Using Intel SGX to Improve Private NN Training and Inference

Ryan Karl, Jonathan Takeshita, Taeho Jung
University of Notre Dame
{rkarl,jtakeshi,tjung}@nd.edu

Abstract
In our modern data-driven society, the importance of leveraging machine learning (ML) algorithms to make critical business and government decisions continues to grow. To dramatically improve performance, such algorithms are often outsourced to the cloud, but within privacy and security sensitive domains, this presents several challenges to data owners for ensuring that their data is protected from malicious parties. One practical solution to these problems comes from Trusted Execution Environments (TEEs), which utilize several hardware and software based technologies to isolate sensitive computations from untrusted software. This paper investigates a new technique utilizing a TEE to allow for the high performance training and execution of Deep Neural Networks (DNNs), an ML algorithm that has recently been used with great success in a variety of challenging tasks, including face and speech recognition.

Background and Applications

Machine learning (ML) is increasingly used in a variety of data driven decision making settings where security is of paramount importance. However, given the explosive growth in the popularity of cloud-based ML frameworks, which hide the complexity of ML algorithms from users, the number of attack points continues to grow. Such frameworks generally require that a data owner trust the cloud provider with either direct access to their data or to run computations over their data at a remote location, which places the privacy of the data at risk [1].

Trusted Execution Environments (TEEs), e.g, Intel SGX, ARM TrustZone, etc., offer a practical solution to this problem. TEEs use a variety of hardware and software technologies to isolate potentially sensitive code from untrusted applications, while still providing users with the functionality to attest their code was correctly executed without any tampering from an adversary [8]. It has been shown that within the context of outsourced ML computations, TEEs outperform pure cryptographic approaches by several orders of magnitude [10].

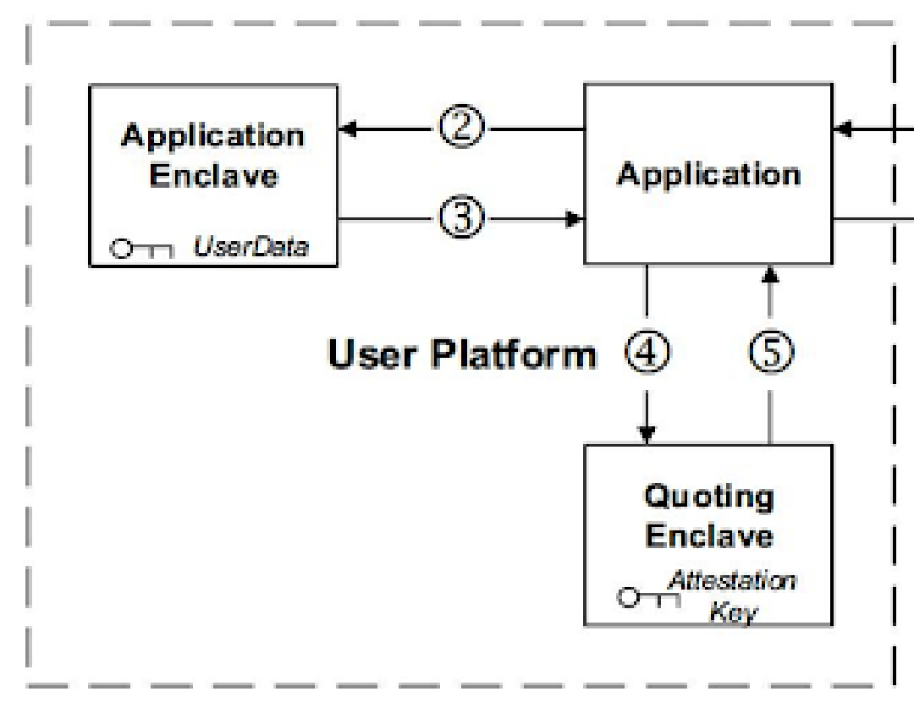

Fig. 1 Diagram of SGX Enclave Creation

Our approach, inspired by the Slalom framework and verifiable ASICs [10], is notably different from ML outsourcing based on purely cryptographic methods [6, 9]. With this framework, computations are delegated between two co-located processors, which support an outsourcing protocol with efficiency that is orders-of-magnitude faster than existing work, while not requiring that the DNN be executed or trained fully in a TEE. By utilizing Freivald's algorithm [5], an efficient verifiable scheme that allows for outsourced matrix multiplication, we can support private ML training and inference. We intend to formally prove the correctness and privacy of our scheme, and evaluate it on multiple canonical DNNs such as VGG19, ResNet, etc. It has been shown that compared to running all computations in SGX, outsourcing linear layers to an untrusted GPU increases throughput by 4x to 11x for verifiable and private inference [10], and we expect a similar performance improvement when using this approach for training.

1 Related Work

Although trusted hardware enclaves have been shown in the past to offer a powerful platform for data analysis [2], it is still challenging to effectively integrate their functionalities for machine learning tasks. More specifically, it is difficult to manage the large size of the data needed while processing the algorithms within the space constraints of the enclave, and allowing for the efficient training of the models within the enclaves while still allowing for flexible updates to model architecture and hyperparameters continues to be difficult. Also, protecting against side-channel attacks further complicates the use of TEEs for ML tasks [4].

Previous work introduced Chiron, a system for privacy-preserving Machine Learning as a service (MLaaS) [6]. Note that the MLaaS framework assumes that an individual data provider will train their own ML model using hardware and algorithms owned by an untrusted party. Chiron manages to increase processing throughput by distributing the model training over multiple enclaves, at the cost of a small decrease in model accuracy. Another similar work features a setup where several data providers train a shared ML model [9]. This paper does not focus on leveraging differential privacy to confront problems relating to model performance or data size, and instead is interested in hiding memory access patterns for SGX based training to avoid side-channel attacks. A similar paper describes Myelin [7], a deep learning framework which combines ideas from these privacy-preserving techniques to build a system for fully private ML. This work must protect multiple data providers from other malicious participants, to ensure they cannot learn the details of other users' data during the training or inference phases. To maximize efficiency, their system is constructed to support multi-threaded computation inside of enclaves. Myelin enclaves use their own optimized libraries built with the TVM complier for training data-oblivious DL models via incrementally fetched data, and its modular nature makes distributed training possible.

Our work is most similar to the Slalom framework [10], which allows for efficient privacy-preserving NN inference (but not training) using via a TEE. They leverage a cryptographic blinding method along with Freivald's algorithm [5], a method for delegating matrix multiplication to an untrusted GPU that can be verified for correctness after the computation is complete. This technique allows for improved performance over a single-threaded enclave.
Contribution: Our work composes DL training in such a way that the one-time pad scheme can be used in tandem with the activation function so that training can be supported and delegated in an iterative fashion to a co-located GPU, and efficiency can be further enhanced by utilizing the TVM complier [3] to optimize a DL library to fit within the TEE. Our approach will expedite the private training and inference using a TEE by allowing more computations to occur in untrusted GPU without needing to frequently retrieve intermediate results for processing in the TEE.

2 Our Protocol

For our inference method, inputs and weights are first quantized and embedded in a field $\mathcal{F}$. Following this, the linear layers are outsourced and then verified via Freivald's algorithm [5]. Finally, the inputs of the linear layers are encrypted with a pre-computed pseudorandom stream (base on the one-time pad scheme) to guarantee their privacy. To simplify notation, we assume there is only one node per layer in the given neural network, but this could be easily extended to architectures with multiple nodes per layer. Notice that only two values are needed to continue the backpropagation algorithm for a given layer of the neural network: (1) the cost C_o where $C_o = a^{(L)} - y)^2$ where $a^{(L)}$ is the activation at layer L (L is the total number of layers) and y is the output of the neural network, and (2) the activation at layer L, denoted $a^{(L)} = \sigma(w^{(L)}a^{(L-1)} + b^{(L)}$ where $w^{(L)}$ is the weight at layer L, $a^{(L-1)}$ is the activation at layer $L-1$, and $b^{(L)}$ is the bias at layer L. For ease of notation we represent the input of the activation function in future sections as $z^{(L)} = w^{(L)}a^{(L-1)} + b^{(L)}$. We model the dependencies of the values needed to compute the backpropagation algorithm as shown in Figure 1.

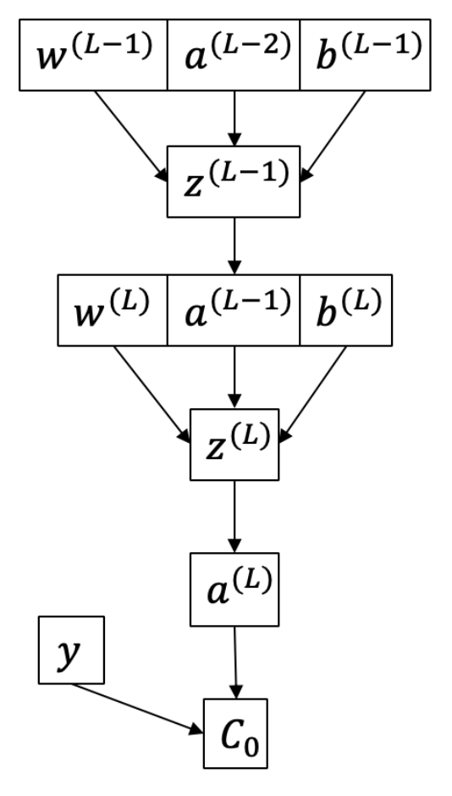

Fig. 2 Diagram of Backpropagation Dependencies

To compute backpropagation, we take the partial derivative of C_o with respect to $w^{(L)}$, with respect to $b^{(L)}$, and with respect to $a^{(L-1)}$. More formally, we would compute theses three equations:

$$\frac{\partial C_o}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial C_o}{\partial a^{(L)}} = a^{(L-1)}\sigma'(z^{(L)})2(a^{(L)-y}) \quad (1)$$

$$\frac{\partial C_o}{\partial b^{(L)}} = \frac{\partial z^{(L)}}{\partial b^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial C_o}{\partial a^{(L)}} = \sigma'(z^{(L)})2(a^{(L)-y}) \quad (2)$$

$$\frac{\partial C_o}{\partial a^{(L-1)}} = \frac{\partial z^{(L)}}{\partial a^{(L-1)}}\frac{\partial a^{(L)}}{\partial z^{(L)}}\frac{\partial C_o}{\partial a^{(L)}} = w^{(L)}\sigma'(z^{(L)})2(a^{(L)-y}) \quad (3)$$

Now with many activation functions, such as the sigmoid function, it is not immediately clear how to build an additive masking scheme as described above to recover the values after delegating the computations to an untrusted cloud. For example, notice that for the sigmoid function, which we write as $\sigma(k) = \frac{1}{1+e^{-k}}$ and the derivative as $\sigma'(k) = \frac{1}{1+e^{-k}}(1 - \frac{1}{1+e^{-k}}) = \sigma(k)(1-\sigma(k))$, if we were to mask the input x with an additive noise r, for the first layer of the network we would have $w^{(0)}(x+r) + b^{(0)} = w^{(0)}x + w^{(0)}r + b^{(0)} = a^{(0)} + w^{(0)}r$. If we input this into σ, we have $\sigma(a^{(0)} + w^{(0)}r) = \frac{1}{1+e^{-(a^{(0)})+w^{(0)}r}} = \frac{1}{1+e^{-(a^{(0)})}e^{w^{(0)}r}}$. If we compute the derivative σ' we have $\sigma'(a^{(0)} + w^{(0)}r) = \frac{1}{1+e^{-a^{(0)}}e^{(w^{(0)}r)}}(1 - \frac{1}{1+e^{-a^{(0)}}e^{(w^{(0)}r)}})$. This means that we would need to compute the three equations described previously and somehow remove the random masking value from (a nontrivial task):

$$\frac{\partial C_o}{\partial w^{(L)}} = a^{(L-1)}\frac{1}{1+e^{-a^{(L)}}e^{(w^{(L)}r)}}(1 - \frac{1}{1+e^{-a^{(L)}}e^{(w^{(L)}r)}})2(a^{(L)-y}) \quad (4)$$

$$\frac{\partial C_o}{\partial b^{(L)}} = \frac{1}{1+e^{-a^{(L)}}e^{(w^{(L)}r)}}(1 - \frac{1}{1+e^{-a^{(L)}}e^{(w^{(L)}r)}})2(a^{(L)-y}) \quad (5)$$

$$\frac{\partial C_o}{\partial a^{(L-1)}} = w^{(L)}\frac{1}{1+e^{-a^{(L)}}e^{(w^{(L)}r)}}(1 - \frac{1}{1+e^{-a^{(L)}}e^{(w^{(L)}r)}})2(a^{(L)-y}) \quad (6)$$

Note the Arctan function is $\sigma(k) = tan^{-1}(k)$ and its derivative is $\sigma'(k) = \frac{1}{k^2+1})$, and we would need to remove the random mask from the following equations (which is possible if r is known):

$$\frac{\partial C_o}{\partial w^{(L)}} = a^{(L-1)}\frac{1}{(w^{(L)}r + a^{(L)})^2+1}2(a^{(L)-y}) \quad (7)$$

$$\frac{\partial C_o}{\partial b^{(L)}} = \frac{1}{(w^{(L)}r + a^{(L)})^2+1}2(a^{(L)-y}) \quad (8)$$

$$\frac{\partial C_o}{\partial a^{(L-1)}} = w^{(L)}\frac{1}{(w^{(L)}r + a^{(L)})^2+1}2(a^{(L)-y}) \quad (9)$$

Note the SoftMax function is $\sigma(k) = log(1+e^k)$ and its derivative is $\sigma'(k) = (\frac{1}{1+e^{-x}})$. This means that we would need to remove the random mask from the following equations:

$$\frac{\partial C_o}{\partial w^{(L)}} = a^{(L-1)}\frac{1}{1+e^{-a^{(L)}}e^{(w^{(L)}r)}}2(a^{(L)-y}) \quad (10)$$

$$\frac{\partial C_o}{\partial b^{(L)}} = \frac{1}{1+e^{-a^{(L)}}e^{(w^{(L)}r)}}2(a^{(L)-y}) \quad (11)$$

$$\frac{\partial C_o}{\partial a^{(L-1)}} = w^{(L)}\frac{1}{1+e^{-a^{(L)}}e^{(w^{(L)}r)}}2(a^{(L)-y}) \quad (12)$$

Although further work is needed to prove the correctness and privacy of this method for delegating computation between the SGX and an untrusted processor, this discussion presents what we hope is an intuitive introduction to how we intend to support private NN training via SGX.

1. **Initialize:** The TEE is initialized with the model F and the input x_1, and the untrusted server S is initialized with F.
2. **Preprocess:** For each element $i \in [1,n]$, the TEE generates random masking value r_i by randomly sampling $r_i \leftarrow F^{m_i}$, and then calculates the associated demasking value as $u_i = r_iW_i$. They then mask the input x_i as $\widetilde{x}_i = x_i + r_i$
3. **Online:** For each element $i \in [1,n]$, the TEE sends $\widetilde{x}_i$ to the untrusted server S, who computes $\widetilde{y}_i = \widetilde{x}_i\widetilde{W}_i$, and sends $\widetilde{y}_i$ back to the TEE.
4. **Verify:** For each element $i \in [1,n]$, the TEE checks that the untrusted S computed $\widetilde{y}_i$ correctly by computing $y_i = \widetilde{y}_i - u_i$ and calling Freivalds(y_i, x_i, W_i). If Freivalds successfully verifies that the $\widetilde{y}_i$ is correct, we continue (otherwise we know that S behaved maliciously and may abort). The TEE then computes the activation function as $x_{i+1} = \sigma(y_i)$.
5. **Return:** The algorithm then returns y_n.

Note the TEE outsources computation of n layers of a model F to the untrusted server S. Each layer is defined by a matrix W_i of size $m_i \times n_i$, followed by activation σ. All operations are over a field $\mathcal{F}$. The Freivalds(y_i, x_i, w_i), function performs k repetitions of Freivalds' check. The pseudorandom elements r_i and precomputed values u_i are only used once.

References

[1] Bae, H., Jang, J., Jung, D., Jang, H., Ha, H., Yoon, S.: Security and privacy issues in deep learning (2018)
[2] Chen, F., Wang, C., Dai, W., Jiang, X., Mohammed, N., Al Aziz, M.M., Sadat, M.N., Sahinalp, C., Lauter, K., Wang, S.: Presage: privacy-preserving genetic testing via software guard extension (2017)
[3] Chen, T., Moreau, T., Jiang, Z., Shen, H., Yan, E., Wang, L., Hu, Y., Ceze, L., Guestrin, C., Krishnamurthy, A.: Tvm: end-to-end optimization stack for deep learning (2018)
[4] Costan, V., Devadas, S.: Intel sgx explained. (2016)
[5] Freivalds, R.: Probabilistic machines can use less running time. (1977)
[6] Hunt, T., Song, C., Shokri, R., Shmatikov, V., Witchel, E.: Chiron: Privacy-preserving machine learning as a service (2018)
[7] Hynes, N., Cheng, R., Song, D.: Efficient deep learning on multi-source private data (2018)
[8] McKeen, F., Alexandrovich, I., Berenzon, A., Rozas, C.V., Shafi, H., Shanbhogue, V., Savagaonkar, U.R.: Innovative instructions and software model for isolated execution. (2013)
[9] Ohrimenko, O., Schuster, F., Fournet, C., Mehta, A., Nowozin, S., Vaswani, K., Costa, M.: Oblivious multi-party machine learning on trusted processors (2016)
[10] Tramer, F., Boneh, D.: Slalom: Fast, verifiable and private execution of neural networks in trusted hardware (2018)