 2 with 4 partitions. The configuration space in our study is 80. As a result, the entropy of CSLR is proportional to $80 * 4!$. It is very hard for an attacker to breach such a large randomization space.

Observation 1: *Even with a small DNN architecture, FlexGE can provide strong security protection with CSLR.*

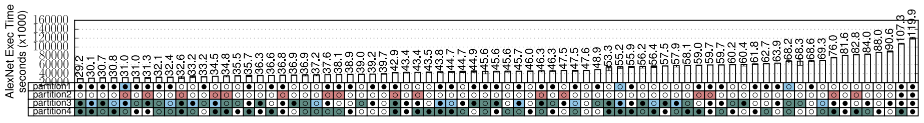


Fig. 5: AlexNet execution time for a range of configurations. Partitions are on the left. Software hardening can be enabled ● or disabled ○ for each partition. The white/blue/red/green color indicates the partitions are placed into. white : enclave, red : encryption, blue : EPT_2M, green : EPT.

7.3 Quantitative Evaluation for Model Partition Policies

Model Stealing Attack. The attacker first analyzes the target defense scheme and then infers the architecture of the protected model based on the offloaded part, and the model output with existing techniques [6] [7] to get an initialized model. Then, the attacker chooses a public model (with the same or an equivalent architecture), trains this model with queried data, and outputs the surrogate model (the recovered victim model). Lastly, the attacker transports the model weights in the offloaded part of the surrogate model to the corresponding parts of the initialized model.

Model Stealing Accuracy. We test the five partition policies and report the evaluation results over five models (AlexNet, ResNet18, ResNet34, and VGG16_BN VGG19_BN) in Table 2 (total 20 cases). As aforementioned, we also report the baseline settings ("$Security_{min}$" and "$Security_{max}$") for comparison. For each partition policy P in section 4.1, we iterate possible configurations to identify $P^{\star}$. For ResNet models, we use the residual layers as the dividing boundaries. For VGG models and AlexNet models, we use convolution layers as boundaries. For each model and dataset, we mark the highest MS attack accuracy in red and the lowest accuracy in blue. The results show that TEESlice and FlexGE achieve the lowest MS attack accuracy. Due to the CSLR mechanism, FlexGE achieves the lowest MS attack accuracy in more cases than TEESlice. Moreover, it maintains the lowest MS attack accuracy across all policies.

Observation 2: *All policies have the potential to achieve the lowest MS attack accuracy.*

Optimal Configuration. For FlexGE, we use the system configuration: Enclave with SFI and ASLR software hardening, and GPU CUDA kernel encryption I/O protection and then measure the values of the Performance($P^{\star}$) as defined in Equation 4 (the smallest value of performance overhead to achieve $Security_{max}$) for a given model partition policies P4. Table 3 reports the performance overhead to achieve $Security_{max}$ for each setting for MS. We highlight the results of FlexGE in orange, while marking the lowest performance($P^{\star}$) of other works in blue and the highest values in red. Overall, Table 3 implies that the performance overhead to achieve $Security_{max}$ is distinct across different DNN models and datasets. For example, to protect AlexNet from MS by putting deep layers into enclave,

		$Security_{min}$	P1(DarkneTZ | FlexGE)	P2(Serdab | FlexGE)	P3(Magnitude | FlexGE)	P4(TEEslice | FlexGE)	P5(SOTER | FlexGE)	$Security_{max}$
AlexNet	C10	85.59%	70.95% | 34.64%	62.86% | 30.32%	60.79% | 29.88%	35.84% | 32.67%	70.76% | 33.98%	20.30%
AlexNet	C100	62.33%	38.99% | 19.22%	42.33% | 20.53%	47.90% | 22.65%	21.97% | 21.21%	50.37% | 24.99%	11.77%
AlexNet	S10	75.44%	73.55% | 35.22%	66.85% | 31.41%	68.49% | 31.89%	33.27% | 33.12%	37.98% | 18.71%	15.32%
AlexNet	UTK	91.03%	85.90% | 60.63%	80.03% | 59.44%	79.29% | 59.12%	63.80% | 60.13%	57.63% | 48.54%	46.44%
ResNet18	C10	92.38%	84.95% | 50.86%	89.86% | 52.13%	83.16% | 51.54%	56.62% | 49.66%	88.45% | 52.23%	17.87%
ResNet18	C100	80.24%	70.15% | 39.78%	75.02% | 41.22%	69.25% | 40.22%	41.88% | 38.95%	75.01% | 41.07%	13.44%
ResNet18	S10	86.53%	83.86% | 51.36%	83.31% | 51.16%	70.08% | 49.97%	55.22% | 48.33%	80.41% | 50.66%	21.36%
ResNet18	UTK	90.23%	83.54% | 53.19%	81.77% | 52.45%	61.37% | 49.88%	53.88% | 47.96%	75.66% | 51.85%	46.88%
ResNet34	C10	90.89%	84.33% | 59.22%	30.17% | 18.56%	20.85% | 15.33%	13.09% | 20.22%	90.06% | 62.39%	12.79%
ResNet34	C100	81.51%	71.32% | 50.88%	73.98% | 51.02%	75.48% | 51.39%	42.56% | 50.89%	79.67% | 52.77%	16.85%
ResNet34	S10	88.12%	83.69% | 54.98%	82.12% | 53.67%	65.97% | 45.33%	50.02% | 46.89%	79.64% | 52.17%	20.23%
ResNet34	UTK	86.91%	86.36% | 55.71%	76.78% | 50.49%	47.65% | 48.66%	47.94% | 47.25%	80.67% | 51.33%	47.06%
VGG16_BN	C10	91.53%	85.29% | 54.65%	90.35% | 56.17%	81.06% | 53.21%	56.01% | 55.02%	90.77% | 56.55%	14.33%
VGG16_BN	C100	71.84%	61.79% | 43.89%	70.48% | 48.13%	62.57% | 44.62%	46.91% | 46.66%	71.93% | 48.73%	10.65%
VGG16_BN	S10	90.22%	87.69% | 51.44%	87.66% | 51.43%	80.09% | 49.77%	52.89% | 50.21%	88.43% | 52.54%	19.13%
VGG16_BN	UTK	90.83%	85.45% | 53.88%	87.88% | 54.98%	55.69% | 53.91%	58.70% | 55.02%	90.21% | 55.28%	46.24%
VGG19_BN	C10	91.95%	89.34% | 55.66%	84.95% | 53.37%	79.66% | 48.65%	41.07% | 48.66%	82.01% | 48.93%	10.88%
VGG19_BN	C100	70.88%	62.97% | 44.65%	69.72% | 45.97%	61.38% | 45.11%	45.01% | 44.03%	46.72% | 45.67%	11.01%
VGG19_BN	S10	90.11%	88.44% | 55.09%	86.82% | 55.01%	82.79% | 54.33%	34.07% | 54.67%	55.44% | 54.89%	19.67%
VGG19_BN	UTK	90.14%	88.99% | 55.67%	87.02% | 55.34%	87.98% | 55.73%	58.32% | 55.62%	88.19% | 55.54%	43.87%

Table 2: MS attack accuracy for FlexGE with a randomized system configuration.

DarknetTZ needs to take about 98.91% performance overhead, which also suggests it puts 98.91% of the protected model in TEE to achieve $Security_{max}$ for CIFAR10 (C10) and CIFAR100 (C100). However, for STL10 (S10) and UTKFace (UTK), it only needs to put 35% of the model in TEE to achieve $Security_{max}$. Compared to other defenses, FlexGE achieves the lowest performance cost in most cases. That is, FlexGE generally incurs less performance overhead while achieving the highest level of black-box defense. One interesting result we find is that, on average, *P2* (Serdab) has more performance cost to achieve $Security_{max}$ compared with *P1* (DarknetTZ), which illustrates that it is more secure to offload deep layers to GPU than offload shallow layers. The reason is that shallow layers are close to input data and respond more to low-level photographic information of the original inputs. In contrast, deep layers represent more abstract and specific feature information.

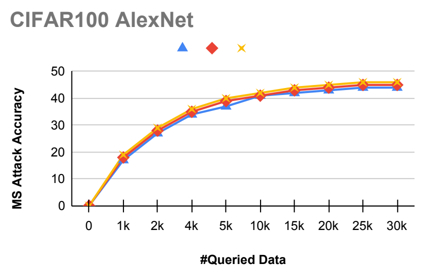
Observation 3: *Shallow layers are close to input data and are more easy to expose model privacy.*

	AlexNet				ResNet18				ResNet34				VGG16_BN			
	c10	c100	s10	UTK	c10	c100	s10	UTK	c10	c100	s10	UTK	c10	c100	s10	UTK
DarkneTZ	98.91%	98.91%	35.28%	35.28%	99.07%	99.07%	100%	35.12%	99.23%	99.23%	100%	35.33%	100%	100%	81.55%	66.45%
Serdab	98.97%	98.97%	99.11%	99.12%	99.91%	99.91%	100%	35.56%	70.50%	100%	100%	50.05%	100%	100%	100%	100%
Magnitude	72.91%	80.58%	100%	68.21%	100%	80.25%	100%	58.48%	19.37%	100%	100%	5.01%	100%	80.34%	100%	100%
TEEslice	64.21%	55.09%	33.22%	31.10%	49.97%	49.97%	51.78%	50.44%	20.87%	41.59%	45.96%	8.27%	55.88%	50.99%	51.56%	52.34%
SOTER	99.98%	99.98%	100%	58.73%	100%	100%	100%	100%	100%	100%	81.23%	100%	100%	100%	100%	100%
FlexGE	49.23%	49.98%	31.17%	31.03%	47.83%	44.22%	41.37%	39.66%	19.56%	35.27%	31.23%	12.44%	48.34%	43.11%	49.69%	40.71%

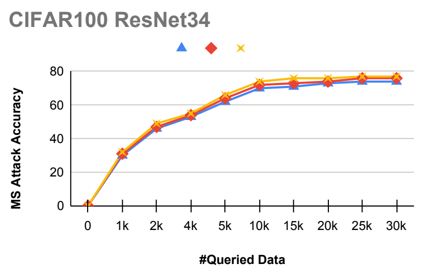
Table 3: Different Performance($P^{\star}$) values of optimal configuration in front of MS. A lower value represents a lower performance overhead. The performance overhead for $Security_{min}$ and $Security_{max}$ baselines are 0% and 100%, respectively.

7.4 Accuracy Loss

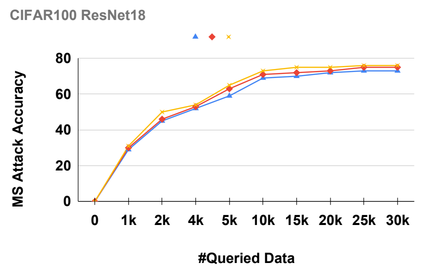
We compare the accuracy between FlexGE and TEESLICE as well as the baseline that puts the whole model into the enclave. Following TEESLICE, we set the queried data sizes as 1K, 2K, 4k 5K, 10K, 15K, 20K, 25K, 30K. We display MS accuracy on CIFAR100 and four models (AlexNet, ResNet18, ResNet34, and VGG16BN) in Figure 6. In general,



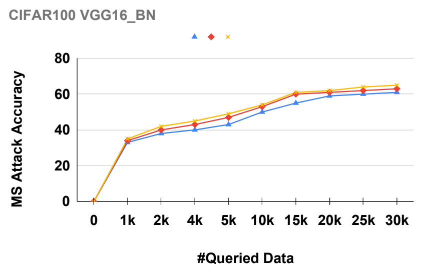
(a) MS attack Accuracy on AlexNet



(b) MS attack Accuracy on ResNet34



(c) MS attack Accuracy on ResNet18



(d) MS attack Accuracy on VGG16_BN

Fig. 6: Comparison of FlexGE, TEESlice, and the whole enclave protection against MS attacks with different sizes of queried data. △ represents TEESLICE, ◇ represents FlexGE, and ✗ represents whole enclave.

FlexGE does not lead to a considerable loss of accuracy corresponding to the baseline. We choose the optimal configuration for FlexGE with the same performance score (81.66%) with TEESLICE based on the evaluation methods in section 7.2 and 7.3. Across most queried data, FlexGE achieves higher accuracy than TEESLICE, which depends on expensive supervised machine learning model training. This reliance can lead to overfitting, potentially resulting in suboptimal outcomes.

7.5 Performance

We run the TEESLICE's source code on the same Desktop PC with FlexGE. We choose the optimal configuration for FlexGE with the same security score with TEESLICE based on the results in section 7.3. We ran all experiments ten times and got the average inference time. The running time deviates is less than 10% from the average. We calculate the throughput (images per second) as it is a common criterion to evaluate the speed of machine learning systems.

Table 4 presents the throughput of FLexGE on three models (AlexNet, ResNet18, and VGG16BN), as well as two baselines: putting the whole model in the enclave and directly running on the GPU. The whole enclave is the throughput lower bound, and the direct GPU is the upper bound. From the results, we can see that the throughput of direct GPU (from 91.54 to 473.72) is much higher than that of the whole-enclave (from 1.52 to 7.63), demonstrating the efficiency of GPU. We choose the same security score (51.38%) for FlexGE and TEESLICE. The throughputs of TEESLICE ranges from 38.09 to 80.10, which is much faster than the whole-enclave baseline but slower than directly running on the GPU. In contrast, FlexGE's throughputs range from 38.77 to 82.01. Most of the results have performance speedups with reference to TEESLICE, which are, on average, 7.53% faster than TEESLICE. The performance speedup mainly comes from FLexGE's EPT I/O protection primitives and optimization mechanism, such as the super page optimization EPT_2 M, compared to the cryptographic I/O protection of TEESLICE.

		AlexNet	ResNet18	VGG16_BN
	Whole Enclave	6.51	7.63	1.52
	Direct GPU	473.72	266.13	91.54
TEESLICE	CIFAR10	40.05	58.43	68.60
	CIFAR100	42.13	41.21	53.74
	STL10	80.10	61.04	66.23
	UTKFace	38.09	53.42	38.56
FlexGE	CIFAR10	44.88	62.78	71.25
	CIFAR100	48.51	40.85	56.62
	STL10	82.01	62.92	70.13
	UTKFace	42.44	52.89	38.77

Table 4: The throughput comparison between shielding-whole-model, no-shield, TEESLICE, and FlexGE .

To analyze the performance of FlexGE further, we also logged the latency of different parts during the inference phase. We break down the inference latency of FlexGE is divided into three parts: data transfer, partition in the enclave, and partition on GPU. Data Transfer refers to the time it takes to transfer data between SGX and GPU. Partition in the enclave refers to the time it takes to compute the layers/filters inside SGX. Partition on GPU is the time to compute the layers on the GPU. Table 5(a) displays the percentage of each part over the total inference latency. From the table, we can see that partition in the enclave occupies 65.09% of the inference time due to the constrained computation resources inside SGX. Data Transfer occupies 32.18% of the inference time. In particular, the data transfer with EPT protection spends 9.44% of the inference time, the data transfer with EPT_2M has 8.32% of the inference time, and encryption protection occupies the most time of data transfer, which is 14.42%. Partition on GPU only occupies 2.73% of the time due to the strong computation ability of the GPU. Note that although FlexGE introduces the additional overhead of Data Transfer, FlexGE still accelerates the overall inference time to a large degree. The reason mainly derives from two aspects. On the one hand, the EPT-based I/O protection introduces a low-performance overhead to the data transfer. On the other hand, the partition in the enclave mainly comes from non-linear layers, and most of the linear layers go to GPU, which largely speeds up the performance.

Observation 4: *EPT based I/O protection introduces much lower performance than encryption based I/O protection.*

Data transfer			Partition(Enclave)	Partition(GPU)
EPT	EPT_2M	Encrypt		
9.44%	8.32%	14.42%	65.09%	2.73%

(a)

	SST-2	**MRPC**	**RTE**	**Average**
Whole-enclave	51.72%	69.67%	49.58%	57.15%
Direct-GPU	93.42%	86.57%	69.99%	82.26%
FlexGE	52.29%	70.34%	50.13%	58.65%

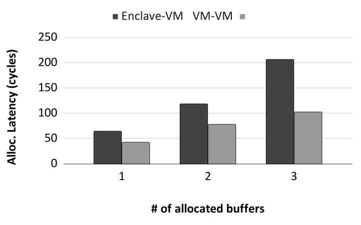
(b)

Table 5: (a)FlexGE inference time breakdown.(b)MS accuracy on BART

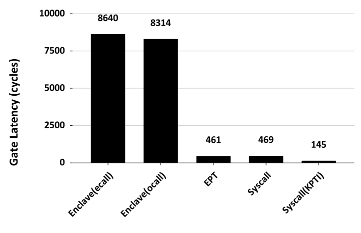
7.6 Overheads: Shared Data Allocations, Gate Latencies

In FlexGE, the weight matrix can be shared via shared memory between Enclave and the EPT backend or between VMs corresponding to the EPT backend and the EPT_2 M backend. To illustrate the benefits of the shared memory, we measure, for each of the mechanisms, the execution time of a DNN model that allocates 1 to 3 shared weight matrices (size 1K Byte) and returns immediately. The results are in Figure 7a. Shared memory allocations between Enclave and EPT are about two times slower than typical VMs shared memory allocations.

Another source of FlexGE overhead is gate latency. To illustrate the raw performance of FlexGE's gates, we measure the gate latency of Enclave ECall gate, Enclave OCall gate, and EPT gate. The results are shown in Figure 7b. The Enclave OCall gate is slightly faster than the Enclave ECall gate and 18x slower than the EPT gate. EPT latencies are similar to syscall latencies without KPTI, illustrating the practicability of the EPT backend.



(a) Allocation latencies



(b) Gate latencies

Fig. 7: FlexGE latency microbenchmarks.

8 Discussion

Scalability to Large Language Models (LLMs). Recently, LLMs (such as ChatGPT [33] and LLaMA [11]) have been largely moved forward and widely used. LLMs bring new challenges to model privacy protection solutions because their sizes usually contain up to hundreds of billions of parameters that are much larger than traditional DNNs (only hundreds of millions of parameters [20]). However, the idea of FlexGE can also be applied to LLMs to protect the sensitive model privacy. To demonstrate the generalization of FlexGE, we evaluate FlexGE on a representative LLMs model, BART [24], and three datasets (SST-2, MRPC, and RTE) from the popular GLUE dataset [49]. We report MS accuracy in Table 5(b).

Other Metrices for Security and Performance Score. Currently we use the model stealing accuracy as the security score metric, however, fidelity [37] and Attack Success Rate (ASR) [37] can be another two metrics for model stealing. Fidelity is the percentage of test samples with identical predictions between the surrogate model and the victim model, including the samples that are misclassified by the victim model. ASR measures the transferability of adversarial samples. Moreover, for Membership Inference Attack (MIA) [52], we can also take gradient-based MIA accuracy, generalization gap [51], and confidence gap [52] as the metrics. For the performance score, we can use FLOPs to measure the utility cost of DNN models [16].

9 Related Work

Compartmentalization. Several compartmentalization frameworks [15], [31], [40], [23] rely on code annotations for application porting. However, FlexGE targets data annotation by annotating the weight matrix offset of the DNN model. A few studies provide various degrees of porting automation through data flow analysis [5,27], and KSplit [17] applies compartmentalization and automation on driver isolation. Their principles can be applied to increase the degree of automation of FlexGE.

GPU TEEs. Recent work explored implementing trusted architectures directly inside GPUs to achieve isolation. Graviton [48] relies on architectural modification to the GPU to support TEE for GPU. Similarly, HIX [18] relies on hardware modification to the CPU, including SGX hardware and PCIe routing. Meanwhile, HETEE [56] supports large-scale confidential computing using PCIe Express-Fabric to distribute computation over server nodes that are physically isolated. StrongBox [9] and GEVisor [50] are software solutions based on existing hardware to build GPU TEE. StrongBox targets for ARM Endpoints based on TrustZone with an integrated GPU, while GEVISOR is designed to support trusted GPU execution with SGX enclaves.

Model Privacy Protection Methods. Existing model privacy protection approaches can be classified into four categories: two-party computation (2PC) based approaches, homomorphic encryption (HE) based approaches, trusted execution environment (TEE) based approaches, and obfuscation approaches. 2PC-based approaches [38], [45], aim to protect the confidentiality of both user data in the client and DNN model on the cloud server. HE-based approaches [13] perform secure inference based on encrypted DNN model and encrypted data. TEE-based approaches as listed in section 3 focus on securing DNN model computations in untrustworthy environments through TEE technologies. Obfuscation approaches can be categorized into approaches that protect the model's input [3] and approaches that safeguard its structure[55,25].

10 Conclusion

In this paper, we have proposed FlexGE, a secure model inference system for DNNs that supports arbitrary partitioning policies between TEE and GPU. It provides the flexibility for delegating partial computations to different backends with different protection primitives and various policies.

Acknowledgments

This work is supported in part by NSF grants CNS-2145744, ONR grant N00014-23-1-2157, and Wistron. Any findings and opinions expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

1. Nvidia h100 tensor core gpu architecture. https://resources.nvidia.com/en-us-tensor-core (2022), `https://resources.nvidia.com/en-us-tensor-core`
2. Abadi, M., Budiu, M., Erlingsson, U., Ligatti, J.: Control-flow integrity principles, implementations, and applications. ACM Transactions on Information and System Security (TISSEC) **13**(1), 1–40 (2009)
3. AprilPyone, M., Kiya, H.: Block-wise image transformation with secret key for adversarially robust defense. IEEE Transactions on Information Forensics and Security **16**, 2709–2723 (2021)
4. Barzasi, G.: The illusion of randomness: demystifying the entropy of aslr on common operating systems (2022)
5. Bauer, M., Rossow, C.: Cali: Compiler-assisted library isolation. In: Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security. pp. 550–564 (2021)
6. Chen, J., Wang, J., Peng, T., Sun, Y., Cheng, P., Ji, S., Ma, X., Li, B., Song, D.: Copy, right? a testing framework for copyright protection of deep learning models. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 824–841. IEEE (2022)
7. Chen, Y., Shen, C., Wang, C., Zhang, Y.: Teacher model fingerprinting attacks against transfer learning. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 3593–3610 (2022)
8. Coates, A., Ng, A., Lee, H.: An analysis of single-layer networks in unsupervised feature learning. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics. pp. 215–223. JMLR Workshop and Conference Proceedings (2011)
9. Deng, Y., Wang, C., Yu, S., Liu, S., Ning, Z., Leach, K., Li, J., Yan, S., He, Z., Cao, J., et al.: Strongbox: A gpu tee on arm endpoints. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 769–783 (2022)
10. Elgamal, T., Nahrstedt, K.: 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). pp. 519–528. IEEE (2020)
11. Facebook AI: Introducing llama: A foundational, 65-billion-parameter large language model. `https://ai.facebook.com/blog/large-language-model-llama-meta-ai/` (2023)
12. Filippini, F., Lattuada, M., Jahani, A., Ciavotta, M., Ardagna, D., Amaldi, E.: Hierarchical scheduling in on-demand gpu-as-a-service systems. In: 2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC). pp. 125–132. IEEE (2020)
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: International conference on machine learning. pp. 201–210. PMLR (2016)
14. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
15. Hedayati, M., Gravani, S., Johnson, E., Criswell, J., Scott, M.L., Shen, K., Marty, M.: Hodor:{Intra-Process} isolation for {High-Throughput} data plane libraries. In: 2019 USENIX Annual Technical Conference (USENIX ATC 19). pp. 489–504 (2019)
16. Hou, J., Liu, H., Liu, Y., Wang, Y., Wan, P.J., Li, X.Y.: Model protection: Real-time privacy-preserving inference service for model privacy at the edge. IEEE Transactions on Dependable and Secure Computing **19**(6), 4270–4284 (2021)
17. Huang, Y., Narayanan, V., Detweiler, D., Huang, K., Tan, G., Jaeger, T., Burtsev, A.: {KSplit}: Automating device driver isolation. In: 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22). pp. 613–631 (2022)
18. Jang, I., Tang, A., Kim, T., Sethumadhavan, S., Huh, J.: Heterogeneous isolated execution for commodity gpus. In: 24th International Conference on Architectural Support for Programming

Languages and Operating Systems (ASPLOS 2019). pp. 455–468. ACM, Providence, RI (2019), http://doi.acm.org/10.1145/3297858.3304021

19. Kato, S., McThrow, M., Maltzahn, C., Brandt, S.: Gdev: First-class GPU resource management in the operating system. In: Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12). pp. 401–412. USENIX, Boston, MA (2012), https://www.usenix.org/conference/atc12/technical-sessions/presentation/kato
20. Keras Contributors: Keras applications. https://keras.io/api/applications/ (2017)
21. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
22. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems **25** (2012)
23. Lefeuvre, H., Bădoiu, V.A., Jung, A., Teodorescu, S.L., Rauch, S., Huici, F., Raiciu, C., Olivier, P.: Flexos: towards flexible os isolation. In: Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 467–482 (2022)
24. Lewis, M.: Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. arXiv preprint arXiv:1910.13461 (2019)
25. Li, J., He, Z., Rakin, A.S., Fan, D., Chakrabarti, C.: Neurobfuscator: A full-stack obfuscation tool to mitigate neural architecture stealing. In: 2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). pp. 248–258. IEEE (2021)
26. Lins, S., Pandl, K.D., Teigeler, H., Thiebes, S., Bayer, C., Sunyaev, A.: Artificial intelligence as a service: classification and research directions. Business & Information Systems Engineering **63**, 441–456 (2021)
27. Liu, S., Tan, G., Jaeger, T.: Ptrsplit: Supporting general pointers in automatic program partitioning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 2359–2371 (2017)
28. Loukas, G.: Cyber-physical attacks: A growing invisible threat. Butterworth-Heinemann (2015)
29. Mo, F., Shamsabadi, A.S., Katevas, K., Demetriou, S., Leontiadis, I., Cavallaro, A., Haddadi, H.: Darknetz: towards model privacy at the edge using trusted execution environments. In: Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services. pp. 161–174 (2020)
30. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th international conference on machine learning (ICML-10). pp. 807–814 (2010)
31. Narayan, S., Disselkoen, C., Garfinkel, T., Froyd, N., Rahm, E., Lerner, S., Shacham, H., Stefan, D.: Retrofitting fine grain isolation in the firefox renderer. In: 29th USENIX Security Symposium (USENIX Security 20). pp. 699–716 (2020)
32. Neiger, G., Santoni, A., Leung, F., Rodgers, D., Uhlig, R.: Intel virtualization technology: Hardware support for efficient processor virtualization. Intel Technology Journal **10**(3) (2006)
33. OpenAI: Chatgpt. https://chat.openai.com (2023)
34. Orekondy, T., Schiele, B., Fritz, M.: Knockoff nets: Stealing functionality of black-box models. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 4954–4963 (2019)
35. O'Shea, K., Nash, R.: An introduction to convolutional neural networks. arXiv preprint arXiv:1511.08458 (2015)
36. Rafkind, J.: Vembyr - multi-language peg parser generator written in python. http://code.google.com/p/vembyr/ (November 2011)
37. Rakin, A.S., Chowdhuryy, M.H.I., Yao, F., Fan, D.: Deepsteal: Advanced model extractions leveraging efficient weight stealing in memories. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 1157–1174. IEEE (2022)

38. Rathee, D., Rathee, M., Kumar, N., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: Cryptflow2: Practical 2-party secure inference. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 325–342 (2020)
39. Redmon, J.: Darknet: Open Source Neural Networks in C. `http://pjreddie.com/darknet/` (2013–2016)
40. Schrammel, D., Weiser, S., Steinegger, S., Schwarzl, M., Schwarz, M., Mangard, S., Gruss, D.: Donky: Domain keys–efficient {In-Process} isolation for {RISC-V} and x86. In: 29th USENIX Security Symposium (USENIX Security 20). pp. 1677–1694 (2020)
41. Seo, J., Lee, B., Kim, S.M., Shih, M.W., Shin, I., Han, D., Kim, T.: Sgx-shield: Enabling address space layout randomization for sgx programs. In: NDSS (2017)
42. Shacham, H., Page, M., Pfaff, B., Goh, E.J., Modadugu, N., Boneh, D.: On the effectiveness of address-space randomization. In: Proceedings of the 11th ACM conference on Computer and communications security. pp. 298–307 (2004)
43. Shen, T., Qi, J., Jiang, J., Wang, X., Wen, S., Chen, X., Zhao, S., Wang, S., Chen, L., Luo, X., et al.: {SOTER}: Guarding black-box inference for general neural networks at the edge. In: 2022 USENIX Annual Technical Conference (USENIX ATC 22). pp. 723–738 (2022)
44. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
45. Srinivasan, W.Z., Akshayaram, P., Ada, P.R.: Delphi: A cryptographic inference service for neural networks. In: Proc. 29th USENIX Secur. Symp. pp. 2505–2522 (2019)
46. Sun, Z., Sun, R., Liu, C., Chowdhury, A.R., Lu, L., Jha, S.: Shadownet: A secure and efficient on-device model inference system for convolutional neural networks. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 1596–1612. IEEE (2023)
47. Tramer, F., Boneh, D.: Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. arXiv preprint arXiv:1806.03287 (2018)
48. Volos, S., Vaswani, K., Bruno, R.: Graviton: Trusted execution environments on gpus. In: 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2018). pp. 681–696. USENIX Association, Carlsbad, CA (2018), `https://www.usenix.org/conference/osdi18/presentation/volos`
49. Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., Bowman, S.R.: Glue: a multi-task benchmark and analysis platform for natural language understanding. corr abs/1804.07461 (2018). arXiv preprint arXiv:1804.07461 (2018)
50. Wu, X., Tian, D.J., Kim, C.H.: Building gpu tees using cpu secure enclaves with gevisor. In: Proceedings of the 2023 ACM Symposium on Cloud Computing. pp. 249–264 (2023)
51. Yeom, S., Giacomelli, I., Fredrikson, M., Jha, S.: Privacy risk in machine learning: Analyzing the connection to overfitting. In: 2018 IEEE 31st computer security foundations symposium (CSF). pp. 268–282. IEEE (2018)
52. Yuan, X., Zhang, L.: Membership inference attacks and defenses in neural network pruning. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 4561–4578 (2022)
53. Zhang, Z., Song, Y., Qi, H.: Age progression/regression by conditional adversarial autoencoder. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 5810–5818 (2017)
54. Zhang, Z., Gong, C., Cai, Y., Yuan, Y., Liu, B., Li, D., Guo, Y., Chen, X.: No privacy left outside: On the (in-) security of tee-shielded dnn partition for on-device ml. arXiv preprint arXiv:2310.07152 (2023)
55. Zhou, T., Ren, S., Xu, X.: Obfunas: A neural architecture search-based dnn obfuscation approach. In: Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design. pp. 1–9 (2022)
56. Zhu, J., Hou, R., Wang, X., Wang, W., Cao, J., Zhao, B., Wang, Z., Zhang, Y., Ying, J., Zhang, L., et al.: Enabling rack-scale confidential computing using heterogeneous trusted execution environment. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 1450–1465. IEEE (2020)