2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS) | 978-1-6654-4513-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICDCS51616.2021.00077

Privacy-Preserving Neural Network Inference Framework via Homomorphic Encryption and SGX

Huizi Xiao*, Qingyang Zhang†, Qingqi Pei* and Weisong Shi‡
*State Key Laboratory of ISN, School of Telecommunications Engineering, Xidian University, Xi'an, China, 710071
†School of Computer Science and Technology, Anhui University, Hefei, China, 230601
‡Department of Computer Science, Wayne State University, Detroit, MI, USA., 48202
{huizi_xiao@stu, qqpei@mail}.xidian.edu.cn, qingyang@thecarlab.org, weisong@wayne.edu

Abstract—**Edge computing is a promising paradigm that pushes computing, storage, and energy to the networks' edge. It utilizes the data nearby the users to provide real-time, energy-efficient, and reliable services. Neural network inference in edge computing is a powerful tool for various applications. However, edge server will collect more personal sensitive information of users inevitably. It is the most basic requirement for users to ensure their security and privacy while obtaining accurate inference results. Homomorphic encryption (HE) technology is confidential computing that directly performs mathematical computing on encrypted data. But it only can carry out limited addition and multiplication operation with very low efficiency. Intel software guard extension (SGX) can provide a trusted isolation space in the CPU to ensure the confidentiality and integrity of code and data executed. But several defects are hard to overcome due to hardware design limitations when applying SGX in inference services. This paper proposes a hybrid framework utilizing SGX to accelerate the HE-based convolutional neural network (CNN) inference, eliminating the approximation operations in HE to improve inference accuracy in theory. Besides, SGX is also taken as a built-in trusted third party to distribute keys, thereby improving our framework's scalability and flexibility. We have quantified the various CNN operations in the respective cases of HE and SGX to provide the foresight practice. Taking the connected and autonomous vehicles as a case study in edge computing, we implemented this hybrid framework in CNN to verify its feasibility and advantage.**

Index Terms—**edge computing; neural network inference; privacy-preserving; Intel SGX; homomorphic encryption.**

I. Introduction

Smart devices are increasingly deployed in our daily life. They are designed to continuously collect data to bring us better services [1]. Neural network is a practical approach to produce inference services with data generated by the above devices for application, such as image processing, face recognition, medical treatment, and behavior analysis in different domains [2]. As a matter of course, the centralized computing paradigm that uploads all data to the cloud for neural network inference will cause communication congestion, increase latency, and aggravate the energy dissipation of smart devices [3, 4]. Edge computing is a promising paradigm that pushes computing, storage, and energy to the edge of the network where data is generated [5]. It can provide neural network inference services with low latency, stable bandwidth, and high reliability. Nevertheless, users must upload their privacy information actively to edge servers to obtain accurate inference services. Meanwhile, the edge server is closer to the smart devices in the internet of everything compared with the cloud computing center in the core network, so it will collect more personal sensitive information of users inevitably. Therefore, the neural network inference in the edge server is more urgent to protect privacy. Taking the connected and autonomous vehicles (CAVs) scenario as an example in edge computing, the car no longer is a simple means of transportation but will evolve into a sophisticated computer on wheels with the continuous increase of on-board sensors and corresponding services provided [6, 7]. CAVs have become the ideal mobile edge computing servers obviously for providing smart devices with complex neural network inference applications. However, the works [8] have already mentioned that collecting personal and terminal data in autonomous driving will leak privacy to service providers and car manufacturers while providing better services. To attract more and more companies and users to adopt their inference application products with confidence, the edge service providers should design corresponding solutions and implementation plans for privacy leakage problems to eliminate users' security and privacy concerns.

Homomorphic encryption (HE) based neural network approaches enable data owners to send encrypted data to edge servers. The service providers directly perform neural network inference on encrypted data and return results in the ciphertext [9]. Since plaintext information has never appeared, security and privacy are protected while accessing HE-based inference. But HE has several operations that cannot be calculated directly, and the computational efficiency is remarkably low. In the inference process of convolutional neural network (CNN), non-polynomial functions, such as Sigmoid and ReLU, need to be approximated by polynomials that will cut down the inference performance drastically and reduce its accuracy [10, 11], so there is a tradeoff between accuracy and computational cost. Meanwhile, a trusted third party is introduced to distribute various keys during the inference preparation stage with HE.

Software guard extension (SGX) proposed by Intel provides a trusted execution environment, which can provide chip-level security for the integrity and confidentiality of data and code in the unreliable environment [12]. With SGX, applications can run in a container called enclave, which shields potentially compromised software from malware and even privileged software, like operating systems, hypervisor, etc. As a hardware-based protection technology, SGX can effectively and efficiently ensure neural network algorithms' confidential execution. However, it is challenging to execute a large-scale network inference task alone due to hardware limitations. Enclave has only limited memory. Thus the pages need to be swapped in and out frequently when the network scale becomes more extensive, which increases the system overhead to guarantee the integrity and confidentiality of the pages [13]. Because SGX is in user mode, it cannot implement some functions by itself and needs to interact with the untrusted operating system in neural network preference services. This behavior will increase security risks and expand side-channel attacks that SGX cannot resist by itself [14]. Furthermore, it has a high economic cost as an exclusive hardware resource.

In the process of using SGX for CNN inference, the introduction of HE can reduce some dangerous behaviors that only use SGX in a large-scale network inference task. Simultaneously, SGX can substitute the part that cannot be calculated directly in HE-based inference to cut down the approximate behaviors. Therefore, we propose this hybrid framework combining HE and SGX to conduct neural network inference in confidential. The proposed framework improves accuracy theoretically and enhances the efficiency and throughput of HE-based inference as much as possible. Such a continuum of cryptographic schemes and hardware technology ensures the confidentiality and privacy of users' data. The main contributions of this work are as follows.

- We proposed a hybrid CNN inference framework that utilizes SGX to accelerate HE-based inference by reducing the approximation operations in HE. The accuracy and throughput of the HE-based inference can be improved theoretically. SGX distributes keys directly to avoid introducing an additional trusted third party, thereby improving the scalability and flexibility.
- We quantified the running time of some basic HE operations in the CNN inference process and compared them with corresponding operations in SGX. We also tested and analyzed each layer's performance in the framework under HE and SGX, respectively, to summarize their characteristics and advantages. This work provides a forward-looking comparative analysis for the combination of HE and SGX in CNN.
- Considering the application scenario of CAVs in edge computing, we implemented and evaluated our hybrid inference framework in an elaborated CNN model, which contains most of the typical operations. Our scheme reduces 39.615% of the inference time compared to the pure HE-based scheme while maintaining network prediction accuracy.

The rest of this paper is organized as follows. In Section II, we presented the background knowledge. Some shortcomings of the previous schemes constituted problem statement in Section III. We propose our hybrid neural network inference framework combining HE and SGX in Section IV. Section V discusses experiment conditions and analyze measurement methodology. We measured and analyzed the experimental data of different operations in the framework to summarize its characteristics and advantages in Section VI. Section VII implemented a case study of CAVs using a CNN model in edge computing. We discussed the problems and challenges revealed by the experiment in Section VIII. Section IX surveyed the related works. We concluded in Section X.

II. Background

A. Neural Networks

Neural Network is a class of machine learning algorithms using multiple layers to extract higher-level features from the raw input progressively. CNN is a specific kind of neural network algorithms. Typically, a CNN consists of several convolutional layers, pooling layers, fully connected layers, and activation layers to extract features and obtain the inference results.

1) Convolutional Layer: The convolution operation is widely used in image processing, and it can be regarded as a feature extraction method independent of each pixel's position. Different convolution kernels can learn different features, such as edge, line, and corner, of each pixel in a picture, successively. Typically, the convolutional layer operations are linear, only including addition and multiplication with model parameters.

2) Pooling Layer: The pooling layer is also called the downsampling layer, which is sandwiched between continuous convolutional layers. It compresses the feature maps by cutting down the data dimension and parameters and reduce overfitting phenomenons. The most common pooling functions of the pooling layer include max-pooling and mean-pooling. The max-pooling outputs the maximum of a kernel sub-area. The mean-pooling outputs the average of a kernel sub-area.

3) Fully Connected Layer: Each neuron of the fully connected layer is linked to each neuron of the last layer with a weighted sum. It plays the role of a classifier in the entire CNN. The convolutional layer, pooling layer, and activation layer goals are to represent the original data to the feature space. The fully connected layer is to map the feature representation to the label space. Its specific operation is the same as the operation of the convolutional layer. This reduces the influence of feature location on classification significantly.

4) Activation Layer: If every processing layer used in the network is linear, then the multi-layer neural network has only linear mapping capabilities. Thus, a neural network containing merely convolutional layers and fully connected layers can only classify the linear recognition tasks [10]. In this case, an activation layer is used to realize non-linear function achieving the purpose of non-linear mapping. There are several activation such as Sigmoid ($\sigma(x) = \frac{1}{1+e^{-x}}$), Relu ($f(x) = max(0, x)$), Tanh ($tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$) and Leaky Relu ($f(x) = max(\alpha x, x)$) function in practice.

B. *Homomorphic Encryption*

Homomorphic encryption belongs to the field of cryptography. Since HE supports the ability to perform calculations such as addition and multiplication over the encrypted data space without decryption, there is an excellent application demand. We will use Fan-Vercauteren (FV) [15] in this paper, and other HE schemes are available in the same way.

The ring $\mathbb{Z}_a[x]/(x^n+1)$ is represented as R_a where the coefficients reduced modulo is a and n is a power of 2. The plaintext space is taken as R_t for integer t is called plaintext modulus. The ciphertext space is the ring R_q which q is coefficient modulus. The error distribution $\mathcal{X}$ is a truncated discrete Gaussian distribution, $e \leftarrow \mathcal{X}$ to denote that e is sampled uniformly from $\mathcal{X}$. Let $\Delta = \lfloor q/t \rfloor$ and $q = \Delta t + r_t(q)$ where $r_t(q) = q \bmod t$. The letter w is a base into which ciphertext elements are decomposed during relinearization. Let λ be the security parameter. We mainly use seven algorithms that including SecretKeyGen, PublicKeyGen, Encrypt, Decrypt, Add, Multiply and EvaluationKeyGen to generate homomorphic computing code in our framework.

• SecretKeyGen(1^λ): sample $\boldsymbol{s} \leftarrow \mathcal{X}$ and set $\boldsymbol{sk} = \boldsymbol{s}$.

• PublicKeyGen($\boldsymbol{sk}$): set $\boldsymbol{s} = \boldsymbol{sk}$, sample $\boldsymbol{a} \leftarrow R_q$, $\boldsymbol{e} \leftarrow \mathcal{X}$ and output $\boldsymbol{pk} = (\boldsymbol{p}_0, \boldsymbol{p}_1) = ([-(\boldsymbol{as}+\boldsymbol{e})]_q, \boldsymbol{a})$.

• Encrypt($\boldsymbol{pk}, \boldsymbol{m}$): For $\boldsymbol{m} \in R_t$, sample $\boldsymbol{u}, \boldsymbol{e}_1, \boldsymbol{e}_2 \leftarrow \mathcal{X}$ and output

$$\boldsymbol{ct} = (\boldsymbol{c}_0, \boldsymbol{c}_1) = ([\boldsymbol{p}_0 \cdot \boldsymbol{u} + \boldsymbol{e}_1 + \Delta \cdot \boldsymbol{m}]_q, [\boldsymbol{p}_1 \cdot \boldsymbol{u} + \boldsymbol{e}_2]_q).$$

• Decrypt($\boldsymbol{sk}, \boldsymbol{c}$): set $\boldsymbol{s} = \boldsymbol{sk}$, compute

$$\left[\left\lfloor \frac{t \cdot [\boldsymbol{c}_0 + \boldsymbol{c}_1 \cdot \boldsymbol{s}]_q}{q} \right\rceil\right]_t.$$

• Add($\boldsymbol{ct}_0, \boldsymbol{ct}_1$): output

$$([\boldsymbol{ct}_0[0] + \boldsymbol{ct}_1[0]], [\boldsymbol{ct}_0[1] + \boldsymbol{ct}_1[1]]).$$

• Multiply($\boldsymbol{ct}_0, \boldsymbol{ct}_1$): output

$$c_0 = \left[\left\lfloor \frac{t \cdot \boldsymbol{ct}_0[0]\boldsymbol{ct}_1[0]}{q} \right\rceil\right]_q,$$

$$c_1 = \left[\left\lfloor \frac{t}{q}(\boldsymbol{ct}_0[0]\boldsymbol{ct}_1[1] + \boldsymbol{ct}_0[1]\boldsymbol{ct}_1[0]) \right\rceil\right]_q,$$

$$c_2 = \left[\left\lfloor \frac{t \cdot \boldsymbol{ct}_0[1]\boldsymbol{ct}_1[1]}{q} \right\rceil\right]_q.$$

• EvaluationKeyGen($\boldsymbol{sk}, w$): The relinearization key is generated by w and $\boldsymbol{sk}$, which is used for relinearization after multiplication to reduce the noise and ciphertext size so that it can be decrypted correctly.

III. Problem Statement

A. *Homomorphic-enabled Neural Network Inference*

HE enables data owners to send encrypted data to the edge servers. The service providers can perform neural network inference on the encrypted data directly without any information about plaintext in the entire process and then return the inference results in encrypted form. This HE-based inference can achieve the purpose of protecting the security and privacy of user's input queries. Fig. 1 illustrates the model of exchange information data flows between different entities in the HE-based inference. Users decrypt the results to obtain what they want using the private key acquired from a trusted third party in advance. Since HE only supports a limited type of calculations and needs to get the available keys before starting service, so there are the following implementations and problems.

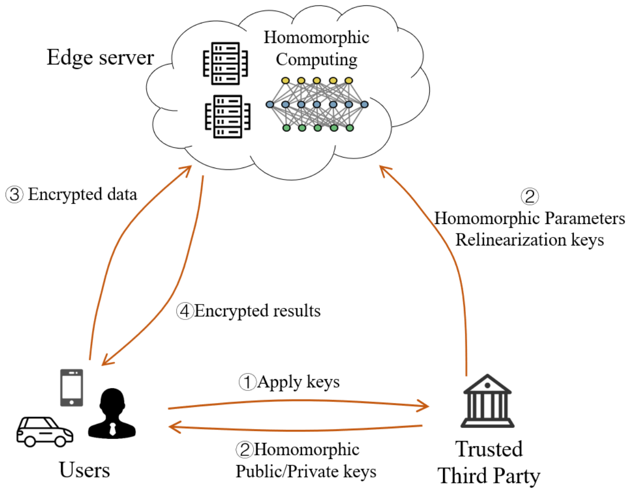


Fig. 1: HE-based neural network inference with a trusted third party.

Trusted third party. Users and the edge server need to use a trusted third party as traditional public key infrastructure (PKI) to issue homomorphic keys and parameters in advance. This is a strong hypothesis, which is to the disadvantage of providing flexible services for users. It is challenging to find a wholly trusted third party in a practical environment. The communication process also increases the probability of eavesdropping and tampering in the keys distribution process.

Convolutional layers and fully connected layers. They are all linear layers and have similar essential operating characteristics, and they can be disassembled into operations that contain multiplication and addition merely. These are operations directly supported by fully homomorphic encryption.

Pooling layers. Comparing and seeking the maximum value or obtaining the average value in the pooling layers are not directly supported by the homomorphic algorithms. So the scaled mean-pooling function that is obtaining the summation of a sub-area is being used to replace mean-pooling. Summation computing is easy to calculate over encrypted data. The only effect is to magnify the output value of this layer several times, which then propagates to the next layer [16]. Actually, this kind of numerical diffusion does not exist in the traditional polling layers, which is a necessity to be considered.

Activation layers. The functions used in the activation layers are all non-polynomial to acquire the ability to solve the non-linear tasks of neural network. There is no excellent way to calculate them directly in encrypted form, which only can utilize high-order polynomials with addition and multiplication to fit the trend of its curve or even look up the table to get complex activation function results. These approximate behaviors are at the expense of cutting down the inference performance drastically and reduce its accuracy [10, 11].

Relinearization. As shown in the FV scheme in Section II-B, if there are the operations of Multiply($\boldsymbol{ct}_0, \boldsymbol{ct}_1$) in the CNN, the ciphertext size will be larger, and the noise will increase exponentially. This will easily beyond the scope of the ciphertext space so that it cannot be decrypted correctly. In HE-based inference, the relinearization is needed to reduce the size of the ciphertext and noise, then ensure that the ciphertext is decrypted correctly. This needs EvaluationKeyGen($\boldsymbol{sk}, w$) to generate relinearization keys. The process requires private key information, which must be produced by the trusted third party in advance and sent to the edge server.

B. SGX-enabled Neural Network Inference

The computation in SGX has excellent efficiency and precision. The attestation allows the remote users to gain confidence that the intended software is running within an enclave on an Intel SGX enabled platform [17]. If the edge server's CPU has the function of SGX, the simplest method is to put the CNN inference into the enclave for computing. But many problems make it unsuitable for excessive or complete use in neural network inference scenarios.

Limited memory. The edge server may have different neural network models for different recognition tasks. The various parameters will be exploded when the network scale increases sharply, and the memory occupied by model storage will increase accordingly. The available physical memory of the enclave is too small. Due to the beginning of the paging and exchanging page, it will cause significant performance degradation.

Side-channel attacks. Because enclave is in user mode, it cannot execute some functions by itself and needs to interact with untrusted operating systems [14]. The exchanging page caused by the small memory mentioned above exactly is a behavior pattern that can be analyzed to become a kind of side-channel attacks [18].

Charges. SGX is a CPU-based hardware security mechanism, and its robust, trusted, and flexible security function is guaranteed by the extended performance of the hardware. As exclusive limited-resource hardware, leasing fees on the edge server are also a consideration.

IV. Our Model

We devise a hybrid neural network inference framework of HE and SGX based on the analysis in Section III. The purpose is to protect the security and privacy of the user's inquiry input. The inference framework avoids introducing an additional trusted third party to distribute all kinds of keys but relies on the build-in remote attestation function of Intel SGX service system. And we utilize SGX to assist in the computing of HE-based inference that is difficult to calculate

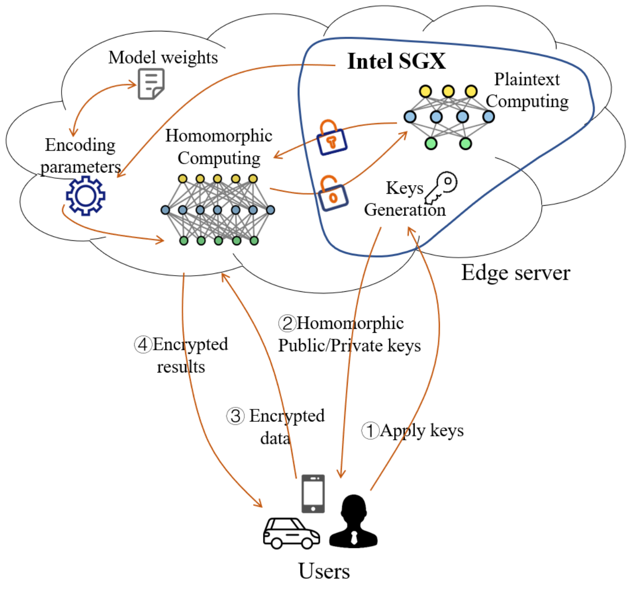


Fig. 2: Hybrid neural network inference framework of HE and SGX.

or needs to be approximated. Fig. 2 illustrates the proposed hybrid framework, which is elaborated as follows.

A. Keys Distribution

The process of proving that the enclave has been established in a secure hardware environment is referred to as remote attestation [19]. This service is provided by Intel and implemented by users using the open-source Intel SGX Data Center Attestation Primitives [20]. When we perform the remote attestation during the set-up phase, the authentication report structure can provide additional user data fields to deliver user-defined information to support more complex interaction structures. Therefore, we can generate the homomorphic parameters and public/private keys in SGX and send public/private to users as customized data. In this way, we can avoid introducing an additional trusted third party into the edge service provider system to distribute keys and homomorphic parameters.

B. Model Weight Parameters Encoding

The inference model is the network structure and weight parameters essentially. We assume that the CNN model parameters have been deployed in the edge server securely after model training or other economic transactions. Because of the service company's reputation and interests, there is no need to tamper with the parameters of the original model with good performance. Before the edge server starts homomorphic computing, the homomorphic parameters must be obtained from SGX ahead of time. The model weight parameters can be encoded in the homomorphic plaintext space and enabled to perform corresponding homomorphic computing outside SGX. This encoded behavior is completed once and used permanently before the service update requires the network to be retrained.

C. Homomorphic Computing outside SGX

After users obtain the homomorphic public/private key, the public key is used to encrypt the data to be inquired and submitted to the edge server. The service provider performs homomorphic computing outside SGX for linear computing, such as convolutional layers and fully connected layers. These are performed in the untrusted part, and the model parameters are involved in the calculation. In this way, we can avoid transferring model parameters to SGX to reduce some dangerous behaviors such as too frequent memory access, exchanging pages, and interaction between internal and external to cut down the side-channel attacks of using SGX.

D. Plaintext Computing in SGX

According to the analysis of each layer in HE-based inference, activation layers and pooling layers do not require parameter weights, and the non-linearization degree of them is awfully high that needs homomorphic approximation. Putting them into SGX to decrypt, conduct plaintext-like computing, then homomorphically encrypt the results to perform the successive homomorphic computing outside of SGX. This can accelerate the HE-based inference with the slightest extension of the dangerous behavior of side-channel attacks. And the non-polynomial functions accurately executed, thereby improving the inference accuracy theoretically.

E. Relinearization

In the ciphertext computing phase outside SGX, if there is a ciphertext multiplication that requires relinearization and noise reduction, it can be passed into SGX to decrypt and re-encrypt, which removes the noise accumulation in the ciphertext naturally. Usually, the evaluating party needs to inform the key generating party in advance whether they need to relinearize, which requires several communication processes. But in this way, we can thoroughly avoid introducing an additional trusted third party to generate relinearization keys to the edge server, thereby improving the scalability and flexibility of the proposed framework.

V. Implementation

A. Prototype Implementation

We have implemented our hybrid HE and SGX CNN inference framework using C++ language based on the Intel SGX driver 2.5.0, the Intel SGX SDK 2.6.100, and the homomorphic library SEAL 2.1 [21]. The selection of HE scheme parameters are polynomial $x^{1024} + 1$ and plaintext modulus $t = 4$. The coefficient modulus q is optimized and selected automatically by the function $ChooserEvaluator :: default\_parameter\_options().at(1024)$ provided by the library.

B. Measurement Analysis

We quantified the time consuming of some basic HE operations in the CNN inference process and compared them with corresponding operations in SGX. We also evaluated the performance and characteristics of each layer in the framework. There are many users at any time in edge application scenarios, and the neural network structure we built makes it possible to predict $batchSize$ encrypted images at a time. We set $batchSize = 10$ in the experiment, which increases the throughput and processing speed of inference. We used Cryptonets [16] as the HE-based CNN inference scheme to compare.

VI. Evaluation

The performance of our framework is evaluated on an SGX-enabled computer, which has an Intel Xeon CPU E3-1225 v6@3.30GHz with four cores, and the operating system version is Ubuntu 16.04.6 LTS 64-bits.

A. Encoding/Decoding and Encryption/Decryption

Keys generation. SGX replaces the function of a trusted third party, generates homomorphic public keys and private keys, then transmits them to users through the remote attestation. TABLE I quantifies the difference in the process of generating a pair of public/private keys. The generation time was measured 1000 times inside and outside SGX, respectively, and the same parameters and key generation procedure plus the time assigned to the global variables. The only difference is the execution environment. The experimental results demonstrate that generating a pair of homomorphic public/private keys in SGX will bring more average time cost of $29.392ms$. We also measured the time it takes to call the $ecall\_generate\_key()$ outside SGX. Most of the time is the same as the time measured in the enclave, and a few will be $1ms$ longer on the millisecond scale. This is the time consumed by entering and exiting SGX. The standard deviation (STD) and the 96% confidence interval (CI) also show that the performance of generating a pair of public/private keys measured in SGX is less stable than the outside.

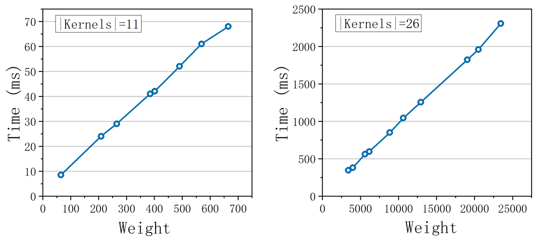
	Average	STD	96% CI
Inside SGX	49.593	3.448	[49.054, 50.132]
Outside SGX	20.201	0.774	[20.062, 20.341]

TABLE I: A pair of public/private keys generation time ($/ms$).

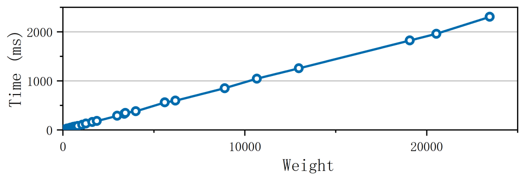
Image encryption. Each pixel of the image is encoded into a polynomial in the plaintext space and encrypted into the ciphertext space. The time of image encoding and encryption are counted once every $batchSize$ and repeated 1000 times. The statistic data of the encoding and encryption time of $batchSize$ images are shown in TABLE II. The average time is 157.013s. The STD and the 96% CI indicating that the results of our experimental data are stable. Therefore, the time cost to encode and encrypt an image is about 15.7s.

$batchSize$	Average	STD	96% CI
10	157.013	1.613	[156.409, 157.617]

TABLE II: Image encoding and encryption time ($/s$).



(a) Fixing the number of kernels, changing kernel size



(b) Changing the number of kernels and kernel size

Fig. 3: The time of weights coding against its number.

Weight parameters encoding. The model is obtained when the weights are generated through training. The edge server can encode the weights in advance that be regarded as preliminary work. The weights are divided into the value of kernels and bias. The number is related to the kernel size and the number of kernels, respectively. As shown in Fig. 3(a), fixing the number of kernels is 11 and 26 respectively, changing the kernel size causes the number of weights to change correspondingly. It can be seen that the encoding time has a linear relationship with the weights' number. In Fig. 3(b), changing the size and number of kernels simultaneously, their encoding time is still linear. The conclusion is that apart from the number of weights that need to be encoded, the encoding time is hardly affected by other factors.

Inference results decryption. After the edge server completes the CNN prediction, it obtains the encrypted results, which need to be transmitted to the requesting user. The user gets the encrypted result and decrypts it using the private key to get the solicited plaintext inference result. We have counted the inferences of $batchSize$ images, and an image inference generates 10 homomorphic data. The statistical experiment data are in TABLE III. There are a total of 100 homomorphic data that need to be decrypted and decoded. The process is looped 100 times. The average decryption and decoding time of each image inference result is $6.2391ms$. We also measured the average time cost of performing an Encoding + Encryption or Decoding+Decryption in TABLE IV, then compared them with the same operations outside SGX on the millisecond scale. Encoding and encryption in SGX bring an average extra cost of 6.042 ms, decryption and decoding in SGX bring a delay of 4.882 ms.

$batchSize$	Average	STD	96% CI
10	62.391	0.941	[61.962, 62.821]

TABLE III: Decryption and decoding of $batchsize$ image inference results ($/ms$).

	Encoding+Encryption	Decoding+Decryption
Inside SGX	18.167 ms	5.250 ms
Outside SGX	12.125 ms	0.368 ms

TABLE IV: An Encoding+Encryption time vs. a Decoding+Decryption time inside and outside SGX respectively.

B. Kernel Size and Homomorphic Convolution

The essence of convolutional layers and fully connected layers are convolutional computing. It can be calculated by HE outside SGX in our framework. This section conducts a time analysis on the relationship between a homomorphic convolution and its kernel size. As can be seen from Section II-A, convolution consists of multiplication and addition operations only. For a feature map, different convolution kernel sizes will lead to different code loop jumps structure and the number of multiplication and addition, resulting in different time delays. We take a 28×28 feature map as input, and the kernel size ranges from 1×1 to 28×28 with an interval of 1. The simplified expression is 1 to 28 in the abscissa of Fig. 4, and the convolution stride is 1. The blue line is the number of ciphertexts multiplied by plaintexts $(C \times P)$ or ciphertexts added by ciphertexts $(C + C)$ in the convolution of a feature map.

Fig. 4 shows that the kernel size of 14 and 15 is the axis of symmetry to get the maximum number of calculations of 44100 times. The kernel on both sides are symmetrical to produce the same number of $C \times P$ and $C + C$, so it should take equal time to calculate, but the small kernel cause more calculation delays than the large kernel, obviously. When kernel size is 14 or 15, the same calculation number is 44100, but it causes a time difference of 0.757s. This time-consuming phenomenon is more serious when it goes to both sides, according to the red line. When the kernel size is 1 and 28, there is a time difference of 15.855s, which is $16.66\times$ the time of the entire convolution when the kernel size is 28. So this extra time-consuming is a multiple of the calculation itself. This is because the small kernel enters the internal homomorphic version of the loop structure more times than the large kernel of the same calculative operation. The homomorphic computing is much slower, making this difference too significant, and even this effect is dominant for the computing time to some extent. This time-consuming effect brought by the homomorphic convolution loop structure cannot be ignored when using homomorphic convolution.

C. SGX and Sigmoid

To calculate the highly non-polynomial activation function accurately and quickly, we put it into SGX for computing. We use Sigmoid as the representative of the activation function.

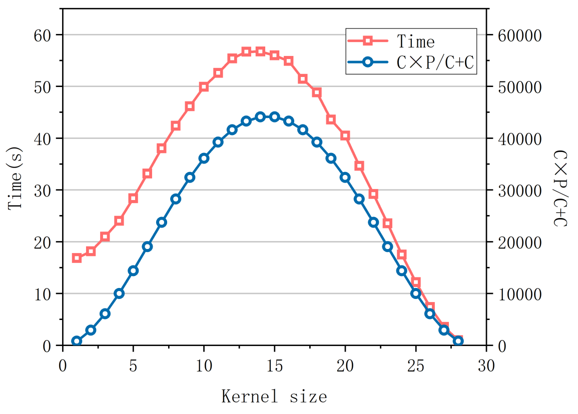


Fig. 4: Homomorphic convolutional time and the different number of $C \times P/C + C$ calculations with different kernel size.

The secondary axis in Fig. 5 represents the *number of calculations*, which is the number of feature values to perform the nonlinear mapping. The primary axis is the activation calculation time of a feature map.

Square computing is used as the function of Sigmoid in HE-based CNN inference, and the ciphertext is relinearized after multiplication. The calculation time of $EncryptSigmoid$ is the red line. The calculated time of SGX is expressed as the yellow line of $SGXSigmiod$, which contains intact variable assignment, ciphertext loading, encryption, decryption, and function computing. The green one of $FakeSGXSigmoid$ indicates the time it takes to execute code outside SGX that the same as in enclave. It can be seen that the more amount of calculations give rise to the more severe time-consuming. Even though the computation in enclave reveals an increment from $34ms$ to $5.62s$ more than the outside time, it still has a significant time advantage from $190ms$ to $37.431s$ than the encrypted one. This shows the efficiency of putting the activation function into SGX for calculation. Many researchers propose fitting the activation function with a higher-order polynomial to obtain more accurate results, which will obviously bring more significant computational cost. There is a tradeoff between accuracy and efficiency, and SGX enables the calculation of diverse activation functions (e.g., Relu and Tanh) flexibly, accurately, and quickly.

D. SGX and Pooling

To use the original classic pooling function in HE-based inference, we put it into SGX to calculate, and our pooling implementation is mean-pooling. We set the input feature map size of the pooling layer to 24.

The line graph based on the secondary axis of Fig. 6 represents the size of the feature map input to SGX for computation. The size of the feature map represents the number of calculations that need to be processed. There are two situations,

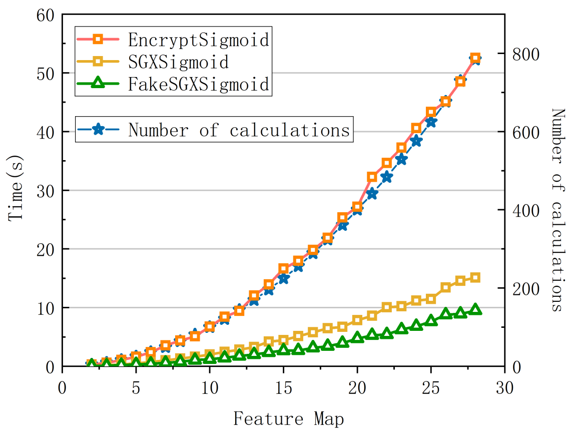


Fig. 5: **Sigmoid computing time with/without SGX.** The blue line in the second legend indicates the number of Sigmoid calculations in a feature map, and the three colored lines in the first legend indicate the Sigmoid computing time of a feature map.

- $SGXDiv$: Use HE to calculate the addition of the pooling window (kernel), so the new feature map obtained will become smaller, and then input it to SGX to calculate the nonlinear division to obtain the mean value.
- $SGXPool$: Input the feature map into SGX directly, and calculate the addition and division to get the mean value. Therefore, regardless of the pooling window size, the size of the feature map input to SGX is 24.

The histogram based on the primary axis of Fig. 6 counts the time of four groups under the condition of changing the polling window size. The statistical time includes the entire variable assignment, ciphertext loading, encryption, decryption, and computation. We set $FakeSGXDiv$ and $FakeSGXPool$ for comparing the time loss induced by SGX itself. There are the following four groups,

- $SGXDiv$: To find the mean value of the pooling window, the time of homomorphic addition $EncryptedSum$ plus the time of division in SGX $SGXDivide$.
- $FakeSGXDiv$: The time to execute code outside SGX that the same as $SGXDiv$, which are $EncryptedSum$ plus $FakeSGXDivide$.
- $SGXPool$: To find the mean value of the pooling window, the time of addition and division in SGX.
- $FakeSGXPool$: The time to execute code outside SGX that the same as $SGXPool$.

The overall tendency is that the larger the window size, the less the amount of calculations and the less time required.

Comparing $SGXPool$ and $FakeSGXPool$, we can observe that $SGXPool$ takes more time from about $1.644s$ to $1.752s$ than $FakeSGXPool$ when the window size becomes larger. At the same time, the time of $SGXDiv$ and $FakeSGXDiv$ has been reduced to a level that can be ignored. And $SGXDivide$ is also slow when the window

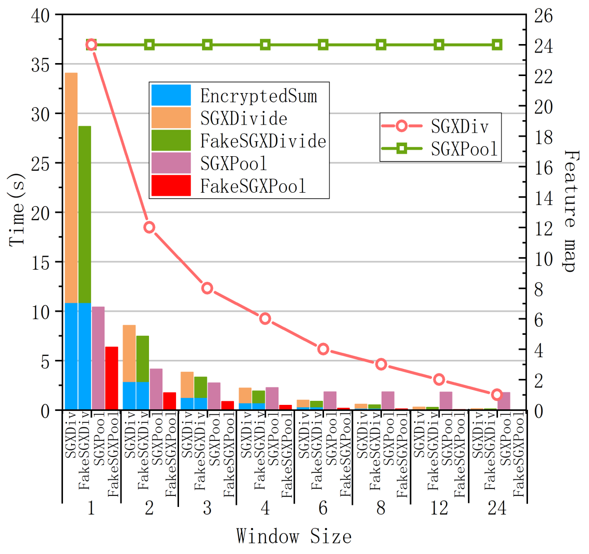


Fig. 6: **Pool computing time with/without SGX.** The line of the right legend indicates the size of the feature map input to SGX. The histogram of the left legend indicates the different computing time of a feature map pooling.

size is small, but it only takes more $4ms$ to $68ms$ than $FakeSGXDivide$ when the window size comparatively large. This is because the number of calculations that need to be divided in SGX has been reduced exponentially after the homomorphic window addition, like the red line trend in Fig. 6. And the degree of this reduction increases with the larger window size. Nevertheless, $SGXPool$ has a fixed input feature map size like the green line that needs to be decrypted to process in SGX. $SGXDivide$ also has many decryption operations when the window size is small. Combined with the extra time-consuming of the decryption in SGX in TABLE IV, we can explain the phenomenon that the large time gap between $SGXDivide$ and $FakeSGXDivide$ when using small window size, and the time of $SGXPool$ does not decrease significantly when using large window size.

The time consumption of $SGXDiv$ is less than the $SGXPool$ after the window size is greater than 3. Consequently, we can choose $SGXPool$ when the window size is less than 3 and select $SGXDiv$ when the window size is larger. Thus we avoid the decryption and decoding time in SGX is even longer than the homomorphic calculation itself. This is a delay trade-off between SGX decryption and direct homomorphic calculation. Therefore, we can combine HE and SGX to achieve the highest pooling layer computing efficiency. Besides, we obviously can only use SGX to perform max-pooling in our scenario.

E. Relinearization

Relinearization can reduce the noise and size of ciphertext after the homomorphic multiplication so that the ciphertext can be decrypted accurately. It is an important technique to extend somewhat HE to fully HE. We count the time of the relinearization, including the key generation and execution. And re-encryption after decryption directly in SGX can avoid calling the relinearization function. We also call $ecall\_DcreaseNoise()$ outside SGX to testify that the time consumption caused by entering and exiting SGX can be ignored in the millisecond.

	Average	STD	96% CI
Reline	65.216	1.472	[64.472,65.928]
SGX	95.55	2.459	[94.335, 96.765]

TABLE V: A relinearization computing time and SGX noise reduction time ($/ms$).

TABLE V denotes that the average time of an SGX noise reduction is about $95.55ms$, the average time of relinearization is about $65.216ms$, and the efficiency of SGX is slower by $30.334ms$. However, we can import the ciphertexts that need noise reduction in a $batchSize$, so one entry and exit of SGX is required only, and the encryption and decryption keys merely need to be loaded once. We get the average time consumption of $23.429ms$ for each SGX noise reduction, which is mainly occupied by encryption and decryption in SGX. Therefore, the effectiveness of the noise reduction method using SGX is satisfactory. TABLE V also shows that the performance of relinearization measured in SGX is less stable than the outside.

VII. Case Study in Edge Computing

The vehicle has become a mobile computing device on wheels and the users in CAVs will generate more complex application requirements, such as association recommendation, image recognition, and interest prediction. Portable smart devices often provide multiple services simultaneously under the constraints of computing resources and energy consumption [22]. Therefore, the neural network inference that expends computing power can be naturally offloaded to the CAVs as the edge computing servers for execution. Nevertheless, it will leak the most sensitive personal information (e.g., images, messages, and browsing history) to service providers and car manufacturers. In the case of using HE to protect the security and privacy of user's data, the ciphertext is transmitted to the edge server. We fed homomorphic encrypted images into our hybrid framework to simulate such a service process using a desktop. The MNIST dataset [23] of handwritten digits from "0" to "9" can be used for training and testing. It has a training set of 60000 samples and a test set of 10000 samples, and the resolution of each image is 28×28 that is represented by its grey level in the range of 0∼255. This will show our framework's feasibility and superiority to provide reliable inference tasks with encrypted information assigned to the edge server, comparing with HE-based inference on CAVs.

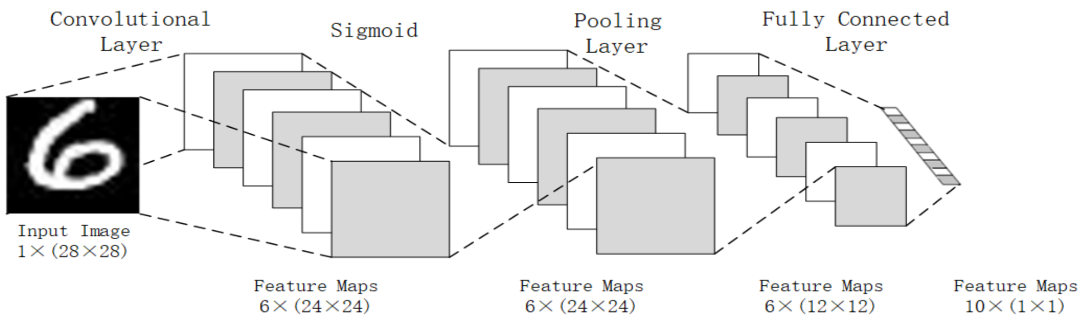


Fig. 7: CNN Model

Input Feature Maps	Layer	Stride	Kernel	Output Feature Maps
$1 \times (28 \times 28)$	Convolutional Layer	(1×1)	$6 \times (5 \times 5)$	$6 \times (24 \times 24)$
$6 \times (24 \times 24)$	Sigmoid	/	/	$6 \times (24 \times 24)$
$6 \times (24 \times 24)$	Pooling Layer	/	$6 \times (2 \times 2)$	$6 \times (12 \times 12)$
$6 \times (12 \times 12)$	Fully Connected Layer	/	$10 \times (12 \times 12)$	$10 \times (1 \times 1)$

TABLE VI: Every layer operation of the CNN model

A. CNN Model

We build the CNN architecture in Fig. 7, which is similar to the network structure in [16]. So we can easily compare it with HE-based CNN inference. This network includes the convolutional layer, pooling layer, activation layer, and fully connected layer at the same time. The following combines the four layers CNN model described in Fig. 7 and TABLE VI specifically. Input a 28×28 MNIST handwritten digital image as an example. There is a detailed analysis of every layer.

1. The first layer is a convolutional layer that has the kernels of size 5×5 and a map count of 6 recorded as $6 \times (5 \times 5)$, each kernel and image performs a convolution operation with a stride of (1×1), generating $6 \times (24 \times 24)$ feature maps.
2. The second layer is an activation layer, using the classic Sigmoid activation function. Input $6 \times (24 \times 24)$, the function mapping output $6 \times (24 \times 24)$ feature maps.
3. The third layer is a pooling layer, the pooling window size is $6 \times (2 \times 2)$, and the mean-pooling method is used to output the average value of the kernel size to generate $6 \times (12 \times 12)$ feature maps.
4. The last layer is a fully connected layer. Input $6 \times (12 \times 12)$, generate kernels $10 \times (12 \times 12)$ that the same size as the input feature map and the same number of output categories automatically. Then, kernels convolve with input feature maps to output 10 categories.

Making use of this elite network, We implement our hybrid inference framework to prove its feasibility and verify that it has significant advantages over a HE-based inference scheme in edge computing.

B. Experiment Results

When the edge server has stored the encoded weight parameters and receives the homomorphically encrypted image request from users and vehicles, the CNN inference service is started. Each $batchSize$ predictions were counted once and repeated 1000 times. All the accuracy rates are consistent with the plaintext predictions, and no case has been found to reduce the accuracy. In Fig. 8, we represent the HE-based CNN inference as $Encrypted$ to do a comparative experiment. Each pixel enters the enclave to calculate as $EncryptSGX(single)$, which causes a delay of $152.5042s$ for each image that did not match our expectation. The $EncryptSGX(single)$ control group will not be considered in our framework because frequent accesses to SGX bring about huge time-consuming. Whereafter, we use feature maps of $batchSize$ as a unit to process together to utilize SGX in our framework that denoted $EncryptSGX$. Putting the same code outside of SGX to compare the time-consumption of using SGX represented $EncryptFakeSGX$.

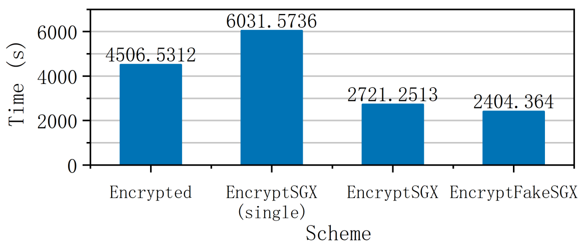


Fig. 8: Prediction time with/without SGX

Our framework implementation $EncryptSGX$ has a prediction time of $272.125s$ per image that saves 39.615% of the time compared to the HE-based scheme $Encrypted$, the time cost of SGX itself is $31.689s$ per image, it can be understood from the above analysis that most of them are caused by the time difference between the encryption and decryption inside and outside SGX. We can integrate our framework into the vehicles as a third-party application for privacy-preserving neural network inference. At the same time, our framework is versatile and can also be used in other edge servers such as roadside units and base stations.

VIII. Discussion

The Chinese Remainder Theorem allows us to perform Single Instruction Multiple Data (SIMD) operations that make it possible to perform batch computing simultaneously, achieving a throughput of thousands of times for different parameters. This operation is supported in the SEAL library directly. In our experiments, we do not utilize the complex coding of SIMD but perform quantitative analysis of the inference operations only, so we should be aware of using SIMD will make our running time shorter, throughput higher, and perform better. Our encryption parameter is $n = 1024$, and if you use SIMD technology, you can get 1024 times the throughput. GPU can also be used to accelerate the homomorphic part in our proposed framework that contains large-number element-by-element modular multiplication [24].

HE is slow relatively, so it is challenging to build different and huge network architecture arbitrarily in experiments to make it compatible with SGX and homomorphic library simultaneously. Therefore, we use a well-designed small network that contains most of the typical operations in four layers to verify its feasibility and superiority. We quantitatively analyze each of these steps to provide the foresight practice of traditional encryption schemes mixed with trusted hardware technology for neural network inference.

IX. Related Work

In recent years, many works have been proposed to protect the security and privacy of neural network applications in different scenarios. The majority of existing methods take advantage of the traditional cryptographic algorithms, such as homomorphic encryption [25], secure multiparty computing [26], differential privacy [27] and Yao's garbled circuit [28], to use mathematical problems to ensure its reliability and confidentiality. Besides, trusted computing technology [29] also can be used to participate in the training and prediction while its hardware attributes ensure confidentiality and privacy.

Graepel et al. [30] proposed to use HE in machine learning algorithms. Aslett et al. [31] show the algorithm for training a machine model on the homomorphically encrypted ciphertext. CryptoNets [16] uses the scaled mean-pool function instead of the max-pool function to avoid division in polling layers, and the square function is adopted to approximate the highly non-linear activation function, while these operations cannot be calculated in HE. Karthik et al. [32] proposed the first approach that attempts training deep neural networks on encrypted data using fully HE in a non-interactive way, and the non-linear function is realized by a piecewise look-up table dexterously to obtain relatively accurate results. Chabanne et al. [10] proposed a novel approach that uses the polynomial approximation of ReLU function with batch normalization and adds a normalized layer before each activation layer to enable a stable and normal distribution at the inputs of the activation function. Intel nGraph [33] takes advantage of the graph compiler toolchain to create a framework for deep learning with HE, but it supported only a limited class of models, restricted to polynomial activations. nGraph-HE2 [34] utilizes a client-aided model to execute deep learning models that contain more types of non-polynomials to maximize throughput. A novel system proposed in [35] utilizes additively homomorphic encryption to protect the gradients over the honest-but-curious cloud server providing deep learning services. The paper [36] utilizes the Taylor theorem to approximate the Sigmoid function in homomorphic-enabled deep learning models. Gazelle [37] is a framework for the secure evaluation of the convolution neural network, which consists of speedy homomorphic implementation and garbled circuits. POCC [38] uses a proxy re-encryption fully homomorphic scheme to encrypt the providers' sensitive data and mixed with Yao's circuit. Taylor and Maclaurin series is used to approximate the non-linear function.

Yerba Buena [29] is an enclave-based model serving system to protect the integrity and confidentiality of user input data. Privado-Converter [39] is a tool that converts Torch [40] to replace data-dependent branches, and it auto-generates a minimal amount of Torch code which runs seamlessly on SGX. Chiron [18] is a system that employs a sandbox called Ryoan [41] on SGX to train the machine learning model on an outsourced service without revealing training data. Ohrimenko et al. [42] proposed data-oblivious machine learning algorithms based on trusted SGX processors.

X. Conclusion

To implement the privacy-preserving neural network inference while avoiding the limitations of the HE and SGX, we propose a hybrid CNN inference framework that combines them to protect the security and privacy of users. In the case of improving accuracy and throughput theoretically, we introduce SGX to accelerate the HE-based CNN inference, eliminating the approximation operations in HE. SGX is also taken as a built-in part to distribute keys instead of an additional trusted third party, thereby improving our framework's scalability and flexibility. In the experiments, we quantified the time-consuming of some basic HE operations in the CNN inference and compared them with corresponding operations in SGX. We also evaluated each layer's performance in the framework under HE and SGX, respectively, and analyzed their characteristics and advantages. Finally, taking the CAVs scenario as a case study in edge computing, we implemented and evaluated our framework using a real CNN model to verify the feasibility and superiority. The proposed framework provides the foresight practice of traditional encryption schemes mixed with trusted hardware technology for privacy-preserving neural network inference.

Acknowledgment

This work is supported by the National Key Research and Development Program of China under Grant 2020YFB1807500, the Key Program of NSFC-Tongyong Union Foundation under Grant U1636209, the Fundamental Research Funds for the Central Universities under Grant XJS210107, XJS210105, XJS211513, and XJS201502, and the Guangdong Basic and Applied Basic Research Foundation under Grant 2020A1515110496 and 2020A1515110079. (*Corresponding author: Qingqi Pei*)

References

[1] M. Katsumata, "A multiple smart device-based personalized learning environment," in *2020 IEEE 10th International Conference on Intelligent Systems (IS)*, 2020, pp. 498–502.

[2] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett, and M. J. Kochenderfer, *Algorithms for Verifying Deep Neural Networks*, 2021.

[3] L. Liu, J. Feng, Q. Pei, C. Chen, Y. Ming, B. Shang, and M. Dong, "Blockchain-enabled secure data sharing scheme in mobile-edge computing: An asynchronous advantage actorccritic learning approach," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2342–2353, 2021.

[4] J. Du, F. R. Yu, G. Lu, J. Wang, J. Jiang, and X. Chu, "Mec-assisted immersive vr video streaming over terahertz wireless networks: A deep reinforcement learning approach," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9517–9529, 2020.

[5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *Internet of Things Journal, IEEE*, vol. 3, no. 5, pp. 637–646, 2016.

[6] Q. Zhang, Y. Wang, X. Zhang, L. Liu, X. Wu, W. Shi, and H. Zhong, "Openvdap: An open vehicular data analytics platform for cavs," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 1310–1320.

[7] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1697–1716, 2019.

[8] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, and W. Shi, "Computing systems for autonomous driving: State-of-the-art and challenges," 2020.

[9] S. Behera and J. R. Prathuri, "Application of homomorphic encryption in machine learning," in *2020 2nd PhD Colloquium on Ethically Driven Innovation and Technology for Society (PhD EDITS)*, 2020, pp. 1–2.

[10] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network." *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 35, 2017.

[11] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: Deep neural networks over encrypted data," *arXiv preprint arXiv:1711.05189*, 2017.

[12] Asylo, "Introducing asylo: an open-source framework for confidential computing." [Online]. Available: http://asylo.dev/

[13] T. Lee, Z. Lin, S. Pushp, C. Li, Y. Liu, Y. Lee, F. Xu, C. Xu, L. Zhang, and J. Song, "Occlumency: Privacy-preserving remote deep-learning inference using sgx," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–17.

[14] F. C.-Y. WANG Juan and C. Yue-Qiang, "Analysis and research on sgx technology," *Journal of Softerware*, pp. 2778–2798, 2018.

[15] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, 2012.

[16] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International Conference on Machine Learning*, 2016, pp. 201–210.

[17] V. Scarlata, S. Johnson, and J. Beaney, *Supporting Third Party Attestation for Intel SGX with Intel Data Center Attestation Primitives*, Intel Corporation, 2018.

[18] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel, "Chiron: Privacy-preserving machine learning as a service," *arXiv preprint arXiv:1803.05961*, 2018.

[19] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen, "Intel® software guard extensions: Epid provisioning and attestation services," *White Paper*, vol. 1, no. 1-10, p. 119, 2016.

[20] Sgx data center attestation primitives. [Online]. Available: https://github.com/intel/SGXDataCenterAttestationPrimitives/

[21] H. Chen, K. Laine, and R. Player, "Simple encrypted arithmetic library-seal v2. 1," in *International Conference on Financial Cryptography and Data Security*. Springer, 2017, pp. 3–18.

[22] J. Feng, F. Richard Yu, Q. Pei, X. Chu, J. Du, and L. Zhu, "Cooperative computation offloading and resource allocation for blockchain-enabled mobile-edge computing: A deep reinforcement learning approach," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6214–6228, 2020.

[23] Mnist database. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[24] T. Morshed, M. M. A. Aziz, and N. Mohammed, "Cpu and gpu accelerated fully homomorphic encryption," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2020, pp. 142–153.

[25] S. Obla, X. Gong, A. Aloufi, P. Hu, and D. Takabi, "Effective activation functions for homomorphic evaluation of deep neural networks," *IEEE Access*, vol. 8, pp. 153 098–153 112, 2020.

[26] S. Sayyad, "Privacy preserving deep learning using secure multiparty computation," in *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*, 2020, pp. 139–142.

[27] Y. Wang, M. Gu, J. Ma, and Q. Jin, "Dnn-dp: Differential privacy enabled deep neural network learning framework for sensitive crowdsourcing data," *IEEE Transactions on Computational Social Systems*, vol. 7, no. 1, pp. 215–224, 2020.

[28] B. D. Rouhani, M. S. Riazi, and F. Koushanfar, "Deepsecure: Scalable provably-secure deep learning," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[29] Z. Gu, H. Huang, J. Zhang, D. Su, H. Jamjoom, A. Lamba, D. Pendarakis, and I. Molloy, "Yerba buena: Securing deep learning inference data via enclave-based ternary model partitioning," *arXiv preprint arXiv:1807.00969*, 2018.

[30] T. Graepel, K. Lauter, and M. Naehrig, "Ml confidential: Machine learning on encrypted data," in *International Conference on Information Security and Cryptology*. Springer, 2012, pp. 1–21.

[31] L. J. Aslett, P. M. Esperança, and C. C. Holmes, "A review of homomorphic encryption and software tools for encrypted statistical machine learning," *arXiv preprint arXiv:1508.06574*, 2015.

[32] K. Nandakumar, N. Ratha, S. Pankanti, and S. Halevi, "Towards deep neural network training on encrypted data," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2019.

[33] S. Cyphers, A. K. Bansal, A. Bhiwandiwalla, J. Bobba, M. Brookhart, A. Chakraborty, W. Constable, C. Convey, L. Cook, O. Kanawi *et al.*, "Intel ngraph: An intermediate representation, compiler, and executor for deep learning," *arXiv preprint arXiv:1801.08058*, 2018.

[34] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "ngraph-he2: A high-throughput framework for neural network inference on encrypted data," in *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2019, pp. 45–56.

[35] Y. Aono, T. Hayashi, L. Wang, S. Moriai *et al.*, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2017.

[36] Q. Zhang, L. T. Yang, and Z. Chen, "Privacy preserving deep computation model on cloud for big data feature learning," *IEEE Transactions on Computers*, vol. 65, no. 5, pp. 1351–1362, 2015.

[37] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "{GAZELLE}: A low latency framework for secure neural network inference," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1651–1669.

[38] P. Li, J. Li, Z. Huang, C.-Z. Gao, W.-B. Chen, and K. Chen, "Privacy-preserving outsourced classification in cloud computing," *Cluster Computing*, vol. 21, no. 1, pp. 277–286, 2018.

[39] K. Grover, S. Tople, S. Shinde, R. Bhagwan, and R. Ramjee, "Privado: Practical and secure dnn inference with enclaves," *arXiv preprint arXiv:1810.00602*, 2018.

[40] Torch. [Online]. Available: http://github.com/torch

[41] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel, "Ryoan: A distributed sandbox for untrusted computation on secret data," *ACM Transactions on Computer Systems (TOCS)*, vol. 35, no. 4, pp. 1–32, 2018.

[42] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, 2016, pp. 619–636.