Occlumency: Privacy-preserving Remote Deep-learning Inference Using SGX

Taegyeong Lee[1*], Zhiqi Lin[2*], Saumay Pushp[1], Caihua Li[3*], Yunxin Liu[4], Youngki Lee[5], Fengyuan Xu[6*], Chenren Xu[7*], Lintao Zhang[4], Junehwa Song[1]

[1]KAIST, [2]University of Science and Technology of China, [3]Rice University, [4]Microsoft Research, [5]Seoul National University, [6]National Key Lab for Novel Software Technology – Nanjing University, [7]Peking University
[1]{tglee, saumay, junesong}@nclab.kaist.ac.kr, [2]ralzq01@mail.ustc.edu.cn, [3]caihua.li@rice.edu, [4]{yunxin.liu, lintaoz}@microsoft.com, [5]youngkilee@snu.ac.kr, [6]fengyuan.xu@nju.edu.cn, [7]chenren@pku.edu.cn

ABSTRACT

Deep-learning (DL) is receiving huge attention as enabling techniques for emerging mobile and IoT applications. It is a common practice to conduct DNN model-based inference using cloud services due to their high computation and memory cost. However, such a cloud-offloaded inference raises serious privacy concerns. Malicious external attackers or untrustworthy internal administrators of clouds may leak highly sensitive and private data such as image, voice and textual data. In this paper, we propose Occlumency, a novel cloud-driven solution designed to protect user privacy without compromising the benefit of using powerful cloud resources. Occlumency leverages secure SGX enclave to preserve the confidentiality and the integrity of user data throughout the entire DL inference process. DL inference in SGX enclave, however, impose a severe performance degradation due to limited physical memory space and inefficient page swapping. We designed a suite of novel techniques to accelerate DL inference inside the enclave with a limited memory size and implemented Occlumency based on Caffe. Our experiment with various DNN models shows that Occlumency improves inference speed by 3.6x compared to the baseline DL inference in SGX and achieves a secure DL inference within 72% of latency overhead compared to inference in the native environment.

CCS CONCEPTS

• **Security and privacy** → *Information flow control*; • **Human-centered computing** → *Ubiquitous and mobile computing systems and tools.*

KEYWORDS

Mobile deep learning; privacy; trusted execution environment; cloud offloading

ACM Reference Format:
Taegyeong Lee, Zhiqi Lin, Saumay Pushp, Caihua Li, Yunxin Liu, Youngki Lee, Fengyuan Xu, Chenren Xu, Lintao Zhang, Junehwa Song. 2019. Occlumency: Privacy-preserving Remote Deep-learning Inference Using SGX. In *The 25th Annual International Conference on Mobile Computing and Networking (MobiCom '19), October 21–25, 2019, Los Cabos, Mexico.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3300061.3345447

1 INTRODUCTION

Deep-learning (DL) techniques are now widely used as core enablers of many life-immersive mobile and Internet of Things (IoT) applications [79]. Advances in Deep Neural Networks (DNNs) has made breakthroughs in many tasks such as object recognition [36], speech recognition [78] and natural language processing [20]. As the execution of accurate DNN models is computation and memory intensive, it is a common practice to conduct DNN-based inference using cloud services [18].

Such cloud offloading offers application (App) developers the opportunity of serving high demands of accuracy-sensitive inference but raises serious privacy concerns which incline users to avoid those services. To use a cloud-based service, a mobile or IoT App sends user's data, such as images, voices, and textual contents, to a multi-tenant cloud for inference. Many of these input data are considered as private and sensitive information, while the cloud platform is hardly

*This work was done in part when the authors were with Microsoft Research as an intern or a visiting researcher.

MobiCom '19, October 21–25, 2019, Los Cabos, Mexico

ACM ISBN 978-1-4503-6169-9/19/10…$15.00
https://doi.org/10.1145/3300061.3345447

trusted and suffers data breaches due to malicious external attacks and untrusted internal administrators[15, 60]. Thus, protecting the user data leakage is critical to App developers in attracting more users while leveraging the cloud-based inference.

It is a critical challenge to enable cloud-based inference while satisfying the balance between users' privacy, inference accuracy, and latency. One approach is to conduct DNN inference over encrypted data using the cryptography methods like fully or semi homomorphic encryption [28, 37]. It achieves a high level of user privacy, but the current performance of such methods is not acceptable in serving mobile users in practice for the deep learning inference. Another approach is *on-device DL inference* [43, 51, 80], which aims to run DL inference locally on off-the-shelf mobile/IoT devices by compressing models or designing new lightweight models. It mitigates the chances of leaking private data; however, compressed and lightweight models hardly achieve the same model accuracy as the large models, due to the trade-off between model size and model accuracy (§2.1). Moreover, on-device inference incurs high energy cost and significantly impacts the lifetime of battery-operated smart devices.

In this paper, we propose Occlumency, a novel cloud-driven solution that strikes a balance between privacy, inference accuracy, and latency without imposing any burden to mobile/IoT devices. Our key idea is to leverage *Trusted Execution Environments* (e.g., Intel Software Guard eXtension (SGX) [9] in our prototype) to perform DL inference in the hardware-protected enclave. A device sends user data directly into an SGX enclave on a cloud server through a secure channel. Occlumency then executes the entire DNN processing pipeline in the enclave and transmits the result back to the device. Consequently, Occlumency protects users' private input data, inference results, and all intermediate outputs throughout the end-to-end offloading process while taking advantages of powerful cloud resources. With Occlumency, devices consume much less resource and energy than the on-device DL inference, minimizing the degradation of user experiences or quick battery drain.

DL inference in SGX enclave, however, impose severe performance challenges, which led prior works to forgo supporting DNN models [41, 56] or a full protection of data privacy [31]. Our experiments show that DL inference inside an SGX enclave is 6.4x slower than the native speed running outside of enclave. We find out two main problems causing such performance degradation. Firstly, memory read and write inside the enclave is slower than the standard execution outside the enclave, which is critical for DL inference requiring a large volume of memory operations. The SGX enclave maintains a protected memory area (*Processor Reserved Memory (PRM)*) where all the data is encrypted through a dedicated chipset, which adds extra steps to encrypt and decrypt data upon every access to memory. Secondly, the protected memory area is small (e.g., 128 MB for Intel Skylake CPU [9]), far less than the size of many accurate DNN models. For example, VGG-16 [67], a widely-used Convolutional Neural Network (CNN) model, requires ≈1 GB of memory, and even AlexNet [50], a shallow CNN model, requires ≈260 MB of memory. SGX on Linux supports the memory usage beyond the PRM size through paging (Windows does not support this and cannot run models larger than the PRM size). However, execution of large DNN models incurs significant overhead to frequently swap in/out pages from protected enclave memory from/to the regular unprotected memory, which results in additional encryption/decryption of data.

Occlumency addresses the performance challenge based on a key observation: many developers use pre-trained public DNN models rather than training their own private models. This is particularly true for large models because it is highly expensive to train a large model, and most developers do not have the expertise to train a fine model. As a result, the confidentiality of those public models is not necessary to be protected. Taking advantages of this observation, Occlumency employs a suite of novel techniques to accelerate DL inference inside the enclave. First, we devise *on-demand weights loading*. We place the DNN model (i.e., weights) in regular main memory and dynamically load a part of model weights needed by the DL inference engine to the protected enclave memory. This technique prevents high latency due to frequent page swapping that could happen by loading the entire DNN model in the enclave. Occlumency uses a hash to check the integrity of the model weights to safeguard model weights against being modified in the unprotected memory and ensure the correctness of model inference. Second, Occlumency employs *memory-efficient inference* to reduce memory usage during DL inference. In particular, it significantly reduces memory usage to store the intermediate data (i.e., feature maps) for inference. Also, we devise a *partitioned convolution* technique to conduct convolution layer computation with a small memory footprint; it consumes far less memory than conventional convolution operations that require a large memory space to reshape input data to convert the convolutions into a single matrix multiplication (called *convolution lowering*). Furthermore, Occlumency builds a parallel pipeline of weights copying, hash checking and model inference to best utilize the hardware resources to optimize the end-to-end system performance.

We have implemented Occlumency on both Linux and Windows based on Caffe [48], a widely-used DL framework. We also evaluated its performance with popular DNN models including AlexNet [50], GoogLeNet [68], ResNet-50/101/152 [36], VGG-16/19 [67], and YOLO [63]. Experimental results show that Occlumency significantly improves the speed of DL inference, and is 3.0 - 4.3x (1.9x for GoogLeNet) faster

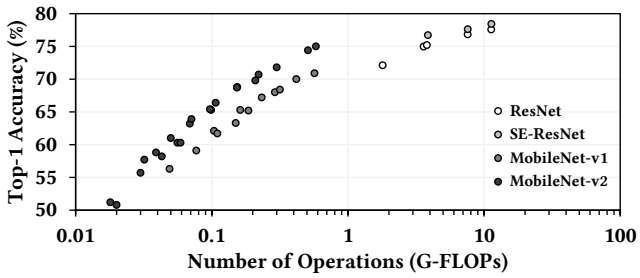

Figure 1: Top-1 accuracy vs. Number of Operations.

than the in-enclave DL inference without our optimization techniques. Compared to the native DL inference running outside of enclave, Occlumency is only 72% slower. Compared to on-device DL inference with Pixel XL, Occlumency reduces the end-to-end model inference latency by 2.1x and saves 53 - 91% of the energy cost at the device side.

The main contributions of this paper are:

- We propose and design the Occlumency system for privacy-preserving remote DL inference using SGX. We develop novel techniques to maximize the DL inference performance running in SGX enclave, including on-demand weights loading, memory-efficient inference and parallel processing pipelines.
- We implement Occlumency on both Linux and Windows including porting the widely-used Caffe framework into SGX. We report implementation details and share our experience learned in porting Caffe into SGX.
- We conduct comprehensive experiments to evaluate the effectiveness of Occlumency with various DNN models. Evaluation results show that Occlumency significantly improves the performance of in-enclave DL inference and outperforms on-device DL inference.

2 BACKGROUND AND MOTIVATION

2.1 Demand of Model-inference Offloading

It is known that there is a trade-off between model accuracy and model size. For the same family of models, a larger model with more layers/parameters has a larger model capacity and thus a higher accuracy. At the same time, a larger model requires more computation and leads to a longer inference latency. For example, Figure 1 shows the trade-off between the accuracy and the number of operations in FLOPs of the state-of-the-art models of ResNet and MobileNet families tested with ImageNet (reported in [36, 39, 40, 65]).

The above trade-off shows that it is highly desirable to offload model inference to a cloud, especially for mobile apps that require a high inference accuracy. Many apps have to use a large model (note that even the large ResNet-152 has a top-1 accuracy < 80%), and thus offloading is essential due to its high computation cost. Offloading also helps reduce the latency and energy cost of model inference using the powerful computation resources of a cloud. Also, cloud-based solutions can provide other advantages: saving the device resources such as the battery, dealing with device heterogeneity, and supporting low-performance hardware.

In this paper, we seek a practical solution to preserve user privacy in remote model inference, targeting at the mobile apps that employ offloading to run highly-accurate large models at low latency or energy cost. There is parallel research on running small models on mobile devices by model compression or designing new models[1] [34, 35, 39, 44, 65], which is complementary to our work.

2.2 Intel SGX

Intel SGX [9] is a set of extensions to the Intel architecture for secure remote execution. It sets aside a secure region of memory address space called an *enclave*. Code execution in the enclave is strongly protected from attacks. SGX ensures the confidentiality and integrity of code and data within the enclave even if the operating system, the hypervisor and the BIOS, are malicious. It also prevents hardware attacks such as memory probes. Even malicious administrators of cloud services cannot access the code and data in the enclave. SGX supports *remote attestation* for a third-party owner to verify the code and data loaded into the enclave.

The SGX enclave provides a hardware-assisted Trusted Execution Environment (TEE) with a minimal attack surface (i.e., the processor boundary). The memory region of the enclave is called *Processor Reserved Memory* (PRM). The CPU protects the PRM from all non-enclave memory accesses including DMA accesses from peripherals. The size of PRM is limited to 128 MB, among which the hardware carves out ≈ 90 MB as the *Enclave Page Cache* (EPC) and the rest is reserved for the meta-data required by the Memory Encryption Engine (MEE) [32]. On Linux, SGX supports *paging* that evicts rarely used EPC pages to unprotected main memory outside the PRM range. To ensure the confidentiality and integrity of the evicted EPC pages, SGX uses extra symmetric key cryptography. With paging, a program in the enclave may use more than 90 MB memory, at the cost of encrypting and decrypting the evicted EPC pages.

Other hardware-assisted TEEs. We have also considered other hardware-assisted TEEs. ARM TrustZone [7] and AMD Secure Memory Encryption (SME) [27] can support large memory size, but they require a trusted OS and thus have a large Trusted Computing Base (TCB). They also cannot protect against attackers with physical DRAM access. There are also research efforts towards providing new TEE designs,

[1]New models may achieve higher accuracy with a smaller size than older ones, but they do not break the accuracy-size trade-off. All the records on model accuracy are from very large models.

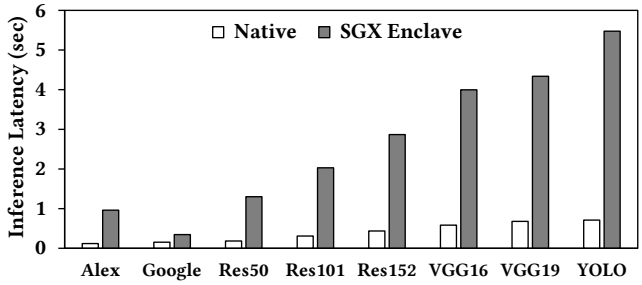

Figure 2: Inference latency of various deep-learning models in native environment and SGX enclave.

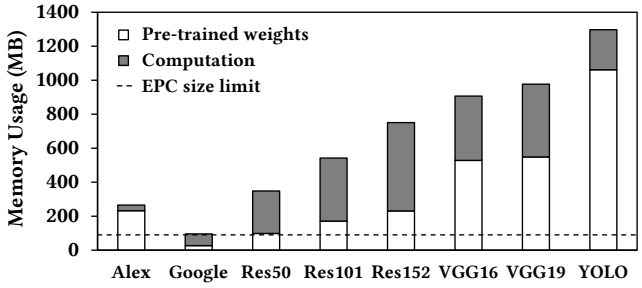

Figure 3: Memory usage of various DNN models compared to the physical memory size limit of EPC.

such as Sanctum [26] and Keystone [13] targeting at RISC-V platforms. They are not commercially available. Indeed, Intel SGX is the most widely deployed TEE in today's data centers. Thus, we choose to use Intel SGX in our implementation, for its strong protection, small TCB, and wide availability.

2.3 Deep-learning Inference in Enclave

To understand how DL inference may work on SGX, we conducted an experiment on a Linux server. We first followed the SGX SDK [10] and ported the Caffe framework into an SGX enclave (§7). We enabled SGX paging so that the enclave may use more memory. Then, we ran DNN models (AlexNet, GoogLeNet, ResNet-50/101/152, VGG-16/19, and YOLO) inside the enclave and measured the latency. We compared to the native case of running the models outside the enclave using Caffe on the same server. Figure 2 shows the results.

Poor model-inference performance in the enclave. From Figure 2, we can see that running DL inference in the enclave is very slow, 6.4x slower than running in the native environment outside the enclave. Two reasons cause this poor performance: 1) the PRM memory is slower than unprotected memory due to the protection overhead of SGX, and 2) there was frequent EPC paging. As aforementioned, the physical memory size of EPC is constrained to only 90 MB, while the model inference is highly memory intensive. It requires a large amount of memory for loading model weights and performing inference computation, particularly for large models. As shown in Figure 3, AlexNet, a primitive CNN model, requires 260 MB memory, and VGG-19, a widely used model, requires ≈1 GB memory space to load model weights, store intermediate feature maps, and compute convolutions. Even GoogLeNet, a lightweight model designed to run on smartphone-like devices, requires a memory size slightly larger than 90 MB. As EPC paging involves heavy computation overhead due to encryption [21, 76], it causes a significant slowdown on the model inference running inside the enclave. Furthermore, on Windows, as paging is not supported, it is impossible to run those models in the enclave.

The above experimental results motivate us to conduct the work of this paper to design Occlumency and develop novel techniques to optimize the performance of DL inference in the enclave.

One may argue whether the physical memory size limit of 128MB is fundamental or not, and such a limit might be significantly increased (e.g., to 4 GB) in the next generation SGX. While we agree that such a limit might be increased (e.g., to 256 MB), but we believe that the increase is unlikely to be significant, compared to the total memory size of a server. SGX is designed to support many enclaves on a single server, and each enclave is supposed to run only the trusted functions (usually small ones) of a program. As all enclaves have the same memory size limit (configured in BIOS and cannot be dynamically adjusted) and the preserved physical memory region of each enclave is exclusively reserved and thus cannot be shared by other enclaves or applications outside enclaves. For this reason, it will be highly memory inefficient and impractical to preserve a large physical memory region for each enclave. Moreover, it is likely that the DNN models will be more complicated and get larger. We believe that Occlumency is not a temporary solution only for the current SGX hardware. Our proposed techniques will bring constant values to future SGX devices as well.

3 PRIVACY AND THREAT MODEL

The security objective of Occlumency is to protect the privacy of mobile user inputs and outputs of deep learning inference services hosted on untrusted clouds. This user data protection is the top priority in terms of service security and also enforced by laws like GDPR [1]. The deep learning models used for inference are assumed to be open source, which is strongly advocated and practiced by both deep learning research community and industry [3, 4, 46], especially for large models pursuing accuracy and robustness.

There are three parties involved in our threat model, the mobile user, the App developer, and the cloud platform. The mobile user trusts the App developer in providing the privacy-preserving inference service running on an untrusted cloud platform rented by the App developer. The App developer trusts the corresponding Intel SGX enclave she deploys

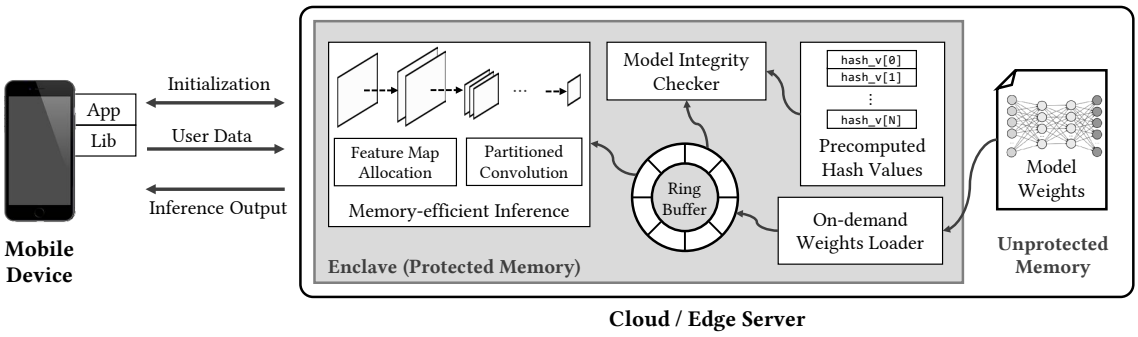

Figure 4: Occlumency system architecture. The shaded part indicates the enclave.

on a cloud as well as software executed inside it (after attestation supported by Intel [25]). The cloud platform is the adversary in our threat model, and its goal is to steal user inputs or thwart correct computations like damaging the integrity of inference models.

The cloud platform is capable of accessing on-cloud software and hardware resources except for the App developer's SGX enclave. We assume the cloud uses off-the-shelf hardware, and GPUs are not used by the App developer, which is orthogonal to work [72]. The SGX enclave is secure in terms of both computations and memory footprints. The communication channel between the user and enclave is protected by the App developer using the TLS or similar secure protocols. We do not consider the SGX side-channel attacks [33, 81] and rollback attacks, which can be prevented [55, 57], as well as Denial of Services (DoS). The memory usage pattern of model inference is dependent on the model architecture, not user data. Thus, it is hard to infer user data from the memory usage pattern. Protection of private models is also out of the scope of this paper, although our system can be used for any customized models without protecting their confidentiality.

4 OCCLUMENCY DESIGN

4.1 Design Goals

We designed Occlumency to achieve following goals:

Privacy protection throughout entire DL inference. Our primary design goal is to protect the end-to-end offloading process for DL inference. In particular, we aim to protect sensitive *raw input data* of DL inference collected from the users' devices and *inference results* after DL inference. We also seek to safeguard *intermediate feature maps* generated during DL inference, which can be used by malicious attackers to reconstruct input data or infer sensitive information.

Model integrity. The second goal is to prevent DNN models from being manipulated by attackers, which may create wrong inference results. Occlumency should continuously monitor whether the pre-trained model weights are modified at runtime and make sure to apply the correct original weights throughout DL inference.

Minimal inference latency. Occlumency should perform inference at a low latency while protecting privacy. The low latency is essential to support interactive applications such as speech recognition and user input prediction and to increase the throughput of the cloud service. Naive DL inference in SGX enclave slows down by 6.4x compared to the native environment and thus it is essential to optimize the latency.

No accuracy loss. Occlumency aims to run unmodified large DNN models (without compression, distillation, or approximation) to achieve high accuracy provided by those models. This goal is important since cloud-added DL inference is likely to be used for applications that require high accuracy based on large models.

4.2 System Overview

Figure 4 shows the architecture of Occlumency. On the device side, Occlumency provides a library for a DL-enabled app to interact with the remote server. The library is designed to provide simple function calls for the app to initialize the remote DL inference, send user data to the remote server, and get the model-inference output from the server. The initialization details are hidden from the app to make it easy for developers to build apps, including setting up a secure communication channel between the device and the server, loading the model data (i.e., weights), creating the enclave, and initializing the code and data structures in the enclave. Then, the app may call the remote inference multiple times. The user data and model output are transmitted using the secure channel between the device and the enclave. The server could be a remote cloud server or a nearby edge server.

The model weights are first loaded into the unprotected memory outside the enclave. The *on-demand weights loader* dynamically loads necessary weights into the enclave as required by the inference engine. The *model integrity checker* ensures the integrity of the loaded model weights using the pre-computed hash values. We designed the *ring buffer* (constituting of multiple reusable sub-buffers connected circularly) to coordinate the weights loading, hash checking and model inference for a parallel processing pipeline. The

memory-efficient inference engine executes model inference in a memory-efficient way by managing the memory allocation of feature maps and using partitioned convolutions.

With this system design, Occlumency achieves all its design goals. By running the whole DL inference engine inside the enclave, Occlumency effectively protects the confidentiality and integrity of the user data, the final model output and all the intermediate feature maps. By storing all the weights of the whole model in the unprotected memory, Occlumency can run large state-of-the-art models without any accuracy loss and model integrity is ensured by hash checking. With on-demand weights loading and memory-efficient inference, Occlumency avoids page swapping and thus significantly reduces the model-inference latency.

5 ON-DEMAND WEIGHTS LOADING AND INTEGRITY CHECKING

A key design decision in Occlumency is storing model weights in unprotected memory outside the enclave and loading the needed weights into the enclave on demand. We made this decision since the enclave has a limited physical memory size smaller than many widely-used DNN models, which causes significant performance degradation as shown in §2.3. This on-demand weights loading can do a better job than page swapping in two folds:1) it directly reads the model weights from unprotected memory without the extra cryptography overhead of page swapping; and 2) it knows more about the internals of the model structure and the inference procedure and thus may load the weights just before they are needed by the inference engine, while page swapping knows nothing about model inference and thus may wrongly swap out the weights required by the inference engine.

Putting the whole model in unprotected memory calls for a mechanism to ensure model integrity, as malicious attackers may modify the model weights. For example, an attacker may alter a speech-recognition model to issue an intentional voice command to trigger undesirable actions on the user's device. Thus, it is critical to ensure the correctness of the model inference. To do it, we propose to use hashing to protect model integrity.

Note that Occlumency does not protect model confidentiality as attackers can read the model data in unprotected memory. Protecting model confidentiality is out of the scope of this paper. However, if protecting model confidentiality is necessary, Occlumency may fall back to allow loading the whole model into the enclave at the cost of page swapping.

5.1 On-demand Weights Loading

After a model is loaded and stored in the unprotected memory, the base address of the model weights is passed into the enclave. Upon an incoming model inference request, e.g., when the user sends an image for object recognition, the on-demand weights loader dynamically loads (i.e., copies) the needed weights into the enclave for model inference.

For efficient weights loading, the on-demand weights loader takes advantages of the internals of model inference such as layer types and data structures. As model inference is executed layer by layer, the on-demand weights loader loads the weights layer by layer, from the first layer of the model to the last one. Ideally, the weights of each layer are already ready in the ring buffer just before the inference engine starts to execute that layer so that the inference engine does not need to wait for the weights loading. After the execution of a layer finished, the corresponding buffer is released so that it can be reused to load the next layer.

Handling large layers. Most layers can fit into a single buffer in the ring buffer. For example, convolutional layers are usually computation intensive but small in size. However, some types of layers such as fully-connected layers may be huge and a single buffer cannot load them. To handle large layers, Occlumency divides them into blocks and loads each block one by one. The majority of data in DNN models is matrices and vectors. In dividing large layers into blocks, Occlumency keeps the boundary of the matrix rows and columns. Doing so makes it possible to compute the multiplication of two large matrixes by computing the multiplications of a set of small matrix blocks. Consequently, the inference engine may partially execute a large layer using its already loaded blocks. Thus, that are even larger than the total size of the ring buffer can be supported.

5.2 Model Integrity Checking

Model integrity checking happens after the ring buffer in the enclave loads the weights. Occlumency employs a hash-based method to ensure the model integrity, which consists of an offline hash building phase and an online hash checking phase. The offline hash building phase is performed in a trusted environment. Occlumency follows the same data boundary used in the on-demand weighs loader to compute hash values. For each small layer, a hash value is computed from the weights of the layer. A large layer is divided into small blocks of the same block size used in the weights loading, and a hash value is computed for each block. All the *pre-computed hash values* remain loaded in the enclave, and are used in online hash checking as shown in Figure 4. They never leave the enclave, and thus, are always protected.

In the online hashing checking phase, after each layer or layer block is loaded into the ring buffer, the model integrity checker computes a new hash value from loaded weights. Then the new hash value is compared to the corresponding *pre-computed hash value* of the same weights. If the two hash values are the same, the model integrity checker marks

the loaded weights as ready for inference and the inference engine can safely use the weights for model inference. Otherwise, the model integrity checker raises an error, stops the inference engine and returns an error message to the device. It is up to the app to handle the failure of inference.

Parallel processing pipeline. Occlumency uses the ring buffer to coordinate the executions of weights loading, hash checking and model inference, and builds a parallel processing pipeline for efficient model inference. Three threads are created for weights loading, hash checking and model inference, respectively, and executed in parallel. Once the ring buffer is ready with a set of weights, the model integrity checker immediately starts to compute and check the hash of the weights. At the same time, the on-demand weights loader starts to load the next set of weights. Similarly, immediately after the successful hash checking, the model inference engine starts to execute the loaded weights, and the model integrity checker goes to check the newly loaded weights. After the inference engine used the weights to compute the layer, it releases the buffer of the weights (if not shared) for loading new weights.

Copying all the model weights into the enclave and checking their hash values introduce considerable overhead, as shown in §8.2. For some applications, it may not require preserving model integrity. In that case, we can apply a different model loading policy. For example, one may want to directly read model data from unprotected memory without any data copying and hash checking, or only check the model integrity once in creating the enclave. In §8.2, we measure the performance of different weights loading policies for the trade-off between the model-inference latency and the model-integrity protection.

6 MEMORY-EFFICIENT INFERENCE

Another critical challenge for SGX-based DL inference is the lack of memory required for inference. Inference should run within ≈80 MB of memory given that the available heap size of EPC is ≈ 85 MB and a part of EPC (5 MB) is already taken to load the model network. Moreover, at least 20 MB of memory is required to load the model weights to get a benefit of parallelism as described in §5.2. However, CNN execution often requires a higher memory space than the available memory (e.g., ≈116 MB for `conv1_2` layer of VGG-16). Unless handled properly, frequent paging occurs during the inference, which slows down inference speed significantly.

Memory usage break-down. To understand the memory requirements of inference, we break down the memory usage of VGG-16, a widely used CNN model. Table 1 shows the result. We identified that the dominant memory usages are two folds: storing feature maps and buffers for convolution computation. In total, feature maps require 60 MB of memory, and convolution computation requires 327 MB. Moreover, some convolutional layers require memory larger than the upper bound of EPC heap size for a single layer. To address the problems, we devised two techniques: *memory-efficient feature map allocation* and *partitioned convolution*.

Table 1: Memory usage of VGG-16 per layer

Type	Memory Usage (KB)		
	Max.	Avg.	Total
Intermediate feature maps	25,690	6,963	60,987
Convolution lowering	115,605	25,150	326,947
Others (pooling/dropout mask, etc.)	1,192	1,163	6,138

6.1 Memory-efficient Feature Map Allocation

Unlike pre-trained weights, Occlumency manages the feature maps (i.e., the outcome of a Neural Network (NN) layer) inside the enclave. Thus, it is essential to reduce the memory size required to store feature maps. A naive approach to keep the entire feature maps throughout the inference would consume high memory space (e.g., ≈60 MB memory for VGG-16), which fits within EPC but leaves too little space for convolution computation.

Occlumency stores feature maps only for the minimum required duration and removes it right away. For instance, a convolutional layer creates a feature map as its output and only the next layer utilizes the feature map. Since the feature map size and its validity are known for a given model, Occlumency can plan how much memory space to allocate for storing feature maps for the entire model during the model initialization phase.

The approach is based on the observation that DL inference computes each layer sequentially not in parallel. The sequential execution allows Occlumency to maintain the feature maps only for a single layer at a time, which is not too large. The system releases the occupied memory for the processed feature map. This way, Occlumency can save memory for storing feature maps significantly.

Here, a challenge is that some models use certain features maps multiple times during inference. For instance, GoogLeNet contains inception modules which consist of multiple convolutional layers sharing the same parent layer, and several layers in ResNets reuse the previous feature maps from past layers. In case of such models, identifying the usage dependency for a feature map is essential.

Occlumency handles this issue by analyzing the structure of the model in advance. Given a model, the usage of each feature map is pre-determined and it is possible to plan when to release each feature map from the enclave memory. In the initialization phase, Occlumency identifies how many future layers access a feature map, and in the inference

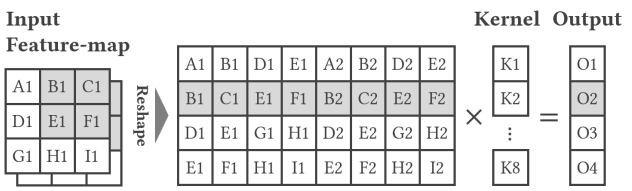

Figure 5: *im2col*-based convolution lowering. (kernel size: 2x2, stride: 1, input feature map size: 3x3)

phase, it releases the feature map after being used for the pre-calculated times.

6.2 Partitioned Convolution

Apart from storing feature maps, convolution layer operations require high memory space (see Table 1), resulting in frequent EPC page swapping and slowdown of inference speed. In particular, we identified that the high memory usage results from *convolution lowering*, the most common method to perform fast convolution operations. Convolution lowering reshapes the input feature map to convert many convolutions into a single matrix multiplication. The reshaped size is much bigger than its original size (e.g., N^2 times bigger when an $N \times N$ kernel is used), causing significant memory overhead. We address the problem by partitioning the convolutions into multiple matrix multiplications.

6.2.1 *Memory Cost of Convolution Lowering.*

We first looked into convolution lowering, which is the main cause of memory usage. Convolution lowering is a common technique to accelerate the computation of convolution layers [23, 45, 52]. In particular, *im2col*-based convolution lowering [23, 47] is widely used in various DL frameworks including Caffe and TensorFlow [14].

As shown in Figure 5, *im2col*-based convolution reshapes the input feature map to convert a convolution operation into single matrix multiplication. Then, linear algebra libraries such as OpenBLAS [74] or Intel MKL [73] optimizes the matrix multiplication using CPU cache based on the fact that the matrix multiplication reads each element of matrix redundantly. The bigger the matrix is, the more redundant access happens to an element. Thus, this optimization becomes more efficient for larger matrices. As a result, *im2col*-based convolution lowering, which converts multiple small matrix multiplications required for convolution into a single large multiplication, is much faster than the *direct* convolution.

However, convolution lowering trades off speed against memory overhead. In particular, the reshaping of the feature map duplicates each feature element multiple times to use matrix multiplication. As shown in Figure 5, the data for "E_1" is copied 4 times through the *im2col* reshaping. The duplication occurs as many times as the size of the kernel

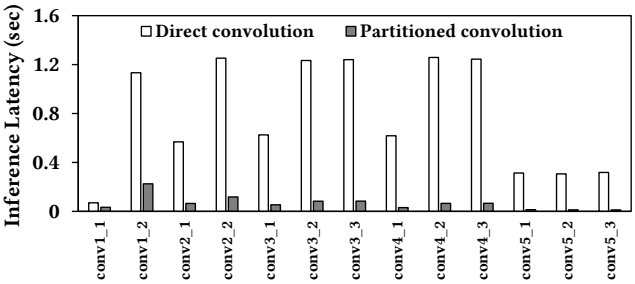

Figure 6: Latency of convolution layers of VGG-16 with direct convolution and partitioned convolution.

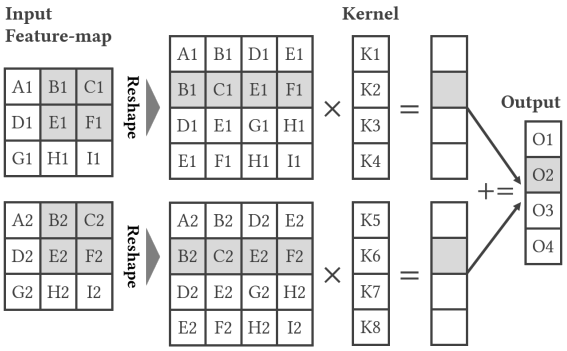

Figure 7: Partitioning convolution by 2.

with stride = 1 (4 in Figure 5). For VGG-16/19 models, the data can be duplicated nine times as they use 3x3 kernels.

A naive way to handle the memory overhead of convolution lowering is to use *direct* convolution without any reshaping of input data. However, a naive direct convolution requires many small matrix multiplications, which cannot take benefits from optimization for large matrix multiplication. Figure 6 shows the slow inference speed of direct convolution. There has been an effort to improve the speed of direct convolution [84], however it requires further investigation to implement it in SGX and evaluate its performance benefit.

As alternatives, FFT-based convolution [38] and Winograd-based convolution [52] are proposed. Nonetheless, these approaches are not designed to reduce the memory overhead, and trade-off inference speed against the memory overhead (which is not suitable for Occlumency where memory space is significantly limited).

6.2.2 *Partitioned Convolution.*

We propose a *partitioned convolution* approach, which requires much less memory than the existing convolution lowering, but still takes advantage of optimization done for large matrix multiplication.

This approach divides a single matrix multiplication, which is derived from the convolution lowering, into multiple small ones. As shown in the equation 1, the matrix multiplication can be expressed as the sum of multiplications of partitioned

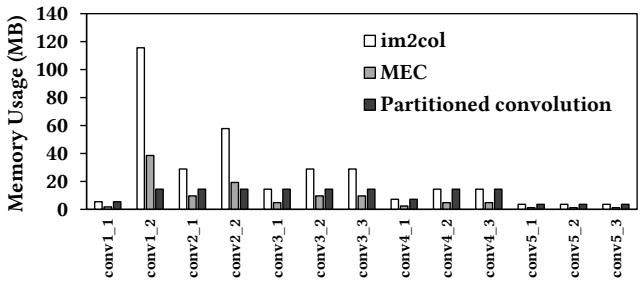

Figure 8: Memory requirement of traditional im2col, MEC, and partitioned convolution for VGG-16.

matrices that part of the original matrix. The size of the partitioned matrix will be reduced by the number of partitions, thereby reducing the required memory space for the convolution layer at a given time.

$$O = \begin{bmatrix} I_1 & I_2 & \vdots\vdots & I_n \end{bmatrix} \cdot \begin{bmatrix} K_1 & K_2 & \vdots\vdots & K_n \end{bmatrix}^\top = \sum_{i=1}^{n} I_i \cdot K_i \quad (1)$$

Figure 7 illustrates how Occlumency computes the input feature map through the convolution layer partitioned by 2. The DL inference engine first lowers half of the input feature map into the im2col buffer, computes matrix multiplication with a half of the kernel matrix, and obtains the partial output. Then, it repeats with the other half of the input feature map to get another partial output. The sum of both partial outputs becomes the final output feature map of the layer. The engine repeats this step as many times as the number of partitions; the im2col buffer is reused for each step.

The number of partitions for a convolutional layer is determined by the memory requirement for inference and available enclave memory. More specifically, Occlumency doubles the number of partitions until the memory space for lowered input feature map requires less than 32 MB of the memory space (here, we observe that 32 MB is the safe lower bound of the available enclave memory for inference). This approach can limit the maximum memory usage by convolution lowering so that Occlumency can operate within the enclave's memory limit regardless of the model.

As a possible alternative, Cho et al. proposed MEC [24] to optimize the memory usage for *im2col*-based convolution lowering. Unlike traditional convolution lowering that lowers input feature maps in both column-wise and row-wise, MEC lowers input feature maps in column-wise only. In the case of VGG-16, MEC requires three times less memory compared to traditional *im2col*-based convolution lowering. However, the memory optimization of MEC is not enough to be applied to Occlumency since it cannot manage memory usage adaptively. The memory reduction ratio of MEC is fixed to the width of the kernel (3 for all layers in VGG-16), and this results in insufficient memory reduction for some layers. Figure 8 shows the memory usage of MEC and partitioned convolution with VGG-16 model. For the layer `conv1_2`, MEC requires ≈40 MB only for convolution lowering and exceeds the EPC size limit with other memory usages such as weights and feature map loading. In case of layer `conv4_1` - `conv5_3`, MEC still reduces the memory usage effectively, but it generates unnecessary computational overhead. Besides, Occlumency determines the number of partitions balancing the memory usage not to exceed the EPC page limit while minimizing the inference latency.

Table 2: Lines of modified or added code

Implementation	Lines of code
Caffe modification	3,388
3rd-party lib modification	2,570
Enclave management	2,697
Hashing, parallelism	3,587
Others	563
Total	12,805

7 IMPLEMENTATION

We have implemented Occlumency based on Caffe [48], a widely-used deep-learning framework, for both Linux and Windows environments. We modified or added 12,805 lines of code in total to implement our system as shown in Table 2.

Building Enclave. The trusted component of Occlumency running in an enclave was built as a shared library with ≈85 MB of `HeapMaxSize` and ≈262 KB of `StackMaxSize`. The trusted components contain the source code of modified Caffe, 3rd-party libraries, and other components of Occlumency including ring buffer, hashing mechanism, and ECALL functions interfacing with the untrusted component.

SGX Compatibility. In SGX enclave, system calls and dynamic-link libraries are forbidden. We ported 3rd-party libraries used in Caffe including OpenBLAS [12], boost [66], and Google protobuf [6]. We blocked all system calls in the 3rd-party libraries and implemented additional interfaces such as file operation to replace the system calls.

Caffe Modification. For ease of implementation, we created Caffe Net in both enclave and outside of the enclave. A net created out of enclave loads trained weights in untrusted memory and passes the address of individual weights into the enclave, and a net in the enclave proceeds the inference. For each layer, it forwards the input feature map with the weights loaded in ring buffer and releases the memory of weights after the forwarding.

Model Integrity Checking. As we explained in § 5.2, Occlumency utilizes three threads for weight loading, hash checking, and inference. These threads share the ring buffer, and they are synchronized by three operations: `read`, `write`, and `consume`. The weight loading thread continuously copies

Table 3: CNN models specifications (Conv: convolutional layer, FC: fully-connected layer)

Model	Input shape	Weight size (MB)			Arch.
		Conv	FC	Total	
AlexNet	227x227	9.1	229.0	238.1	5conv, 3fc
GoogLeNet	224x224	23.3	4.0	27.3	57conv, 1fc
ResNet-50	224x224	91.6	8.0	100.0	53conv, 1fc
ResNet-101	224x224	165.6	8.0	174.4	104conv, 1fc
ResNet-152	224x224	226.5	8.0	235.7	155conv, 1fc
VGG-16	224x224	57.4	483.0	540.4	13conv, 3fc
VGG-19	224x224	78.2	483.0	561.2	16conv, 3fc
YOLO	448x448	235.0	826.4	1061.3	24conv, 2fc

the weights from untrusted memory into weight buffer calling `write` while the hash checking thread and inference thread access the loaded weights with `read`. If the weight is not loaded yet, both hash checking thread and inference thread wait for the weight loading thread. The loaded weights are released from enclave memory when the inference thread calls `consume` after each layer forwarding. For hashing, we used xxHash [8], a high-speed non-cryptographic hash function.

The maximum size of the weight buffer is decided according to the remaining enclave heap size. Since the memory usage for each layer differs, we chose the maximum weight buffer size dynamically. As a result, the maximum weight buffer size $W_{max}(k)$ for kth layer is defined as

$$W_{max}(k) = H - \max(L_k; L_{k+1}; : : : ; L_N) \quad (2)$$

where H denotes the remaining heap size in enclave, L_i denotes the memory used by ith layer, and N denotes the number of layers in the model.

Mobile App. We also implemented an image processing Android app to evaluate the end-to-end performance. It sends images to a server for model inference via Occlumency and receives the results back. The images are re-sized to the shape of the model's input (e.g., 227x227 for AlexNet) and encrypted with AES before transferred.

8 EVALUATION

We evaluate the performance of Occlumency in terms of *inference latency* and *memory usage* with a variety of DNN models. We also measure the *energy saving* of a smartphone by offloading the model inference to a remote server.

8.1 Experimental Setup

Models. We evaluate our system with well-known CNN models shown in Table 3. The models were provided by the Caffe Model Zoo [11], pre-trained with the Imagenet (ILSVRC 2012) dataset [64]. AlexNet and VGG models contain three fully-connected layers with a large weight size. However, GoogLeNet and ResNet models contain only a single fully-connected layer with a very small weight size. Instead, they have a large number of convolutional layers.

Baselines. We compare Occlumency to three baseline approaches: 1) *SGX-paging* that runs model inference inside the enclave with page swapping enabled but without using the optimization techniques of Occlumency, as described in § 2.3, to show the effectiveness of Occlumency in reducing the in-enclave model inference latency; 2) *Native* that runs model inference in unprotected memory outside the enclave, to study the overhead of Occlumency in preserving privacy; and 3) *On-device* that runs model inference on a smartphone, to show the enhanced inference speed and the energy saved in the smartphone by offloading model inference to a remote server.

Hardware. We use a Linux machine equipped with an Intel Core i7-7700 (Kabylake) 3.6GHz CPU of 4 cores and 16 GB DDR4 RAM. As on Windows, EPC page swapping is not supported, and thus we cannot run the *SGX-paging* baseline approach in Windows, we only report the experiment results on Linux. To run the *On-device* baseline approach, we use three smartphones: Nexus 5X (Qualcomm Snapdragon 808 1.8GHz CPU, 2GB RAM), Google Pixel XL (Qualcomm Snapdragon 821 2.15GHz CPU, 4GB RAM) and Google Pixel 3 XL (Qualcomm Snapdragon 845 2.80GHz CPU, 4GB RAM).

8.2 Inference Latency

We first measure the inference latency that is the total time to execute a model on user input to generate the output. All the error bars are only ≈1% and thus not shown in the figures.

Inference speed improvement in SGX enclave. Figure 9 shows the results of inference latency in Occlumency and *SGX-paging*. Occlumency significantly outperforms *SGX-paging*, reducing the inference latency by 3.6x on average, which demonstrates the effectiveness of the optimization techniques used in Occlumency, including the on-demand weights loading, the memory-efficient inference and the parallel processing pipeline. GoogLeNet has only a 1.9x performance improvement despite its small fully-connected layer portion. This is because the memory size required by GoogLeNet is just over 90 MB, which incurs less frequent EPC page swapping than other models.

Inference latency overhead compared to native Caffe. Figure 10 shows the results of inference latency in Occlumency and native Caffe. It shows that compared to native DL inference running outside the enclave, Occlumency still introduces considerable overhead, ranging from 22% to 113% and with an average value of 72%. This overhead is the cost paid for preserving user privacy. AlexNet and VGG-16/19

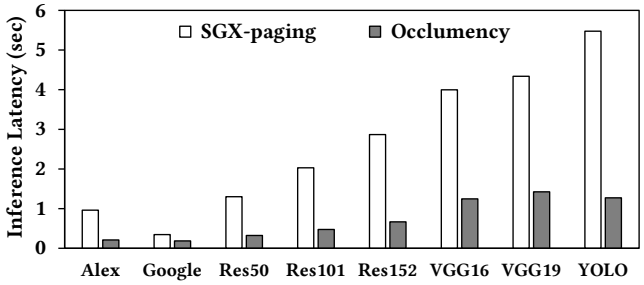

Figure 9: Comparison with *SGX-paging*.

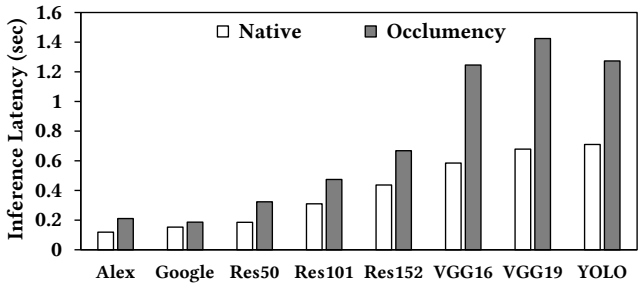

Figure 10: Comparison with *Native*.

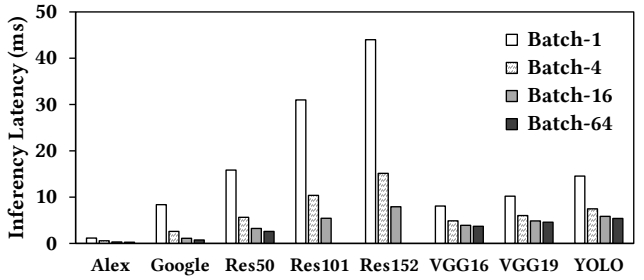

Figure 11: Inference latency of *Native* with GPU.

with large fully-connected layers show a 79% to 113% slowdown, while GoogLeNet and ResNet have a smaller overhead. In particular, the small GoogLeNet model runs only 22% slower than native Caffe.

Comparison with GPU. For further understanding, we also measured the inference latency under *Native* environment using GPU: NVIDIA Tesla P100 16 GB. Figure 11 shows the results with various batch sizes.[2] Compared to Occlumency, GPU-based cloud performs ≈18x faster for GoogLeNet and ResNets, and ≈125x faster for AlexNet and VGG-16/19.

Since Occlumency is a CPU-only solution, it can not take advantage of batching. Also, it is difficult to support batching due to its limited memory. However, with multiple enclaves, Occlumency can support parallel inferences.

Effect of parallelism. As we described in §5.2, Occlumency utilizes three threads for on-demand weight loading, hash checking and DL inference engine to form a parallel processing pipeline. To evaluate the benefit of the parallelism, we

[2]ResNet-101/152 fail to run with batch size >16 due to insufficient memory.

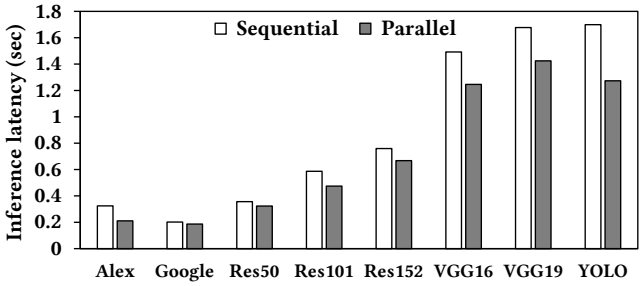

Figure 12: Speed improvement with parallelism.

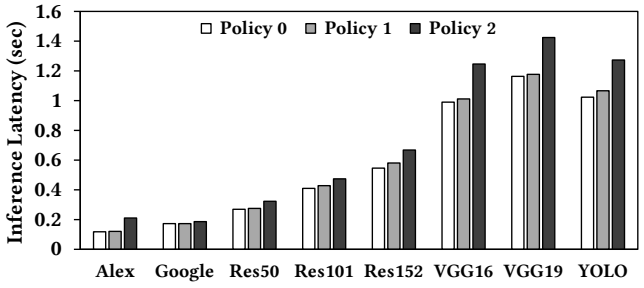

Figure 13: Inference latency of different policies.

Table 4: Weight corruption attacks

Attacks	Description
A_1	Naively corrupt weight before inference
A_2	Corrupt weight right after hashing
A_3	A_2 + restore after inference

Table 5: Model integrity checking policies (O: detect always, △: possibly detect, X: cannot detect)

Policy	Checking scheme	A_1	A_2	A_3
Policy 0	None	X	X	X
Policy 1	Hash only	O	△	X
Policy 2	Weight load & hash	O	O	O

compared the inference latency with and without the parallelism as shown in Figure 12. It shows that the parallelism reduces the inference latency by 17.5% on average.

Alternative model-integrity-checking policies. As discussed in §5.2, we implemented model integrity checking in Occlumency. However, faster inference speed may take priority over high-level integrity in cases where the service is less sensitive to model contamination.

To study the trade-off between model-integrity protection and inference performance, we measured the performance of three model-integrity-checking policies, as shown in Table 5 including whether each policy can address the three weight corruption attacks described in Table 4. *Policy 0* directly reads weights from unprotected memory and does nothing to preserve the model integrity. *Policy 2* is the default policy used in Occlumency as described in §5.2. It does both weights copying and hash checking to ensure model

Table 6: Maximum enclave memory usage of computation for small models (weight loading is excluded)

	AlexNet	Google	Res-50	Res-101	Res-152
Max Memory Usage (MB)	17.1	20.3	22.1	22.5	23.0

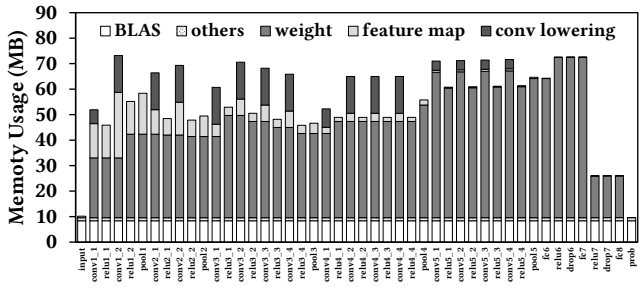

Figure 14: Example memory usage with VGG-19.

integrity. Unlike policy 2, *policy 1* does hash checking but does not copy weights into the enclave. Same to policy 0, it directly reads weights in untrusted memory (after hash check). It is faster than policy 2 but cannot prevent advanced attacks aware of memory access of Occlumency as shown in Table 5. Note that all the three policies preserve user privacy.

Figure 13 shows the result. Even policy 1 does hash checking, it is only about 2 - 4% slower than policy 0, for the benefit of parallelism. However, policy 2 is much slower, 27% slower than the others due to the data copy overhead. AlexNet has an overhead of 79% with policy 2, due to its large fully-connected layers. Fully-connected layers are highly memory intensive with a large number of weights, and thus AlexNet suffers from the overhead of weights copying.

8.3 Enclave Memory Usage

We next measure the memory usage breakdown of Occlumency. Table 6 shows the maximum memory usage (excluding the memory of weights loading) for AlexNet, GoogLeNet, and ResNets. With these models, an inference can be conducted within 25 MB of memory for storing feature maps, forwarding convolutional layer, etc. Occlumency uses fixed 50 MB of enclave memory for the ring buffer to store weights, which leads to more benefits of parallelism.

For the large models including VGG-16/19 and YOLO, the available memory size for the ring buffer is less than 50 MB. Thus, the maximum buffer size for weights changes dynamically. As an example, Figure 14 shows the enclave heap memory utilization for each layer in VGG-19. As CNN layers locate in the front part of models and have large feature maps requiring a large buffer for convolution lowering, we can see that the memory allocated for weights starts with ≈20 MB and increases as each layer is forwarded. By this dynamic memory control, Occlumency ensures the total memory

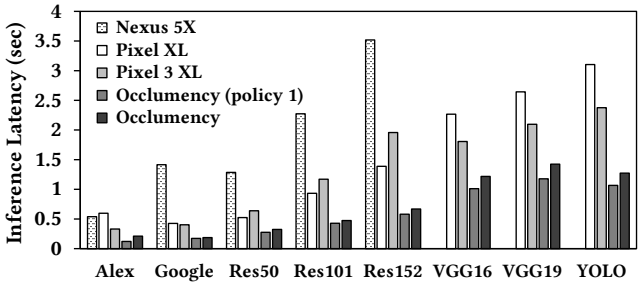

Figure 15: Comparison with *On-device*.

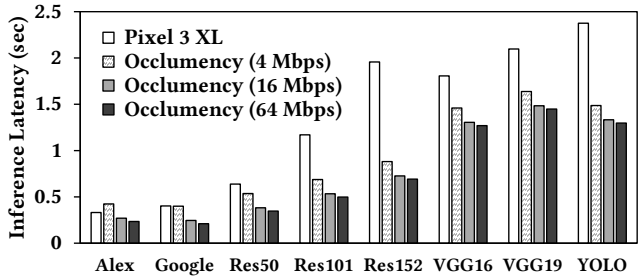

Figure 16: Inference latency under different network bandwidth.

usage is always under the available EPC memory size (85 MB) and thus avoids EPC paging for maximum performance.

8.4 Comparison with On-device Inference

We also compare Occlumency with the *on-device* baseline to show the benefits of remote DL inference in terms of latency reduction and energy saving. We built the original Caffe via Android NDK with arm64-v8a ABI and installed it on the smartphones. For the fair comparison, we used the mobile app described in §7 to measure the end-to-end latency of Occlumency including the network latency. The smartphones are connected to TP-Link AC1900 802.11ac wireless router [5] with 2.4 GHz band and 600 Mbps of maximum bandwidth.

Inference latency. Figure 15 shows the inference latency of running CNN models on the smartphones compared to Occlumency. [3] In average, Occlumency is 2.1x faster than Pixel XL and 2.0x faster than Pixel 3 XL. For a higher speed of Occlumency with policy 1, it achieves inference speed 2.6x faster than Pixel XL and 2.4x faster than Pixel 3 XL. In the case of Nexus 5X, Occlumency is 5.7x and 4.8x faster with policy 1 and 2, respectively. These results demonstrate that remote DL inference can significantly reduce inference latency and thus improve user experience, in addition to saving computation resources of smartphones.

Network conditions. Since the performance of Occlumency is affected by network conditions, we conducted an additional experiment to evaluate Occlumency with limited network bandwidth. As shown in Figure 16, with the exception

[3] Nexus 5X fails to run VGGs and YOLO due to insufficient memory.

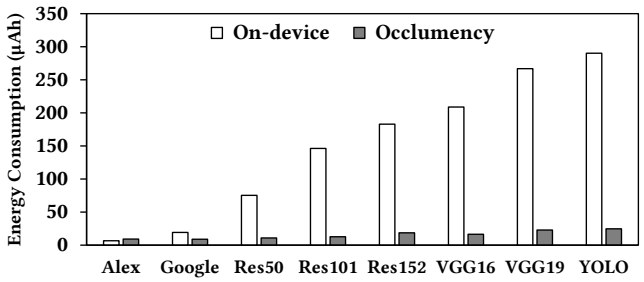

Figure 17: Energy consumption comparison.

Table 7: Compressed models specifications

Alias	Model	Compression Method	Compression Rate
M1	AlexNet	DeepCompression [34]	27x
M2	VGG-16	Low-rank regularization [69]	5x

Table 8: Light-weight architecture models

Alias	Model	Weight Size (MB)	Architecture
M3	SqueezeNet [44]	4.8	26conv
M4	MobileNet [39]	16.6	28conv
M5	MobileNetV2 [65]	13.8	54conv

of AlexNet, Occlumency outperforms *On-device* in case of Pixel 3 XL even with a network bandwidth of 4 Mbps.

Energy consumption. We measured the energy consumption of Pixel XL with Monsoon power monitor [2]. Figure 17 shows the results. Except for AlexNet[4], using clouds to run DL models with Occlumency consumes 2.1 - 11.7x less energy of smartphone than running them on a smartphone. Thus, Occlumency (and remote DL inference in general) can significantly reduce the energy cost on mobile devices.

8.4.1 *Compressed and Light-weight Models.*

As many optimization techniques have been proposed to compress large models into small ones (e.g., weight compression [34, 35], low-rank factorization [49, 69], knowledge distillation [83]) or design new light-weight models [39, 44, 65], one may wonder how Occlumency may compare with running those compressed or light-weight models (tiny models) on device. We did such a comparison using the models in Table 7 and Table 8 and the same Android app.

Figure 18 shows the results. First, we compare the performance of Occlumency and *SGX-paging* with compressed and light-weight models. As shown in Figure 18, it is clear that Occlumency is also beneficial for tiny models. Although tiny models, especially SqueezeNet and MobileNets, have very small weight sizes, they require a memory which exceeds the memory limit of SGX and leads to paging.

[4]AlexNet contains a small number of convolutional layers and requires less computation. As a result, energy cost for network transmission is higher than the reduced energy for the computation by cloud offloading.

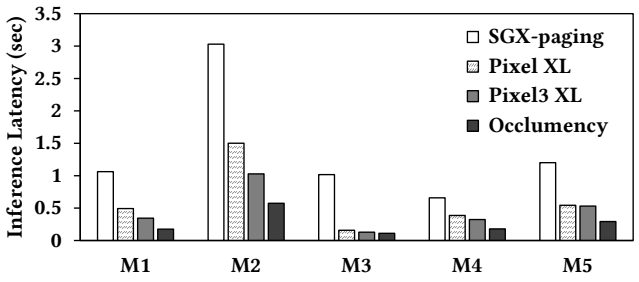

Figure 18: Inference latency of tiny models.

Compared to *on-device* case, in average, Occlumency is 2.2x faster than Pixel XL and 1.7x faster than Pixel 3 XL. This implies that Occlumency is still faster than on-device inference with compressed/light-weight models designed for mobile devices, although the improvement is not always significant. Note that Occlumency is not designed to replace on-device model inference using small models. Instead, we enable privacy-preserving, low-latency remote model inference for the apps that leverage a cloud to run large models.

9 DISCUSSION

Supported models. In this paper, we focused on commonly used CNN models, especially vision models, in Occlumency. Occlumency supports not only common layers used in CNN models including convolution and fully-connected layers but also LSTM and embedding layers for RNN models. We envision that our method adopted in Occlumency can be easily applied in other layers; yet, it cannot support models that require to maintain large feature maps for a long time such as image-to-image models due to insufficient memory.

Though DNN models based on low-resolution images (224x224 - 448x448), which follow the typical practice, are used in our evaluation, models using larger size of input can be supported by Occlumency. It is true that a larger image requires a longer transmission time, but it will also increase the number of computations for inference and maximize the benefits of using the cloud.

Side-channel attacks. Prior studies suggested various side-channel attacks to SGX applications (on memory management [75], and CPU cache [22, 30, 33]) although SGX provides strong confidentiality and integrity of codes and data. In particular, the authors of [70] showed that the side-channel attacks using memory access pattern can extract the sensitive data during the deep-learning inference task in the enclave. Currently, we did not focus on the side-channel attacks. For the future work, we can apply prior techniques such as [70] to protect enclave data from side-channel attacks.

Lack of GPU. GPU cannot be used with the SGX enclave. Building a trusted execution environment for GPU and enabling DL inference in the environment remains as an important future research direction [72]. Occlumency (using the

CPU-based DL execution), however, make faster and energy-efficient DL inference possible compared to on-device DL inference to protect user privacy.

Model training and fine-tuning. Occlumency focuses on model inference. It is difficult to train or fine-tune a model inside SGX enclave, but [58, 82] proposed efficient methods for model fine-tuning. We leave this as a future work.

Hardware accelerators on mobile devices. Recently, dedicated AI chipsets have been built into mobile devices, e.g., Huawei NPU [17]. Despite the increased power of the hardware accelerators, the cloud-driven method is still relevant due to its benefits in energy saving and easy deployment. Further, it is also a trend to build hardware accelerators for the cloud (e.g., Google TPU [16]) and consideration of AI accelerators could be a possible expansion of this study.

10 RELATED WORK

Privacy-preserving distributed deep learning. To protect the user privacy, some previous works [54, 59] leveraged the concept of distributed deep learning. They offload non-private part of the deep learning functionality on to the cloud, whereas the client (e.g., user's smartphone) performs the private computation. These methods, however, incur extra computational and battery overhead on the client side when the design of the DNN pipeline becomes more complicated. They also impose heavy communication overhead to send large feature maps to the cloud. In contrast, Occlumency achieves efficient and transparent execution of DL-inference tasks of a given state-of-the-art DL model.

Other methods for preserving privacy. Various privacy-preserving methods have been proposed even before DL became a hot research topic, such as encryption-based operations [61], randomized noise addition [19], k-anonymity [53], and hybrid deep learning [60]. While such methods may be applied for DL, they are either very computation intensive, or designed for low-dimensional data. Thus, they are hard to be applied to high-dimensional DL, or cannot protect the end-to-end model inference pipeline.

Running ML/DL in SGX. There have been several prior works to protect user information in ML/DL inference tasks in the cloud. Ohrimenko et al. [56] investigated *data obliviousness* for a range of Machine Learning algorithms (e.g., neural networks, support vector machines, and decision trees) applied in SGX. Chiron [41], Ryoan [42], and Slalom [71] used SGX for ML-inference tasks, including both training and inference. They both enable data holders to train ML models on an outsourced server without revealing their training data. None of them addresses the performance issue of SGX or ports a widely-used DL framework like Caffe into SGX.

DeepEnclave [31] aims at securing user input during DL inference using SGX. To overcome the SGX's limited memory size, the system splits the DL network – it executes the first few layers in an enclave, and latter layers outside the enclave. As a result, DeepEnclave only copes with input data protection. However, Occlumency provides much stronger privacy protection, securing not only the user input but also the whole end-to-end pipeline of DL inference including all the intermediate feature maps and the final inference output. Occlumency also preserves model integrity.

Privado [70], to defend side-channel based attacks on DNN models, loads several DNN models inside the SGX. Doing so imposes an even higher requirement for memory usage in the enclave. Our work is complementary with Privado and Occlumency can be used together with Privado to improve the model inference performance.

Low-memory deep learning. Several approaches have been proposed to optimize the DNN models including model compression [51], distillation [83], half-floating point operation [43], parameter quantization [29, 77], and binary layer [62]. These methods not only drop the accuracy of the model, but also mostly focus on reducing the model parameters, which still require large memory size for computation.

MEC [24] and direct convolution [84] have been proposed to optimize memory usage of convolutional layers without any modification of the model. Yet, these methods are not suitable for Occlumency compared to our partitioned convolution as explained in §6.2.2.

11 CONCLUSION

We propose Occlumency, a novel cloud-driven solution to protect user privacy while taking benefit from powerful cloud resources to run large DNN models for high accuracy and low latency. It leverages SGX enclave to execute DL inference and protect user privacy throughout the end-to-end offloading. We implemented Occlumency based on Caffe, and our experiments show that Occlumency improves the in-enclave inference speed by 3.6x and imposes only an overhead of 72% compared to the native inference. Also, it achieves 2.0x faster inference and up-to 91% of energy saving compared to on-device inference.

12 ACKNOWLEDGEMENTS

We thank our shepherd Nicholas D. Lane and the anonymous reviewers for their valuable comments. This work is partly supported by the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (2017R1A2B3010504, 2017M3C4A7065963, 2019R1C1C1006088), NSFC-61872180, NSFC-61802007, and MSRA Collaborative Research 2018 Grant Award.

REFERENCES

[1] General Data Protection Regulation. Retrieved July 18, 2019 from https://eugdpr.org

[2] Monsoon Power Monitor. Retrieved July 18, 2019 from https://www.msoon.com/online-store

[3] ONNX Open Source Model Zoo. Retrieved July 18, 2019 from https://github.com/onnx/models

[4] The Microsoft Cognitive Toolkit. Retrieved July 18, 2019 from https://www.microsoft.com/en-us/cognitive-toolkit

[5] TP-Link AC1900. Retrieved July 18, 2019 from https://www.tp-link.com/us/products/details/cat-9_Archer-C9.html

[6] Protocol Buffers. Retrieved July 18, 2019 from http://code.google.com/apis/protocolbuffers/

[7] ARM Security Technology: Building a Secure System using TrustZone® Technology. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf

[8] xxHash. Retrieved July 18, 2019 from https://cyan4973.github.io/xxHash/

[9] Intel Software Guard Extensions (Intel SGX). Retrieved July 18, 2019 from https://software.intel.com/en-us/sgx

[10] Intel Software Guard Extensions (Intel SGX) SDK. Retrieved July 18, 2019 from https://software.intel.com/en-us/sgx-sdk

[11] Caffe Model Zoo. Retrieved July 18, 2019 from http://caffe.berkeleyvision.org/model_zoo.html

[12] OpenBLAS. Retrieved July 18, 2019 from https://www.openblas.net/

[13] Keystone Enclave: An Open-Source Secure Enclave for RISC-V. Retrieved July 18, 2019 from https://docs.keystone-enclave.org/en/latest/

[14] TensorFlow: An open source machine learning framework for everyone. Retrieved July 18, 2019 from https://www.tensorflow.org/

[15] Facebook Security Breach Exposes Accounts of 50 Million Users. Retrieved July 18, 2019 from https://www.nytimes.com/2018/09/28/technology/facebook-hack-data-breach.html

[16] Google Cloud TPU. Retrieved July 18, 2019 from https://cloud.google.com/tpu

[17] Huawei Kirin 970 - HiSilicon. Retrieved July 18, 2019 from https://en.wikichip.org/wiki/hisilicon/kirin/970

[18] Microsoft Azure Cognitive Services. Retrieved July 18, 2019 from https://azure.microsoft.com/en-us/services/cognitive-services/

[19] Dakshi Agrawal and Charu C. Aggarwal. 2001. On the Design and Quantification of Privacy Preserving Data Mining Algorithms. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '01)*. ACM, New York, NY, USA, 247–255. https://doi.org/10.1145/375551.375602

[20] Hany Hassan amd Anthony Aue, Chang Chen, Vishal Chowdhary, Jonathan Clark, Christian Federmann, Marcin Junczys-Dowmunt Xuedong Huang, William Lewis, Mu Li, Shujie Liu, Tie-Yan Liu, Renqian Luo, Arul Menezes, Tao Qin, Frank Seide, Xu Tan, Fei Tian, Lijun Wu, Shuangzhi Wu, Yingce Xia, Dongdong Zhang, Zhirui Zhang, and Ming Zhou. 2018. Achieving Human Parity on Automatic Chinese to English News Translation. (March 2018). https://www.microsoft.com/en-us/research/uploads/prod/2018/03/final-achieving-human.pdf

[21] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, David Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. 2016. SCONE: Secure Linux Containers with Intel SGX. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, Berkeley, CA, USA, 689–703. http://dl.acm.org/citation.cfm?id=3026877.3026930

[22] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software Grand Exposure: SGX Cache Attacks Are Practical. *CoRR* abs/1702.07521 (2017). arXiv:1702.07521 http://arxiv.org/abs/1702.07521

[23] Kumar Chellapilla, Sidd Puri, and Patrice Simard. 2006. High Performance Convolutional Neural Networks for Document Processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*, Guy Lorette (Ed.). Université de Rennes 1, Suvisoft, La Baule (France). https://hal.inria.fr/inria-00112631

[24] Minsik Cho and Daniel Brand. 2017. MEC: Memory-efficient Convolution for Deep Neural Network. In *Proceedings of the 34th International Conference on Machine Learning (ICML '17)*, Vol. 70. PMLR, Sydney, NSW, Australia, 815–824. http://proceedings.mlr.press/v70/cho17a.html

[25] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* (2016), 86. http://eprint.iacr.org/2016/086

[26] Victor Costan, Ilia A. Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *25th USENIX Security Symposium (USENIX Security '16)*. USENIX Association, Austin, TX, 857–874. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan

[27] Tom Woller David Kaplan, Jeremy Powell. AMD memory encryption. http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf

[28] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of The 33rd International Conference on Machine Learning (ICML '16)*, Vol. 48. PMLR, New York, NY, USA, 201–210. http://proceedings.mlr.press/v48/gilad-bachrach16.html

[29] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir D. Bourdev. 2014. Compressing Deep Convolutional Networks using Vector Quantization. *CoRR* abs/1412.6115 (2014). arXiv:1412.6115 http://arxiv.org/abs/1412.6115

[30] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache Attacks on Intel SGX. In *Proceedings of the 10th European Workshop on Systems Security (EuroSec'17)*. ACM, New York, NY, USA, Article 2, 6 pages. https://doi.org/10.1145/3065913.3065915

[31] Zhongshu Gu, Heqing Huang, Jialong Zhang, Dong Su, Ankita Lamba, Dimitrios Pendarakis, and Ian Molloy. 2018. Securing Input Data of Deep Learning Inference Systems via Partitioned Enclave Execution. *CoRR* abs/1807.00969 (2018). arXiv:1807.00969 http://arxiv.org/abs/1807.00969

[32] Shay Gueron. A Memory Encryption Engine Suitable for General Purpose Processors. Cryptology ePrint Archive, Report 2016/204. https://eprint.iacr.org/2016/204

[33] Marcus Hähnel, Weidong Cui, and Marcus Peinado. 2017. High-Resolution Side Channels for Untrusted Operating Systems. In *2017 USENIX Annual Technical Conference (ATC '17)*. USENIX Association, Santa Clara, CA, 299–312. https://www.usenix.org/conference/atc17/technical-sessions/presentation/hahnel

[34] Song Han, Huizi Mao, and William J. Dally. 2015. Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. *CoRR* abs/1510.00149 (2015). arXiv:1510.00149 http://arxiv.org/abs/1510.00149

[35] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both Weights and Connections for Efficient Neural Network. In *Advances in Neural Information Processing Systems 28 (NIPS '15)*. Curran Associates, Inc., Montreal, Quebec, Canada, 1135–1143. http://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-network

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '16)*. Las Vegas, NV, USA, 770–778. https://doi.org/10.1109/CVPR.2016.90

[37] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. 2017. CryptoDL: Deep Neural Networks over Encrypted Data. *CoRR* abs/1711.05189 (2017). arXiv:1711.05189 http://arxiv.org/abs/1711.05189

[38] Tyler Highlander and Andres Rodriguez. 2016. Very Efficient Training of Convolutional Neural Networks using Fast Fourier Transform and Overlap-and-Add. *CoRR* abs/1601.06815 (2016). arXiv:1601.06815 http://arxiv.org/abs/1601.06815

[39] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR* abs/1704.04861 (2017). arXiv:1704.04861 http://arxiv.org/abs/1704.04861

[40] Jie Hu, Li Shen, and Gang Sun. 2018. Squeeze-and-Excitation Networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '18)*. Salt Lake City, UT, USA, 7132–7141. https://doi.org/10.1109/CVPR.2018.00745

[41] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. 2018. Chiron: Privacy-preserving Machine Learning as a Service. *CoRR* abs/1803.05961 (2018). arXiv:1803.05961 http://arxiv.org/abs/1803.05961

[42] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. 2016. Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, Berkeley, CA, USA, 533–549. http://dl.acm.org/citation.cfm?id=3026877.3026919

[43] Loc N. Huynh, Youngki Lee, and Rajesh Krishna Balan. 2017. DeepMon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '17)*. ACM, New York, NY, USA, 82–95. https://doi.org/10.1145/3081333.3081360

[44] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *CoRR* abs/1602.07360 (2016). arXiv:1602.07360 http://arxiv.org/abs/1602.07360

[45] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up Convolutional Neural Networks with Low Rank Expansions. In *Proceedings of the British Machine Vision Conference (BMVC '14)*. BMVA Press, Nottingham, UK. https://doi.org/10.5244/C.28.88

[46] Yujie Ji, Xinyang Zhang, Shouling Ji, Xiapu Luo, and Ting Wang. 2018. Model-Reuse Attacks on Deep Learning Systems. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. ACM, New York, NY, USA, 349–363. https://doi.org/10.1145/3243734.3243757

[47] Yangqing Jia. 2014. *Learning Semantic Image Representations at a Large Scale*. Ph.D. Dissertation. University of California, Berkeley, USA. http://www.escholarship.org/uc/item/64c2v6sn

[48] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia (MM '14)*. ACM, New York, NY, USA, 675–678. https://doi.org/10.1145/2647868.2654889

[49] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. 2015. Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications. *CoRR* abs/1511.06530 (2015). arXiv:1511.06530 http://arxiv.org/abs/1511.06530

[50] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25 (NIPS '12)*. Curran Associates, Inc., Lake Tahoe, Nevada, USA, 1106–1114. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

[51] Nicholas D. Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. 2016. DeepX: A Software Accelerator for Low-power Deep Learning Inference on Mobile Devices. In *Proceedings of the 15th International Conference on Information Processing in Sensor Networks (IPSN '16)*. IEEE Press, Piscataway, NJ, USA, Article 23, 12 pages. http://dl.acm.org/citation.cfm?id=2959355.2959378

[52] Andrew Lavin and Scott Gray. 2016. Fast Algorithms for Convolutional Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '16)*. Las Vegas, NV, USA, 4013–4021. https://doi.org/10.1109/CVPR.2016.435

[53] Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. 2005. Incognito: Efficient Full-domain K-anonymity. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (SIGMOD '05)*. ACM, New York, NY, USA, 49–60. https://doi.org/10.1145/1066157.1066164

[54] Meng Li, Liangzhen Lai, Naveen Suda, Vikas Chandra, and David Z. Pan. 2017. PrivyNet: A Flexible Framework for Privacy-Preserving Deep Neural Network Training with A Fine-Grained Privacy Control. *CoRR* abs/1709.06161 (2017). arXiv:1709.06161 http://arxiv.org/abs/1709.06161

[55] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David M. Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. 2017. ROTE: Rollback Protection for Trusted Execution. In *26th USENIX Security Symposium (USENIX Security '17)*. USENIX Association, Vancouver, BC, 1289–1306. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/matetic

[56] Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious Multi-Party Machine Learning on Trusted Processors. In *25th USENIX Security Symposium (USENIX Security '16)*. USENIX Association, Austin, TX, 619–636. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/ohrimenko

[57] Oleksii Oleksenko, Bohdan Trach, Robert Krahn, Mark Silberstein, and Christof Fetzer. 2018. Varys: Protecting SGX Enclaves from Practical Side-Channel Attacks. In *2018 USENIX Annual Technical Conference (ATC '18)*. USENIX Association, Boston, MA, 227–240. https://www.usenix.org/conference/atc18/presentation/oleksenko

[58] Maxime Oquab, Léon Bottou, Ivan Laptev, and Josef Sivic. 2014. Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '14)*. Columbus, OH, USA, 1717–1724. https://doi.org/10.1109/CVPR.2014.222

[59] Seyed Ali Ossia, Ali Shahin Shamsabadi, Ali Taheri, Kleomenis Katevas, Hamid R. Rabiee, Nicholas D. Lane, and Hamed Haddadi. 2017. Privacy-Preserving Deep Inference for Rich User Data on The Cloud. *CoRR* abs/1710.01727 (2017). arXiv:1710.01727 http://arxiv.org/abs/1710.01727

[60] Seyed Ali Ossia, Ali Shahin Shamsabadi, Ali Taheri, Hamid R. Rabiee, Nicholas D. Lane, and Hamed Haddadi. 2017. A Hybrid Deep Learning Architecture for Privacy-Preserving Mobile Analytics. *CoRR* abs/1703.02952 (2017). arXiv:1703.02952 http://arxiv.org/abs/1703.02952

[61] Antonis Papadimitriou, Ranjita Bhagwan, Nishanth Chandran, Ramachandran Ramjee, Andreas Haeberlen, Harmeet Singh, Abhishek Modi, and Saikrishna Badrinarayanan. 2016. Big Data Analytics over Encrypted Datasets with Seabed. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, Berkeley, CA, USA, 587–602. http://dl.acm.org/citation.cfm?id=3026877.3026922

[62] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands*. Springer International Publishing, Cham, 525–542. https://doi.org/10.1007/978-3-319-46493-0_32

[63] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '16)*. Las Vegas, NV, USA, 779–788. https://doi.org/10.1109/CVPR.2016.91

[64] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[65] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '18)*. Salt Lake City, UT, USA, 4510–4520. http://openaccess.thecvf.com/content_cvpr_2018/html/Sandler_MobileNetV2_Inverted_Residuals_CVPR_2018_paper.html

[66] Boris Schäling. 2011. *The boost C++ libraries*. Boris Schäling.

[67] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *CoRR* abs/1409.1556 (2014). arXiv:1409.1556 http://arxiv.org/abs/1409.1556

[68] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR '15)*. Boston, MA, USA, 1–9. https://doi.org/10.1109/CVPR.2015.7298594

[69] Cheng Tai, Tong Xiao, Xiaogang Wang, and Weinan E. 2015. Convolutional neural networks with low-rank regularization. *CoRR* abs/1511.06067 (2015). arXiv:1511.06067 http://arxiv.org/abs/1511.06067

[70] Shruti Tople, Karan Grover, Shweta Shinde, Ranjita Bhagwan, and Ramachandran Ramjee. 2018. Privado: Practical and Secure DNN Inference. *CoRR* abs/1810.00602 (2018). arXiv:1810.00602 http://arxiv.org/abs/1810.00602

[71] Florian Tramèr and Dan Boneh. 2018. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. *CoRR* abs/1806.03287 (2018). arXiv:1806.03287 http://arxiv.org/abs/1806.03287

[72] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. 2018. Graviton: Trusted Execution Environments on GPUs. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'18)*. USENIX Association, Berkeley, CA, USA, 681–696. http://dl.acm.org/citation.cfm?id=3291168.3291219

[73] Endong Wang, Qing Zhang, Bo Shen, Guangyong Zhang, Xiaowei Lu, Qing Wu, and Yajuan Wang. 2014. Intel math kernel library. In *High-Performance Computing on the Intel® Xeon Phi™*. Springer, 167–188.

[74] Qian Wang, Xianyi Zhang, Yunquan Zhang, and Qing Yi. 2013. AUGEM: Automatically Generate High Performance Dense Linear Algebra Kernels on x86 CPUs. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '13)*. ACM, New York, NY, USA, Article 25, 12 pages. https://doi.org/10.1145/2503210.2503219

[75] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A. Gunter. 2017. 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 (CCS '17)*. ACM, New York, NY, USA, 2421–2434. https://doi.org/10.1145/3133956.3134038

[76] Nico Weichbrodt, Pierre-Louis Aublin, and Rüdiger Kapitza. 2018. Sgxperf: A Performance Analysis Tool for Intel SGX Enclaves. In *Proceedings of the 19th International Middleware Conference (Middleware '18)*. ACM, New York, NY, USA, 201–213. https://doi.org/10.1145/3274808.3274824

[77] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. 2016. Quantized Convolutional Neural Networks for Mobile Devices. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '16)*. Las Vegas, NV, USA, 4820–4828. https://doi.org/10.1109/CVPR.2016.521

[78] Wayne Xiong, Lingfeng Wu, Fil Alleva, Jasha Droppo, Xuedong Huang, and Andreas Stolcke. 2017. The Microsoft 2017 Conversational Speech Recognition System [Technical Report]. (August 2017). https://www.microsoft.com/en-us/research/publication/microsoft-2017-conversational-speech-recognition-system/

[79] Mengwei Xu, Jiawei Liu, Yuanqiang Liu, Felix Xiaozhu Lin, Yunxin Liu, and Xuanzhe Liu. 2019. A First Look at Deep Learning Apps on Smartphones. In *The World Wide Web Conference (WWW '19)*. ACM, New York, NY, USA, 2125–2136. https://doi.org/10.1145/3308558.3313591

[80] Mengwei Xu, Mengze Zhu, Yunxin Liu, Felix Xiaozhu Lin, and Xuanzhe Liu. 2018. DeepCache: Principled Cache for Mobile Deep Vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18)*. ACM, New York, NY, USA, 129–144. https://doi.org/10.1145/3241539.3241563

[81] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In *2015 IEEE Symposium on Security and Privacy (SP '15)*. San Jose, CA, USA, 640–656. https://doi.org/10.1109/SP.2015.45

[82] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *Advances in Neural Information Processing Systems 27 (NIPS '14)*. Curran Associates, Inc., Montreal, Quebec, Canada, 3320–3328. http://papers.nips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks

[83] Xiao Zeng, Kai Cao, and Mi Zhang. 2017. MobileDeepPill: A Small-Footprint Mobile Deep Learning System for Recognizing Unconstrained Pill Images. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '17)*. ACM, New York, NY, USA, 56–67. https://doi.org/10.1145/3081333.3081336

[84] Jiyuan Zhang, Franz Franchetti, and Tze Meng Low. 2018. High Performance Zero-Memory Overhead Direct Convolutions. In *Proceedings of the 35th International Conference on Machine Learning (ICML '18)*, Vol. 80. PMLR, Stockholmsmässan, Stockholm Sweden, 5776–5785. http://proceedings.mlr.press/v80/zhang18d.html