國立臺灣大學電機資訊學院資訊工程所

碩士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

最佳化以全同態加密作為推論隱私保護機制之效能

Optimizing Privacy-Preserving Inference with Fully Homomorphic Encryption

簡辰哲

Chen-Che Chien

指導教授: 洪士灝 博士

Advisor: Shih-Hao Hung Ph.D.

中華民國 111 年 10 月

October, 2022

國立臺灣大學碩士學位論文
口試委員會審定書

最佳化以全同態加密作為推論隱私保護機制之效能

Optimizing Privacy-Preserving Inference with Fully Homomorphic Encryption

本論文係簡辰哲君（學號R09922146）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 111 年 8 月 31 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

洪士灝
（指導教授）

郭大維　　施吉昇

逄愛君

系 主 任　洪士灝

Acknowledgements

在此特別感謝指導教授洪士灝老師，老師非常重視學生的疑問，總是給予專業的解答與方向，在實驗室設備方面，老師提供完善且足夠的環境供學生進行實驗，讓我在老師的指導下受益匪淺，並且完成此論文。

特別感謝凌華科技股份有限公司提供獎學金，讓我不需要花多餘的心力在經濟上，因此可以專心致志的進行學術研究。

感謝我的家人，支持我讀研究所，在我有困難時給予支持與關心，陪伴我度過重重的困難與挫折，並且順利完成學業。

感謝實驗室同學以及所有認識的系所同學，兩年以來在課業與研究上的討論，互相砥礪教學相長，日常生活中有了你們才不至於無聊，期望大家畢業後都能有好發展。

摘要

機器學習已經在許多領域發展出了實際應用，而這樣的廣大商機促使許多公司提供機器學習即服務（MLaaS），而這需要用戶將他們的數據上傳到雲端才能使用。然而，一些數據被認為是私人且敏感的，例如醫療和財務記錄，使用這些數據需要承擔相關的法律責任。為了解決這個隱私安全問題，之前的研究採用了使用同態加密技術來進行隱私保護推論，由數據所有者上傳加密過的資訊，讓模型所有者在無法得知確切資訊的前提下進行推論。如此雖然能夠有效防止資訊洩露，但也衍生出額外、沉重的代價，尤其是準確率下降、計算時間長、記憶體膨脹等三個問題，密切影響機器學習即服務的性價比。

雖然先前已有單獨解決這三個問題的的研究文獻，但其實這三個問題並非完全獨立。因此，本論文提出了一種綜合方法，希望有系統地應用了一系列優化的方法共同解決這些問題。除了前人提出的方法之外，我們引入了兩種融合方法: 非線性融合 *(non-linear fusion)* 和 常數融合 *(constant fusion)*，與現有的 線性融合 *(linear fusion)* 技術一起解決計算和記憶體容量問題。

實驗結果顯示，使用我們的方法，CIFAR-10 資料集的隱私保護推理的準確度提升 2.42%、時間減少 86% 、記憶體減少 78%。希望本論文所提出的方法能為未來更完整的自動化優化方案奠定基礎。

關鍵字：密碼學、全同態加密、隱私保護、神經網路、人工智慧

Abstract

Machine learning has enabled practical applications in many fields, leading many companies to offer Machine Learning as a Service (MLaaS). However, it requires users to upload their data to the service provider, including private and sensitive data, such as medical and financial records, which may be associated with legal responsibilities. In order to solve such privacy and legal concerns, previous studies have employed homomorphic encryption (HE) for privacy-preserving inferences, where the data owner uploads encrypted data and the model owner makes inferences without knowing the exact information.

While adopting HE is capable of preventing information leakage, it also incurs substantial overheads, especially in terms of dropping accuracy, lengthy computing time and inflated memory, which seriously affect the cost-performance of MLaaS. While there are works to address these three issues individually, they are not entirely independent. Therefore, in this thesis, we propose an integrated approach which systematically applies a series of methods for optimizing HE-based privacy-preserving inference to mitigate the issues jointly. In addition to the methods proposed by previous works, we introduce two fusion methods: *non-linear fusion* and *constant fusion*, which are combined with the existing *linear fusion* techniques to solve computational and memory problems simultaneously.

Experimental results show that our approach provides a +2.42% accuracy boost, an 86% time decrease, and a 78% RAM reduction for privacy-preserving inference on the CIFAR-10 dataset. Hopefully, this thesis paves the groundwork for more complete automated optimization schemes in the future.

Keywords: Cryptography, Fully homomorphic encryption, Privacy-preserving, Neural networks, Artificial Intelligence

Contents

List of Figures

List of Tables

Chapter 1　Introduction

Machine learning has flourished in the past decade and produced many useful models. Instead of deploying the models to the users, the models are often kept in the cloud service to avoid direct access from the users, as a way to protect the models as intellectual properties. However, this means the users have to send the data to the cloud service providers, which causes security and privacy concerns. In order to resolve these security and privacy concerns, methods such as encrypted computation [16, 19, 36] and hardware enclaves [27] can be applied with additional performance overheads and costs.

In this thesis, we focus on privacy-preserving inference based on **Homomorphic Encryption** (HE). The concept of HE has been proposed to perform encrypted computation since 1978 [30]. Since then, algorithms for realizing HE have been developed [5, 10, 11, 15–17, 33]. Meanwhile, HE has been proposed to carry out privacy-preserving inferences by Graepel et al. [20]. The data owner encrypts the data using HE and then outsources it to the cloud for inferences. The data remains in the encrypted state throughout the inference process, thereby achieving zero trust. Nonetheless, due to the high computational cost associated with HE, such methods have not been widely adopted in practice until some recent breakthroughs in algorithm design [6, 8, 12, 18, 21], compiler techniques [3, 4, 13, 14], and hardware acceleration [1, 29, 31, 32]. Still, to further encourage the adoption of HE for privacy-preserving inference, there are three major issues:

1. **Dropping Accuracy.** A critical handicap of using HE is that it cannot be applied to some non-linear layers directly. Popular schemes such as ReLU and Max Pooling have to be approximated. Previous work [12] shows a sharp drop in accuracy, as excerpted in Table 1.1.

Activation	CIFAR-10 Train Acc.	CIFAR-10 Test Acc.
ReLU	93.10	86.76
ReLU-Approx	77.95	75.99
Swish	91.55	86.24
Swish-Approx	77.30	75.41

Table 1.1: Approximation Results

2. **High Computational Cost.** HE adds excessively high computational overheads. As shown in Table 1.2, it took CryptoNets [18] 570 seconds to complete an encrypted inference on one CIFAR-10 image, which is about 10,000× over unencrypted computation on the same CPU. Thus, hardware acceleration [1, 29, 31, 32] and software optimization [4, 6, 9, 12] are keys to reducing the overhead, and the effects are clearly shown in the table.

3. **Inflated Memory Usage.** It consumes at least 100 GB of memory to evaluate standard convolutional networks with CryptoNets' encoding scheme on tiny CIFAR-10 images (32×32×3 pixels). Thus, memory requirement for larger images, e.g., 789 ~1183 GB for mid-range (224×224×3 pixels) resolution medical images [12], would exceed the memory capacity of today's computers.

Although interactive methods [26, 28] may reduce the computational requirements

Publication	Year	MNIST	CIFAR-10	Hardware
CryptoNets [18]	2016	570 sec	-	CPU
Faster CryptoNets [12]	2018	39.1 sec	22,372 sec	CPU
HCNN [1]	2020	5.16 sec	304.43 sec	GPU
F1 [31]	2021	0.17 ms	241 ms	FPGA
CraterLake [32]	2022	0.14 ms	50.50 ms	FPGA

Table 1.2: Computational Cost

on the server side and carry out non-linear layers on the client side to improve the accuracy, they demand a fast, stable network connection to provide good service. While this paper focuses on non-interactive privacy-preserving inferences, some of the proposed optimization methods are still applicable to interactive methods. While there are works to address these three issues individually, they are not entirely independent. For example, one optimization method may reduce the computational cost with increased memory usage. Large models could potentially improve the accuracy, but would significantly increase the computing/memory overheads, which is why we aims to provide an integrated approach to mitigate these inter-related issues. Therefore, in this thesis, we propose an integrated approach which systematically applies a series of methods for optimizing HE-based privacy-preserving inference to mitigate the issues jointly.

As a proof of concept, we include several existing techniques and introduce new techniques in this work and illustrate how to make trade-offs. Hopefully, this approach paves the groundwork for more complete automated optimization schemes in the future. The contributions of this thesis are listed below:

- This thesis integrates three methods, which are separately proposed in the literature

[8, 23], to address the dropping accuracy issue.

- Two fusion methods, *non-linear fusion* and *constant fusion*, are introduced in this thesis to address the computational and memory capacity issues together with the existing *linear fusion* technique.
- Experimental results show that our approach provides a +2.42% accuracy boost, an 86% time decrease, and a 78% RAM reduction for privacy-preserving inference on the CIFAR-10 dataset.

The remaining of this thesis are organized as the following. Chapter 2 explains how HE functions and surveys how HE combined with neural networks performs privacy-preserving inferences as well as the practical issues. Chapter 3 describes our methodology to address the practical issues systematically with an integrated approach to mitigate the problems introduced by HE. The proposed methodology is further evaluated with experiments in Chapter 4. Chapter 5 concludes the findings of this thesis and points out possible future research directions.

Chapter 2 Background

In Chapter 2.1, we explain how homomorphic encryption (HE) works. Subsequently, we survey how HE combined with neural networks performs privacy-preserving inferences as well as the practical issues in Chapter 2.2.

2.1 Homomorphic Encryption

The advent of HE allows a computer to perform certain operations on data in an encrypted state without accessing the secret key, and only with the secret key can the outcome be decrypted. The decrypted outcome is the same as the output of the equivalent computation performed on the unencrypted data. In the field of HE, there are a variety of encryption schemes developed, which can be divided into 3 categories:

- **Partially Homomorphic Encryption:** PHE allows addition or multiplication with unlimited depth. That is, one can perform either an unlimited number of additions or an unlimited number of multiplications on the ciphertext.

- **Somewhat Homomorphic Encryption:** SHE supports both addition and multiplication with limited depth. Namely, both addition and multiplication can be applied a limited number of times to the ciphertext.

- **Fully Homomorphic Encryption:** Being developed from SHE, FHE also supports addition and multiplication. The difference is that FHE supports unlimited depth. In other words, both addition and multiplication can be performed an unlimited number of times on the ciphertext.

Rivest et al. [30] came up with the idea of HE in 1978, but not until 2009 did Gentry propose the first plausible encryption scheme [16] to realize FHE. Gentry's scheme starts by constructing a SHE cryptosystem and then uses bootstrapping techniques to transform it into a FHE cryptosystem. All schemes developed afterwards follow this blueprint of Gentry's construction. Over a decade, many FHE schemes [5, 10, 11, 15, 17, 33] have been proposed to improve the functionality and performance, bridging the theoretical and practical usage gaps.

Current FHE schemes, based on lattice-based cryptography, use some noise to mask plaintext messages, and with the secret key, the noise can be removed completely during the decryption procedure. The noise grows within the ciphertext as each addition and multiplication is performed. Once the accumulated noise exceeds a given budget, depending on the encryption parameters, HE decryption fails. The bootstrapping operation produces a substitute ciphertext with lower noise, but it is extremely computationally expensive.

In abstract algebra, a homomorphism is defined as a structure-preserving mapping between two algebraic structures of the same type, e.g., two groups or two rings. For example, if $f : A \to B$ is a map between two sets A, B of the same structure, and if $\cdot$ is a binary operation of the structure, then $f(x \cdot y) = f(x) \cdot f(y)$ for every pair x, y in A. HE is capable of preserving two operations, multiplication and addition, between the plaintext ring and the ciphertext ring through encryption and decryption functions. It is worth noting that HE does not support division. Figure 2.1 depicts the computation flow of HE. x1 and x2 denote the plaintexts. Enc, Dec, pk, and sk represent the encryption and decryption functions, as well as the public and private keys. $\otimes$ is the corresponding ring operation to $\times$. The unencrypted computation follows the black arrow, while the encrypted computation is along the red arrows.

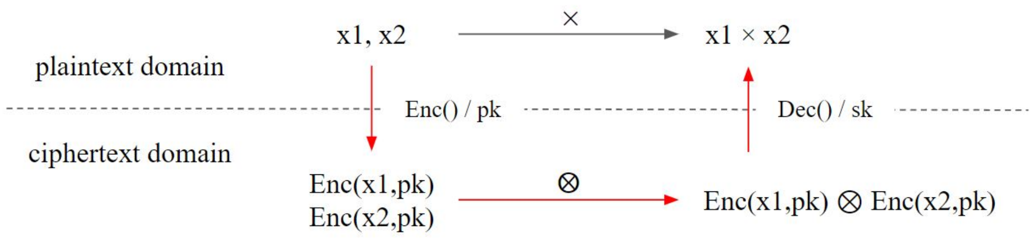


Figure 2.1: A High-level View of HE Computation Flow

2.2 Privacy-Preserving Inferences

Privacy-preserving inferences aim to address the privacy concerns between the data providers and the model providers. In order to preserve privacy between the two parties, two conditions must be met:

(i) The model owner learns nothing about the data provided by the data owner.

(ii) The data owner learns nothing from the model owner.

The fact that HE permits computations to be performed on encrypted data directly makes it ideal to handle both of the above conditions when it comes to issues of intellectual property. As shown in Figure 2.2, HE can be used to realize privacy-preserving inferences, where the data owners send encrypted data to the cloud services over the Internet, so the model owner cannot know the information contained in the data, which satisfies Condition (i). From the perspective of the model owners, only the final outcome and some intermediate results are revealed to the data owners. While malicious users may try to extract model weights through model inversion attacks, it would be very challenging as millions of requests would be needed, which requires significantly more efforts due to the overhead of encryption and decryption on the user side. Hence, Condition (ii) is addressed.

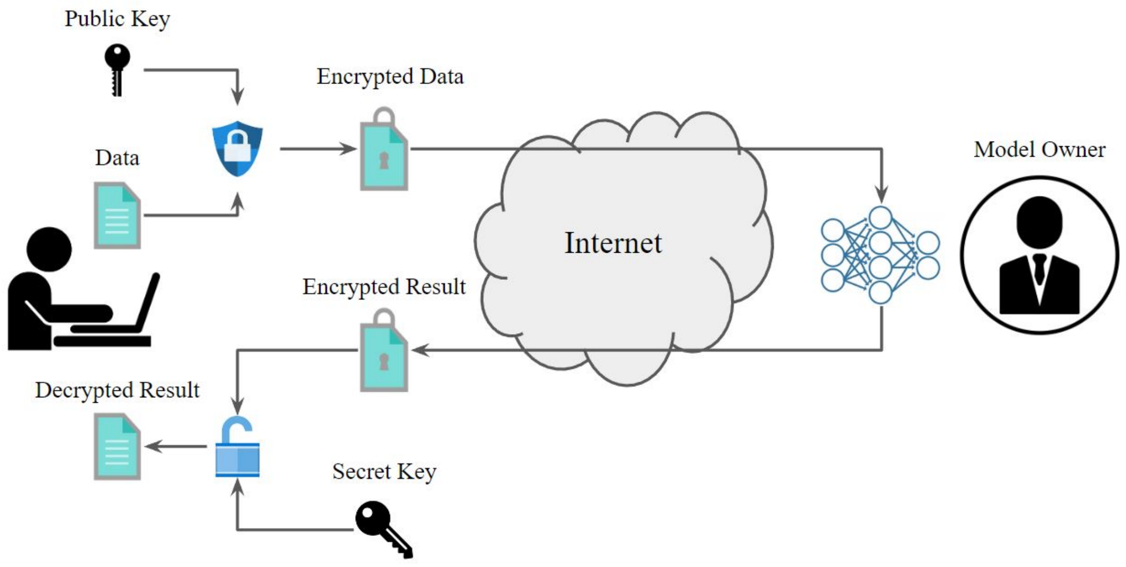


Figure 2.2: Privacy-Preserving Inference Procedure

In practice, the procedure can be implemented with an interactive scheme or a non-Interactive scheme. The advantages and disadvantages are discussed in Chapter 2.2.1. Since today's HE schemes can only perform addition and multiplication operations on encrypted data, so the pooling layers and the activation functions in existing neural networks have to be approximated with a series of addition and multiplication, especially for the case of non-interactive inferences, which is further discussed in Chapter 2.2.2. Finally, HE introduces excessively high computational and memory overheads, which are discussed in

Chapter 2.2.3. Note that, due to the computing/memory overhead, HE-based models tend to be small with limited accuracy, which is why we aims to provide an integrated approach to mitigate these inter-related issues.

2.2.1 Interactive versus Non-Interactive

Privacy-preserving inferences based on HE can be divided into two categories, interactive and non-interactive, which are further elaborated below:

- **Interactive Privacy-Preserving Inferences.** Interactive privacy-preserving Inferences allow the data owner to interact with the compute node (the model owner) during the inferences. Since the intermediate results can be passed back to the data owner to reduce the noise via re-encryption, the computational complexity caused by deep multiplication levels can be reduced. Also, the data owner can help the compute node in performing functions that cannot be done exactly with HE, which would have to be approximated with reduced accuracy. However, it requires communication between the data owner and model owner, which not only adds latency and communication costs but also increases the risk of data leakage.

- **Non-Interactive Privacy-Preserving Inferences.** Non-interactive privacy-preserving inferences forbid the data owner to interact with the compute node (the model owner) during the inferences. Thus, the model owner reveals less knowledge to the data owner and the risk of data leakage from eavesdroppers is reduced. However, non-interactive schemes have to deal with the following issues: (1) Since certain functions (e.g., ReLU) cannot be computed exactly with HE, approximation methods would reduce the accuracy of inference. (2) The multiplication levels from deep neural networks would accumulate noise and cause problems such as corrupted results and high computational costs.

In comparison, interactive privacy-preserving inference methods would reduce the computational requirements on the server side but demand a fast, stable network connection to provide good service. On the contrary, non-interactive privacy-preserving inference methods depend on the servers to perform all the required operations on encrypted data, which are more challenging in terms of the design and optimization of the neural network. While this paper focuses on non-interactive privacy-preserving inferences, some of the proposed optimization methods are still applicable to interactive methods.

2.2.2 Accuracy Issues

For the case of non-interactive inferences, popularly used activation/pool functions need to be approximated with polynomial functions so they can be manipulated with HE add/multiply operations:

- **Max Pooling** is an operation that selects the maximum value for sub-regions of an input feature map. The purpose of applying max pooling is to create a downsampled feature map. **Avg Pooling** is a HE-friendly alternative to Max Pooling, which averages over sub-regions instead of taking the maximum as shown in Figure 2.3. As mentioned above, HE does not provide division. The sum of the sub-region is multiplied by $\frac{1}{\text{pool size}}$ instead of being divided by pool size while doing average.
- **Activation functions** commonly applied in neural networks are usually not polynomial, which cannot be expressed as a combination of additions and multiplications. For example, Rectified Linear Unit (ReLU), the most widely used activation

function, contains conditions for comparison that cannot be computed in the homomorphic domain. However, replacing activations with polynomials, as shown in Figure 2.4 causes a drop in accuracy, and therefore some studies [8, 12, 21, 22] aim to find a good approximation to retain as much accuracy as possible. To make a good approximation, a high degree polynomial is desired. On the other hand, the multiplication depth of the model increases, which introduces huge overhead. To balance the two, most works [1, 6, 18, 22] employ a degree two polynomial.

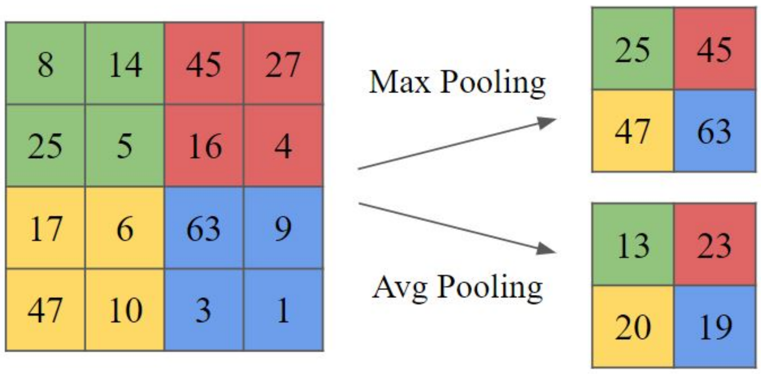


Figure 2.3: Pooling with Pool Size 2×2

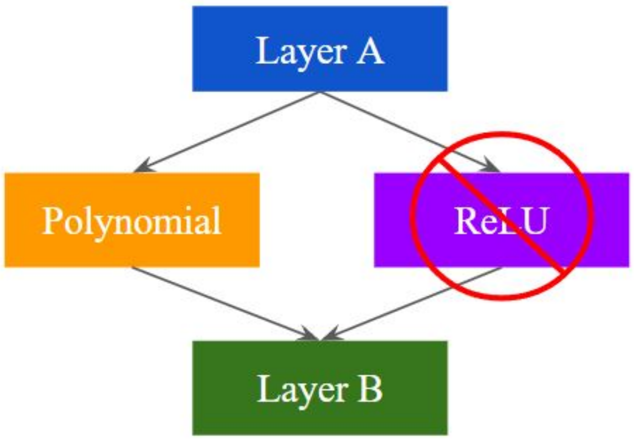


Figure 2.4: Substituting ReLUs with polynomials

2.2.3 Computing/Memory Overheads

A naive approach encodes each pixel of the input image separately and requires a huge memory and operations to store and process the pixel-wise ciphertext data for the input image and the feature maps for each layer.

With CryptoNets [18], it consumes at least 100 GB of memory to evaluate standard convolutional networks on tiny CIFAR-10 images (32×32×3 pixels) using the naive pixel-wise encoding approach. For a mid-range resolution image, e.g., 224×224×3 pixels, the same approach demands 789 ~1183 GB [12] of memory capacity, which would not fit in many of today's computers.

As shown in Table 1.2, it took CryptoNets 570 seconds to finish an encrypted inference on one CIFAR-10 image with HE, which is about 10,000× over unencrypted computation on the same CPU. Thus, hardware acceleration [1, 29, 31, 32] and software optimization [4, 6, 9, 12] are keys to reducing the overhead, and the effects are clearly shown in the table.

To reduce the memory and computing overheads, Lola [6] proposed an encoding approach that encodes across many pixels of the input and the feature maps. However, as it cannot be used to perform inference on a batch of input images, this approach works only for single-image inference. In this thesis, we mainly focus on reducing the multiplication level and possibly the number of operations with layer/constant fusing techniques. While the fusing techniques work for batch inference, it can be combined with Lola to deliver the best results.

Chapter 3 Methodology

This chapter describes our methodology to address the practical issues systematically with an integrated approach to mitigate the problems introduced by HE. While there are works to address these three issues individually, they are not entirely independent. For example, one optimization method may reduce the computational cost with increased memory usage. Therefore, in this thesis, we propose an integrated approach which systematically applies a series of methods for optimizing HE-based privacy-preserving inference to mitigate the issues jointly.

First, we target the accuracy issue introduced by approximation. As mentioned in Chapter 2.2, when a neural network is applied with homomorphic encryption, certain layers must be substituted with HE-friendly layers, which would harm the accuracy of the model. Also, due to the computing/memory overhead, HE-based models tend to be small, which limits their accuracy, too. Chapter 3.1 introduces three methods to address such accuracy issues. Secondly, to reduce the computing and memory overheads caused by HE, Chapter 3.2 provides two layer fusion methods and two constant fusion methods. While the proposed approach can integrate many existing techniques, as a proof of concept, we include several existing techniques and introduce new techniques in this thesis to illustrate how to make trade-offs with the proposed methodology.

3.1 Mitigating for Accuracy

This section dissects three main factors which cause the drop in accuracy of a model and the corresponding three methods to mitigate the accuracy drop. The three main factors are:

1. **Polynomial Approximation.** Using polynomials to approximate non-polynomial activation/pooling functions is inevitable. A good approximation preserves accuracy. Conversely, a bad approximation results in poor accuracy. For example, Figure 3.1 compares ReLU to its degree 2 polynomial approximation, and the errors vary greatly with the inputs.

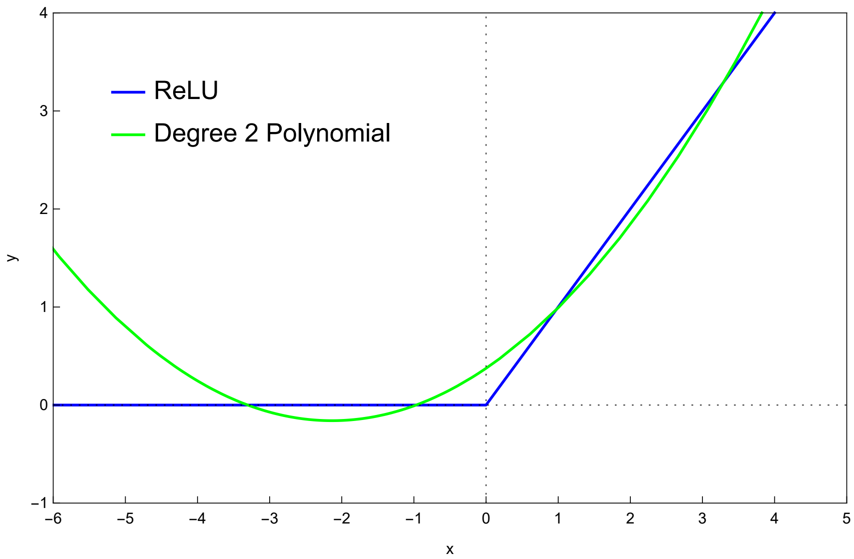


Figure 3.1: Approximation of ReLU by a Degree 2 Polynomial

2. **Training for Approximation.** Previous work [12] shows that a sharp drop in accuracy happens after substituting all the activations with polynomials at once and re-training the neural network from scratch (see Table 1.1). CIFAR-10 training accuracy drops from 93.10% to 77.95% and testing accuracy drops from 86.76% to

75.99% with ReLU-A, giving over 15% lower training accuracy and over 10% lower testing accuracy. Is there a smarter way to train the neural network in order to close the accuracy gap?

3. **Model Compression.** Due to computing/memory constraints, existing privacy-preserving inference schemes are usually done with small models, which are not capable of capturing all the information in the training data and thus result in poor accuracy. While this is a practical limitation for inferences, it should not affect the training process. One could train a large model to retain the knowledge and compress the model to reduce the resource consumption for HE-based inferences. However, traditional neural network compression techniques may require modifications since the resource consumption and accuracy are affected by HE.

3.1.1 Improving Polynomial Approximation

As shown in Figure 3.1, a polynomial function usually does not fit an activation perfectly, and there is a huge variance of approximation error at different points, e.g., a big error at x = -6 and a small error at x = -1. Thus, one can make a polynomial approximation behave as close to an activation function as possible by limiting the range of the inputs. Without limiting the range of the inputs, they are dispersed across the real number space, and an arbitrary interval is used to approximate the activation. Some input values will occur where the error between the polynomial and the activation is immense. Meanwhile, by limiting the range of the inputs, one can approximate the activation within that range, and the input values will occur at the point where the approximation is more accurate.

Chabanne et al. [8] propose a method that places a batch normalization layer (BN)

[24] before every polynomial activation layer (POLY). The BN makes the inputs to the POLY as close to a normal distribution as possible, giving the input a mean of zero and a variance of one. One can limit most (99.7%) of the inputs within [-3,3] in theory and use a polynomial to approximate the activation in [-3,3] or a little wider interval. In such a situation, most the input values will occur at the point where the approximation error is tiny, and therefore, the model accuracy improves.

3.1.2 Training for Approximation

To find a smarter way to substitute activation functions with approximated polynomial functions during the training process, we apply the technique proposed in AdaStart, proposed by my labmate, Chuan-Chi Wang, which is a training algorithm that transforms a model into an HE-supported model by iteratively replacing activation layers from the front to the rear of the model and fine-tuning after each replacement. In the first iteration, AdaStart begins with the training of the original model with non-polynomial activations. For each subsequent iteration, an activation is replaced with the polynomial, resulting in a changed model. The changed model inherits the weights, which are better initial values than random numbers, from the previous iteration and is then fine-tuned. An example is showed in Figure 3.2. The experimental results indicate that AdaStart outperforms retraining from scratch.

3.1.3 Model Compression

Model compression techniques such as pruning and knowledge distillation are ideal for training a privacy-preserving neural network since they aim to reduce the size of large

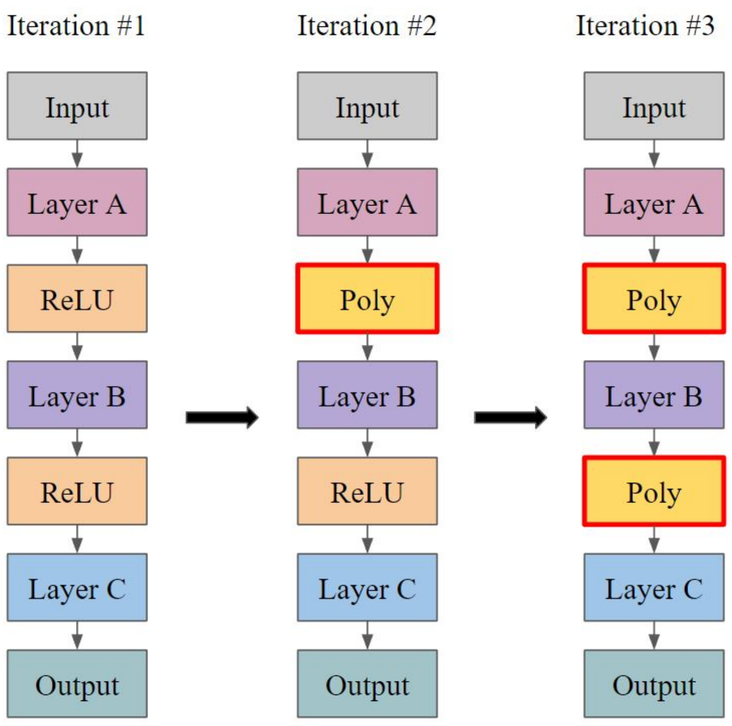


Figure 3.2: An Example of Training with AdaStart

models without sacrificing too much on accuracy. In other words, they aim to improve the accuracy of small models. The reduction in size means that the compressed model has a smaller number of parameters, contributing to less memory consumption and inference time. Such benefits are desirable especially for HE-based inferences, but the trade-offs are more complicated.

As a case study, we adopt *knowledge distillation* (KD) to compress the models. KD was first proposed by Bucila et al. [7] and generalized by Hinton et al. [23]. In KD, the small model is referred to as the ”student” model, and the large model, i.e., a deep and wide neural network or an ensemble of many models, is referred to as the ”teacher” model. The pre-trained teacher model transfers its knowledge to the student model. In other words, the student model is trained to imitate the teacher model. The student model learns from the teacher model by optimizing the modified loss function below:

$$\mathcal{L}(x; W) = \alpha \times \mathcal{H}(y, \sigma(z_s; T = 1)) + \beta \times \mathcal{H}(\sigma(z_t; T = \tau), \sigma(z_s; T = \tau)) \tag{3.1}$$

where x is the input, W are the weights of the student model, $\mathcal{H}$ is the cross-entropy function, y is the ground truth label, σ is the softmax function parameterized by the temperature T, z_s and z_t are the logits of the student and teacher model respectively, and α and β are hyperparameters, being the proportions of learning from ground truth and the teacher model respectively. Figure 3.3 visualizes the KD loss function.

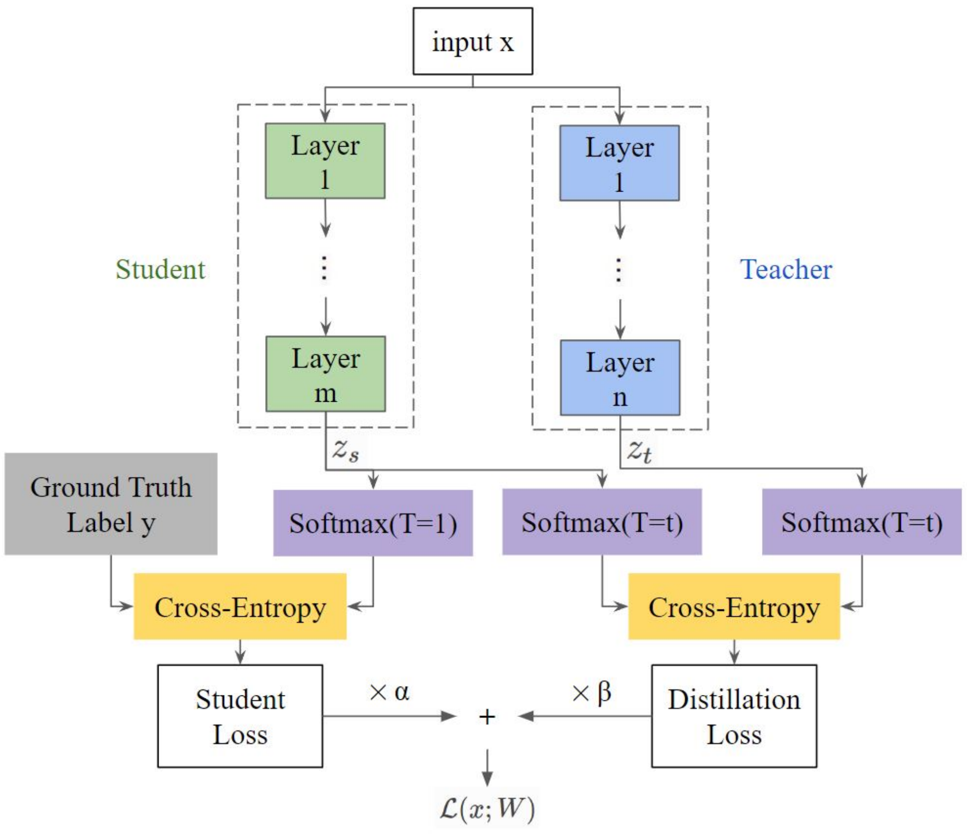


Figure 3.3: Knowledge Distillation Procedure

3.1.4 Summary - Solving Accuracy Issues

Normalizing input of polynomial activations with batch normalization, method #1, is used in the model architecture designing phase. AdaStart, method #2, and Knowledge Distill, method #3, are employed in the training phase. These three methods can be utilized simultaneously.

Batch normalization is capable of smoothing the surface of the loss function; therefore, it makes optimization easier. As far as we know, batch normalization serves as a building block in almost all works. The difference is that one typically does not consider the order of a batch normalization layer and an activation. Chapter 3.2.1, we will discuss the fusion, called non-linear fusion, of a batch normalization layer and a polynomial activation layer, which totally eliminates the computation cost of a batch normalization layer. As a consequence, method #1 is a must-use.

3.2 Mitigating for Computing/Memory Overhead

While homomorphic encryption provides high-level security, it also introduces tremendous overhead in terms of memory usage and computation time, as described in Chapter 2.2.3. The multiplication level refers to the number of multiplications (depth) that data goes through from the model input to the model output. This section illustrates the effects of multiplication level in terms of memory and time. We describe separately below:

1. **Inflated Memory.** Table 3.1 shows the measurement of memory consumption for encrypting a CIFAR-10 image as the input to a neural network with multiplication level **L** using the open-source package TenSEAL [2] and Python memory-profiler.

N is polynomial modulus degree, which means a ciphertext is represented as a degree (N-1) polynomial. As **L** gets larger, **N** must also increase to maintain the same level of security, e.g., **N** increases from 8192 to 16384 as **L** increases from 7 to 8. Reducing **L** can reduce memory usage, and when **L** is reduced to a certain extent, **N** can be lowered, which can reduce memory usage more significantly.

N \ **L**	1	2	3	4	5	6	7	8
8192	2.144	2.888	3.756	5.036	5.712	6.357	7.295	-
16384	3.988	5.475	7.242	10.308	11.164	12.406	14.367	16.953
32768	7.655	10.601	14.092	19.459	22.115	24.608	28.291	33.761

Table 3.1: Memory Usage of Encrypting a CIFAR-10 Image with TenSEAL

2. **Lengthy Computing Time.** Table 3.2 lists the time complexity of RNS-CKKS , the mainstream homomorphic encryption scheme for privacy-preserving inference. **N** is the polynomial modulus degree, and **r** is the level of the ciphertext when the operation takes place. For non-interactive inference, to minimize the execution time, at the beginning, **r** equals the model's multiplication level **L** for receiving an input ciphertext. When the ciphertext undergoes a multiplication, **r** decreases by one and eventually reaches zero for the model outputs. Since the initial value of **r** must be larger or equal to **L**, it is important to reduce **L** for shortening the inference time.

Both Chapter 3.2.1 and Chapter 3.2.2 provides methods to reduce the multiplication level. In Chapter 3.2.1, we propose non-linear fusion and combine linear fusion to merge different layers aggressively. In Chapter 3.2.2, we generalize the constant fusion concept to fuse multiple constants.

Operation	Time Complexity
PT-CT Add	$\mathcal{O}(Nr)$
CT-CT Add	$\mathcal{O}(Nr)$
PT-CT Mul	$\mathcal{O}(Nr)$
CT-CT Mul	$\mathcal{O}(N(\log N)r^2)$
Rotation	$\mathcal{O}(N(\log N)r^2)$

Table 3.2: Time Complexity of RNS-CKKS Operations

3.2.1 Layer Fusion

The multiplication level plays a big role when dealing with overhead. By fusing successive layers, the multiplication level of the model is reduced, and with a reduced multiplication level, we can use smaller encryption parameters settings, leading to less memory usage and inference time. For example, for plaintext inference, *linear fusion* is used to improve data reuse and reduce the inference time for embedded applications []. For HE-based inference, the benefit of fusion is much greater and should be aggressively exploited as it reduces the multiplication level. Furthermore, since certain non-linear pooling/activation layers are approximated by polynomials with HE, we propose a new method called *non-linear fusion* to expand the scope of fusion by including non-linear layers.

- **Linear Fusion.** All linear layers must have their own matrix representation,and they have the general form of computation $Mx + N$, where M is a matrix and N is a vector. By converting successive linear layers to their general computation form, one can easily fuse them with simple matrix multiplication and vector addition (see Figure 3.4).

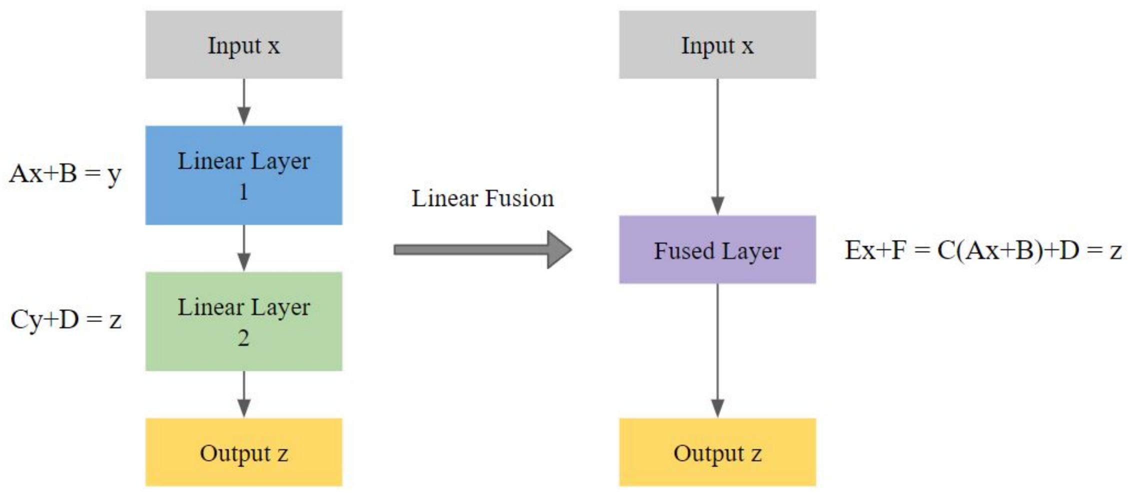


Figure 3.4: Fusing with Matrix Multiplication & Vector Addition

While fusing linear layers reduces multiplication level, it may also changes the number of operations for additions and multiplications:

- Fewer Additions and Fewer Multiplications: The total cost is decreased.
- The Other Cases: Cost estimation is needed to determine the benefit.

Increased operations does not necessarily grow the inference time, since the reduced multiplication level affects the computation time of all layers (see Table 3.2). This trade-off issue is discussed in future work (Chapter 5).

- **Non-linear Fusion.** In Chapter 3.1.1, we mention that a batch normalization layer should be placed before a polynomial activation layer to limit the input range. It would be helpful if these two layers could be fused, since they are connected. The equation of batch normalization is a degree one polynomial.

$$f(x) = ax + b \tag{3.2}$$

The equation of activation is a degree K (≥ 2) polynomial. Suppose K = 2.

$$f(x) = cx^2 + dx + e \tag{3.3}$$

Substituting the equation of batch normalization into the equation of activation will result in another degree 2 polynomial.

$$\begin{aligned} f(x) &= c(ax+b)^2 + d(ax+b) + e \qquad (3.4) \\ &= ca^2x^2 + (2abc + ad)x + (cb^2 + bd + e) \qquad (3.5) \\ &= fx^2 + gx + h \qquad (3.6) \end{aligned}$$

The operations performed by a batch normalization layer are completely hidden! Specifically speaking, the number of operations in the fusion layer and the activation layer is the same as they are both polynomials of the same degree. The multiplication level of the model is reduced by one level, and the total number of operations is also reduced after doing non-linear fusion, which is always beneficial.

3.2.2 Constant Fusion

A series of multiplying with constants can be fused to save the number of operations and the multiplicative level. nGraph-HE [4] proposed *AvgPool folding*, *Activation folding* and *Batch-Norm folding*, and the core concept of these folding methods is *constant fusion*. We think that the concept of constant fusion should not be limited to certain layers and should be generalized to cover all possible cases and included in our optimization workflow. Below, we apply constant fusion to two cases that are not included in nGraph-HE.

1. **Forward Constant Fusion.** In Avg Pooling, multiplying with $\frac{1}{\text{pool size}}$ can be moved to its previous or next layer and fused with other constants. Assume that a degree 2 polynomial activation layer is placed before an Avg Pooling layer and the pool size is 3x3, and we want to move the constant $\frac{1}{\text{pool size}}$ forward to the polynomial activation layer. All we have to do is change the coefficients and the polynomial goes from

$$f(x) = ax^2 + bx + c \tag{3.7}$$

to

$$\begin{aligned} f(x) &= \frac{1}{3 \times 3}(ax^2 + bx + c) && (3.8) \\ &= \frac{a}{9}x^2 + \frac{b}{9}x + \frac{c}{9}. && (3.9) \end{aligned}$$

Figure 3.5 shows the difference after doing forward constant fusion. The input passes through a modified polynomial and skips the averaging of Avg Pooling.

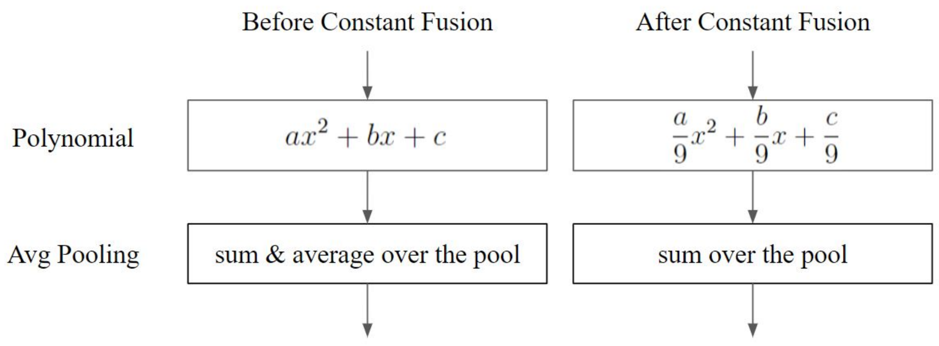
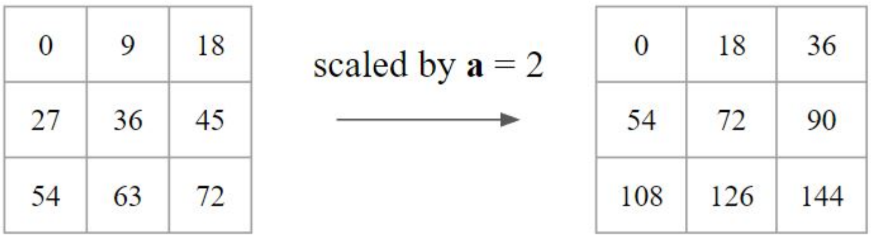


Figure 3.5: Forward Constant Fusion

2. **Backward Constant Fusion** In the polynomial activation layer, multiplying the leading coefficient can be moved to its previous or next layer and fused with other

constants. Assume that a convolution layer is placed after an polynomial activation layer and the filter size are 3x3, and we want to move the leading coefficient a backward to the convolution layer. All we have to do is scale the filters by a (see Figure 3.6).



0	9	18
27	36	45
54	63	72

scaled by **a** = 2

0	18	36
54	72	90
108	126	144

Figure 3.6: A Scaled Filter

Figure 3.7 shows the difference after doing backward constant fusion. The input passes through a modified polynomial and a scaled convolution layer.

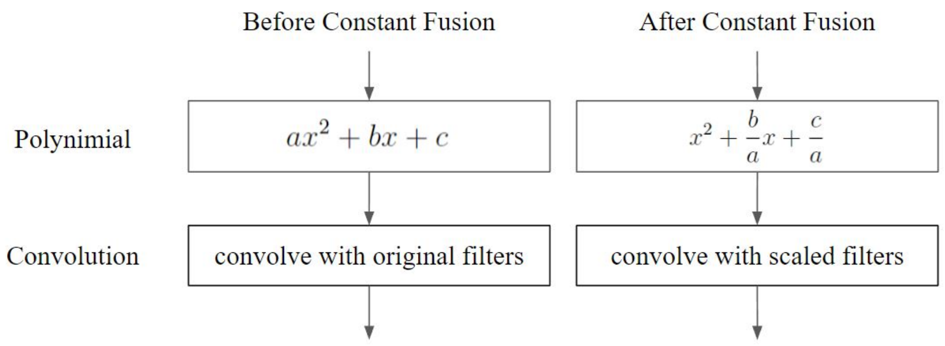

Before Constant Fusion
After Constant Fusion
Polynimial
$ax^2 + bx + c$
$x^2 + \frac{b}{a}x + \frac{c}{a}$
Convolution
convolve with original filters
convolve with scaled filters

Figure 3.7: Backward Constant Fusion

3.2.3 Summary - Solving Computing/Memory Issues

We believe that both layer fusion and constant fusion are critical techniques to reduce the multiplication level. Although it seems both methods can be applied jointly in any order, the order may affect the results in practice. While constant fusion is always beneficial, linear fusion may adversely prolong the execution time as it increases the total number of operations and offsets the benefit of decreased multiplication level. Thus, platform-specific performance estimation/measurement is required to guide the application of fusion methods, which is further illustrated in the case study examples in Chapter 4.

Chapter 4 Empirical Results

In this chapter, we show several experimental results for evaluating the proposed optimization approach. Chapter 4.1 describes the experimental setup in terms of the benchmark dataset, the hardware specifications, and the target neural network models used in the experiments. Chapter 4.2 compares different training methods that help improve accuracy. Chapter 4.3 provides multiple fusing strategies to minimize inference overhead.

4.1 Experimental Setup

We use the standard CIFAR-10 benchmark dataset for our experiments. The CIFAR-10 dataset comprises of 60,000 32x32 color images in 10 classes, with 6,000 images per class, divided into 50,000 training images and 10,000 testing images. The server consists of a high-end AMD 64-core CPU Ryzen Threadripper, an advanced NVIDIA GPU 3080, and 4 $\times$ 64GB RAM. The GPU is used to accelerate the model training processes, and we utilize the CPU for the inference. We define two model structures, *Shallow* and *Deep* (see Figure 4.1 and Figure 4.2). As discussed in section 3.1.1, a batch normalization layer is always put before a polynomial activation. *Shallow* has two polynomial activations and *Deep* has four polynomial activations. The first six layers of *Deep* are identical to the layers of *Shallow*.

CPU	AMD Ryzen Threadripper 3990X, 64C128T, 2.9GHz (base)
GPU	NVIDIA GeForce RTX 3080
RAM	256GB

Table 4.1: Hardware Specifications

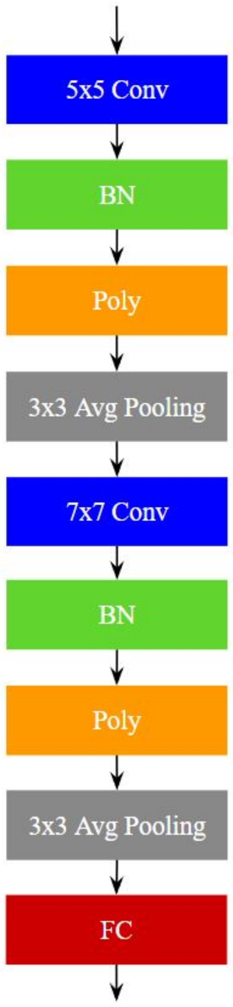


Figure 4.1: Shallow

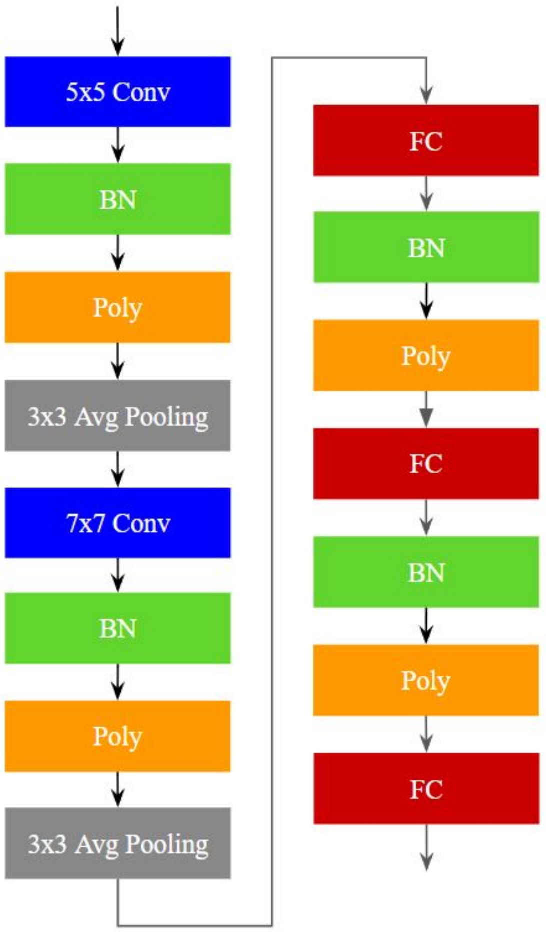


Figure 4.2: Deep

4.2 Training Accuracy

We have described three methods to improve the accuracy of privacy-preserving inference with HE, including normalization before non-linear functions, AdaStart, and knowledge distillation (KD). For the two neural networks used in this experimental study, we have placed a batch normalization (BN) layer before each polynomial activation layer to improve the accuracy. Since the non-linear fusion technique described in Chapter 3.2.1 is capable of fusing a BN layer and a polynomial activation layer, the computation costs of the BN layers are zero. The remainder of this section focuses on evaluating the benefits of AdaStart and KD.

Table 4.2 lists the accuracy acquired by using different training methods with the approximation methods mentioned in Chapter 3.1, including native training, KD, AdaStart, and the combination of AdaStart and KD. The first two rows in the table show the accuracy without approximation, and the rest are with polynomial approximation. It is clear that approximation of ReLU reduces the accuracy, especially for Shallow. For Deep, while it is possible to obtain good accuracy with approximation during the training, the accuracy for the test data set is still impaired by the approximation. The last two rows show that the use of AdaStart has improved the accuracy, with the combination of AdaStart and KD being the best.

In knowledge distillation, we train the student model to imitate the pre-trained teacher model (see Figure 3.3), and the teacher model used here is ResNeXt-29 [35] with 99% training accuracy and 96% testing accuracy. With AdaStart, the original ReLU model is used in the first iteration of training, and the later iterations inherit the weights obtained from the previous iteration (see Figure 3.2). While using AdaStart+KD (last row

	Shallow				***Deep***			
	W = 1		W = 2		W = 1		W = 2	
	Train	Test	Train	Test	Train	Test	Train	Test
Native (ReLU)	82.31	81.71	88.36	84.16	96.21	87.21	98.83	89.14
K.D. (ReLU)	81.05	81.21	89.22	84.42	95.70	87.52	98.75	90.25
Native (Poly)	75.59	75.40	80.74	78.22	88.55	82.97	91.45	83.63
KD (Poly)	74.92	75.10	81.95	78.24	94.45	83.05	96.09	85.02
AdaStart (Poly)	78.67	76.13	82.11	78.80	95.16	**84.16**	96.37	85.56
AdaStart + KD (Poly)	76.45	**76.17**	81.99	**78.91**	95.36	83.90	96.29	**86.05**

Table 4.2: Accuracy Results

of Table 4.2), we let the second iteration inherit the weights obtained by KD (2nd row of Table 4.2) since KD uses a special loss function and we chose to use the same loss function for every iteration. When using KD (4th row of Table 4.2), the second iteration inherit the weights obtained by native training (1st row of Table 4.2) for the same reason.

4.3 Computing and Memory Resources

This section illustrates how to reduce the usage of computing and memory resources on *Shallow* by adopting the mitigating techniques described in Chapter 3.2. We have experimented with the following three strategies:

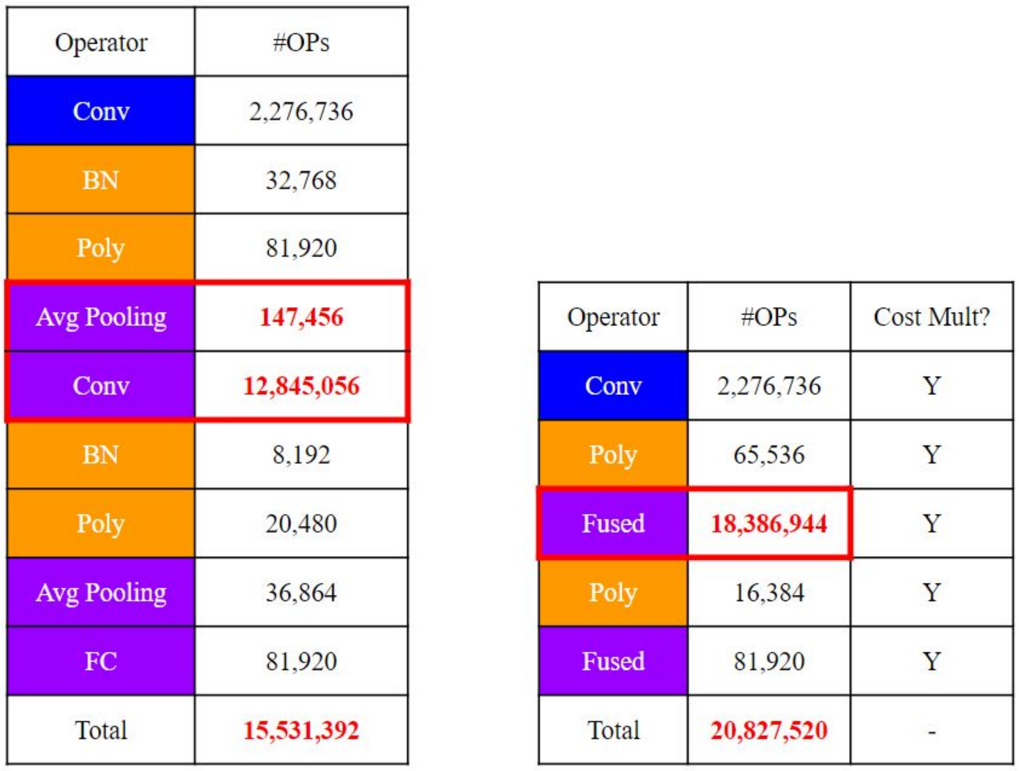
- **Strategy #1:** The constant fusion is applied before layer fusion, and the applied fusion is listed in Table 4.3. The layers of the same color indicate that they are fused (see Figure 4.3). The multiplication level decreases from 11 to 5, saving a total of 6 levels. A tradeoff needs to be considered between decreased multiplication level and increased operations (see red boxes in Figure 4.3).

	Constant Fusion
1	Forward the leading coefficient of the 1st BN layer to the 1st convolution layer.
2	Backward the leading coefficient of the 1st polynomial to the 2nd convolution layer.
3	Forward the leading coefficient of the 2nd BN layer to the 2nd convolution layer.
4	Backward the leading coefficient of the 2nd polynomial to the fully connected layer.
	Layer Fusion
6	non-linear fusion between the 1st BN layer and the 1st polynomial activation
7	linear fusion between the 1st Avg Pooling layer and the 2nd convolution layer
8	non-linear fusion between the 2nd BN layer and the 2nd polynomial activation
9	linear fusion between the 2nd Avg Pooling layer and the fully connected layer

Table 4.3: Fusion Applied in Strategy #1

- **Strategy #2:** Based on Strategy #1, this Strategy does not fuse the 1st Avg Pooling and the 2nd convolution layer, but additionally uses constant fusion to transfer the multiplication of the 1st Avg Pooling to the 2nd convolution layer. The applied fusion is listed in Table 4.4. The multiplication level decreases from 11 to 5, saving a total of 6 levels. The total operations decreased, indicating that there is no trade-off (see Figure 4.4).

- **Strategy #3:** Layer fusion may increase the number of operations. Conversely, layer expansion can also decrease the number of operations. For example, a 5x5 convolution layer with stride equal to 1 can be expanded into two 3x3 convolution layers with stride equal to 1 (see Figure 4.5). However, expansion costs one more multiplication level, and we need to make a trade-off again. In this strategy, the



Operator	#OPs
Conv	2,276,736
BN	32,768
Poly	81,920
Avg Pooling	**147,456**
Conv	**12,845,056**
BN	8,192
Poly	20,480
Avg Pooling	36,864
FC	81,920
Total	**15,531,392**

Operator	#OPs	Cost Mult?
Conv	2,276,736	Y
Poly	65,536	Y
Fused	**18,386,944**	Y
Poly	16,384	Y
Fused	81,920	Y
Total	**20,827,520**	-

Figure 4.3: Comparison of the Number of Operations Before and After Fusion Using Strategy #1

	Constant Fusion
1	Forward the leading coefficient of the 1st BN layer to the 1st convolution layer.
2	Backward the leading coefficient of the 1st polynomial to the 2nd convolution layer.
3	Backward ”1/(pool size)” of the 1st Avg Pooling to the 2nd convolution layer.
4	Forward the leading coefficient of the 2nd BN layer to the 2nd convolution layer.
5	Backward the leading coefficient of the 2nd polynomial to the fully connected layer.
	Layer Fusion
6	non-linear fusion between the 1st BN layer and the 1st polynomial activation
7	non-linear fusion between the 2nd BN layer and the 2nd polynomial activation
8	linear fusion between the 2nd Avg Pooling layer and the fully connected layer

Table 4.4: Fusion Applied in Strategy #2 and #3

Operator	#OPs
Conv	2,276,736
BN	32,768
Poly	81,920
Avg Pooling	147,456
Conv	12,845,056
BN	8,192
Poly	20,480
Avg Pooling	36,864
FC	81,920
Total	15,531,392

Operator	#OPs	Cost Mult?
Conv	2,276,736	Y
Poly	65,536	Y
Avg Pooling	131,072	N
Conv	12,845,056	Y
Poly	16,384	Y
Fused	81,920	Y
Total	15,416,704	-

Figure 4.4: Comparison of the Number of Operations Before and After Fusion Using Strategy #2

2nd convolution layer is expanded into 2 convolution layers, and ~4.5M operations are saved in exchange with one more multiplication. The applied fusion is listed in Table 4.4. The multiplication level decreases from 11 to 6, saving a total of 5 levels (see Figure 4.6).

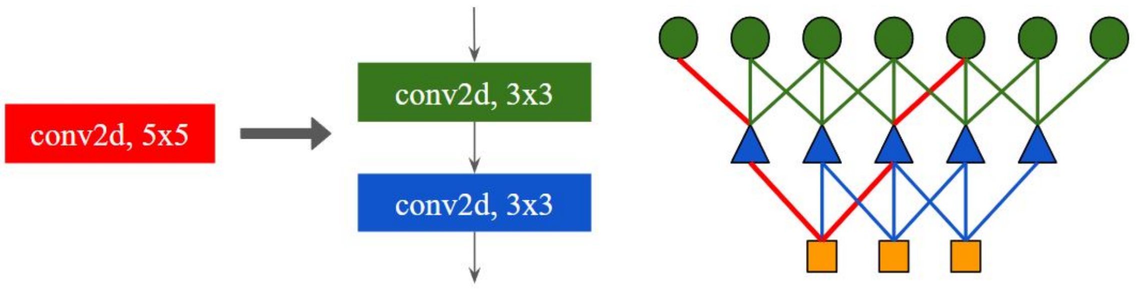
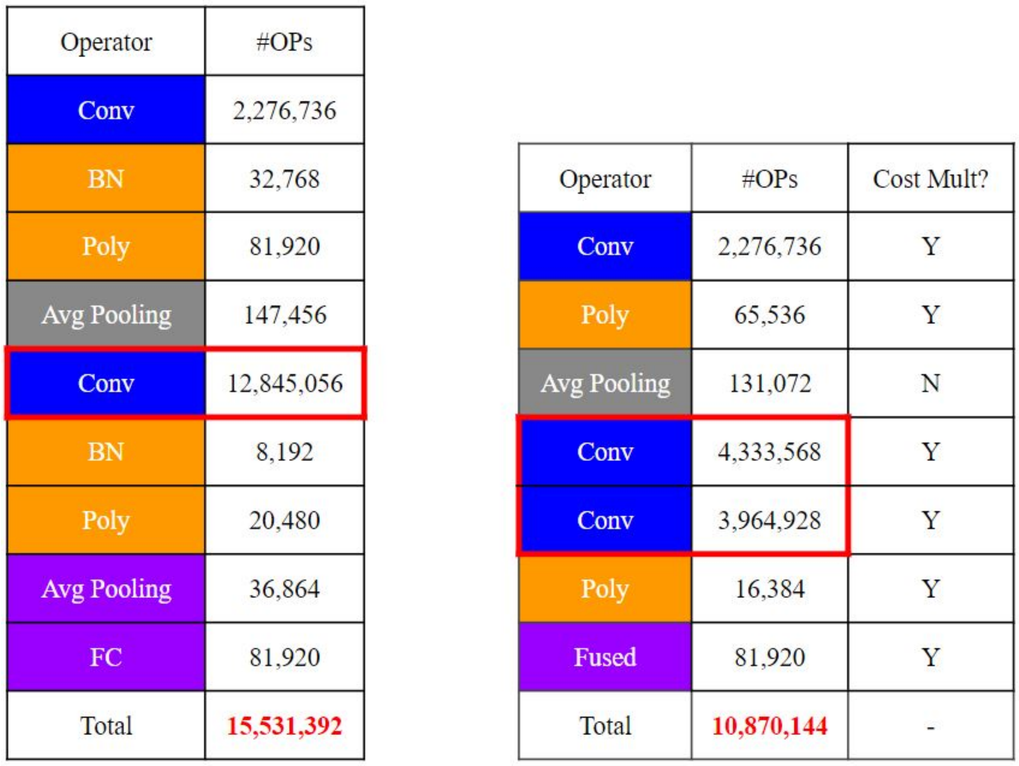


Figure 4.5: Expanding Convolution Layers



Operator	#OPs
Conv	2,276,736
BN	32,768
Poly	81,920
Avg Pooling	147,456
Conv	12,845,056
BN	8,192
Poly	20,480
Avg Pooling	36,864
FC	81,920
Total	**15,531,392**

Operator	#OPs	Cost Mult?
Conv	2,276,736	Y
Poly	65,536	Y
Avg Pooling	131,072	N
Conv	4,333,568	Y
Conv	3,964,928	Y
Poly	16,384	Y
Fused	81,920	Y
Total	**10,870,144**	-

Figure 4.6: Comparison of the Number of Operations Before and After Fusion Using Strategy #3

Table 4.5 shows the reduced inference time and the reduced memory consumption after applying the three strategies. by using layer fusion and constant fusion. Compared to the plaintext implementation, the original version with HE adds significant amount of overheads. Note that the memory consumed by the plaintext implementation (4.6 GB) includes the Python runtime and libraries. The memory consumed by the neural network alone is 52.18 MB, which is far less than that. Since the multiplication level is reduced with the fusing strategies, we are able to decrease the polynomial modulus degree (PMD) from 16384 to 8192, which is the main impacting factor for reduced time and memory, as discussed in Chapter 3.2. In this case study, Strategy #2 delivers the best results reducing the latency to 80.2 seconds and the memory to 47.2 GB, with 86% and 78% reduction ratios respectively. Compared to Strategy #1, Strategy #2 produces fewer number of operations (#OPs) and plaintext-ciphertext multiplications (#PT-CT Mults), hence it results in less execution time. Although Strategy #3 further decreases #OPs, the execution time is actually longer than Strategy #1 and #2, because the effect of increased multiplication level is more impactful in this case, as it affects the computing time of all layers (see Table 3.2). For this case study, we use actual measurements to compare and choose the best strategy, but it is possible to employ an estimation model to predict the execution time and memory consumption, which is a potential future work to be further discussed in Chapter 5.

	Plaintext	**HE**			
		Original	**Strategy#1**	**Strategy#2**	**Strategy#3**
#Layers	9	9	5	6	7
#Mult Levels	11	11	5	5	6
PMD	16,384	8,192	8,192	8,192	8,192
#OPs	15,531,392	15,531,392	20,827,520	15,416,704	10,870,144
#PT-CT Mults	-	15,367,552	10,393,280	7,622,336	5,349,056
#PT-PT Mults	-	20,480	20,480	20,480	20,480
Time (s)/Speedup	1.3/452.7x	588.6/1x	81.3/7.24x	**80.2/7.34x**	100/5.89x
Memory (GB)	4.6/47.07x	216.5/1x	**47.2/4.59x**	**47.2/4.59x**	48.3/4.48x

Table 4.5: Comparison of the results with three fusing strategies.

Chapter 5 Conclusion

Privacy concerns in machine learning have received more and more attention in order to comply with regulations and the rise of privacy awareness. Privacy-preserving solutions using HE have emerged but are associated with additional costs, such as dropping accuracy, lengthy computation time, and inflated memory, which have severely limited the application of HE on machine learning. This thesis analyzes the critical issues behind the drop in accuracy and mentions three methods to address these issues, which result in an accuracy boost of +2.42% in our experimental studies on *Deep*. As for reducing the computing and memory resources, this thesis investigates the impact of multiplication level from the perspective of time complexity and memory profiling, and introduces non-linear fusion and constant fusion with the existing linear fusion technique to make a model more efficient to meet real-world circumstances. The experiment results show an 86% time shortening and a 78% RAM cut on *Shallow*.

While fusing linear layers reduces multiplication level, it may also changes the number of operations for additions and multiplications. Unfortunately, reducing multiplication level does not always reduces the number of operations, and vice versa. Since the total execution time depends on the platform, we have resorted to experimental measurements to choose the best fusing strategy, which can be quite tedious and time-consuming in practice. To address this trade-off, we suggest that one can follow some previous works

[25, 34] and train latency predictors and memory predictors based on layer information, HE settings, and hardware configuration to predict the latency or memory usage of the model.

References

[1] A. Al Badawi, C. Jin, J. Lin, C. F. Mun, S. J. Jie, B. H. M. Tan, X. Nan, K. M. M. Aung, and V. R. Chandrasekhar. Towards the alexnet moment for homomorphic encryption: Hcnn, the first homomorphic cnn on encrypted data with gpus. *IEEE Transactions on Emerging Topics in Computing*, 9(3):1330–1343, 2021.

[2] A. Benaissa, B. Retiat, B. Cebere, and A. E. Belfedhal. Tenseal: A library for encrypted tensor operations using homomorphic encryption, 2021.

[3] 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*, WAHC'19, page 45–56, New York, NY, USA, 2019. Association for Computing Machinery.

[4] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski. ngraph-he: A graph compiler for deep learning on homomorphically encrypted data. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, CF '19, page 3–13, New York, NY, USA, 2019. Association for Computing Machinery.

[5] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoreti-*

cal Computer Science Conference, ITCS '12, page 309–325, New York, NY, USA, 2012. Association for Computing Machinery.

[6] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha. Low latency privacy preserving inference. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 812–821. PMLR, 09–15 Jun 2019.

[7] C. Bucila, R. Caruana, and A. Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 535–541, New York, NY, USA, 2006. Association for Computing Machinery.

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

[9] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song. A full rns variant of approximate homomorphic encryption. In C. Cid and M. J. Jacobson Jr., editors, *Selected Areas in Cryptography – SAC 2018*, pages 347–368, Cham, 2019. Springer International Publishing.

[10] J. H. Cheon, A. Kim, M. Kim, and Y. Song. Homomorphic encryption for arithmetic of approximate numbers. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham, 2017. Springer International Publishing.

[11] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In J. H. Cheon and T. Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 3–33, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[12] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *CoRR*, abs/1811.09953, 2018.

[13] R. Dathathri, B. Kostova, O. Saarikivi, W. Dai, K. Laine, and M. Musuvathi. Eva: An encrypted vector arithmetic language and compiler for efficient homomorphic computation. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, page 546–561, New York, NY, USA, 2020. Association for Computing Machinery.

[14] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz. Chet: An optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 142 – 156, New York, NY, USA, 2019. Association for Computing Machinery.

[15] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144, 2012. https://eprint.iacr.org/2012/144.

[16] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, volume 9, pages 169–178, 01 2009.

[17] C. Gentry, A. Sahai, and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In R. Canetti and J. A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 75–92, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[18] 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 M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 201–210, New York, New York, USA, 20–22 Jun 2016. PMLR.

[19] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, page 218–229, New York, NY, USA, 1987. Association for Computing Machinery.

[20] T. Graepel, K. Lauter, and M. Naehrig. Ml confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptography – ICISC 2012, International Conference on Information Security and Cryptology - ICISC 2012, Lecture Notes in Computer Science, to appear*. Springer Verlag, December 2012.

[21] E. Hesamifard, H. Takabi, and M. Ghasemi. Cryptodl: Deep neural networks over encrypted data. *CoRR*, abs/1711.05189, 2017.

[22] E. Hesamifard, H. Takabi, and M. Ghasemi. Deep neural networks classification over encrypted data. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, CODASPY '19, page 97–108, New York, NY, USA, 2019. Association for Computing Machinery.

[23] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network, 2015.

[24] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[25] D. Justus, J. Brennan, S. Bonner, and A. S. McGough. Predicting the computational cost of deep learning models. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3873–3882, 2018.

[26] 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)*, pages 1651–1669, Baltimore, MD, Aug. 2018. USENIX Association.

[27] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. Innovative instructions and software model for isolated execution. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, HASP '13, New York, NY, USA, 2013. Association for Computing Machinery.

[28] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2505–2522. USENIX Association, Aug. 2020.

[29] B. Reagen, W.-S. Choi, Y. Ko, V. T. Lee, H.-H. S. Lee, G.-Y. Wei, and D. Brooks. Cheetah: Optimizing and accelerating homomorphic encryption for private inference. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 26–39, 2021.

[30] R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation, Academia Press*, pages 169–179, 1978.

[31] N. Samardzic, A. Feldmann, A. Krastev, S. Devadas, R. Dreslinski, C. Peikert, and D. Sanchez. F1: A fast and programmable accelerator for fully homomorphic encryption. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, page 238–252, New York, NY, USA, 2021. Association for Computing Machinery.

[32] N. Samardzic, A. Feldmann, A. Krastev, N. Manohar, N. Genise, S. Devadas, K. Eldefrawy, C. Peikert, and D. Sanchez. Craterlake: A hardware accelerator for efficient unbounded computation on encrypted data. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, page 173–187, New York, NY, USA, 2022. Association for Computing Machinery.

[33] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In H. Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 24–43, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[34] C.-C. Wang, Y.-C. Liao, M.-C. Kao, W.-Y. Liang, and S.-H. Hung. Perfnet: Platform-aware performance modeling for deep neural networks. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, RACS '20, page 90–95, New York, NY, USA, 2020. Association for Computing Machinery.

[35] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016.

[36] A. C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pages 160–164, 1982.