The Avg-Act Swap and Plaintext Overflow Detection in Fully Homomorphic Operations Over Deep Circuits

Ihyun Nam
Honors Thesis in Computer Science
Stanford University
May 2024

ABSTRACT

Fully homomorphic encryption is a cryptographic scheme that enables any function to be computed on encrypted data. Although homomorphic evaluation on deep circuits has many real-life applications, fully homomorphic encryption is not commercialized due to its low speed and huge computational overhead.

For the purpose of making fully homomorphic operations faster and bridge the gap between security and practicality, we introduce the Avg-Act Swap. The Avg-Act Swap is a deployable tool in privacy-preserving machine learning; it places the average pool layer *before* the activation layer as opposed to the conventional practice of ordering them the other way around in neural networks over unencrypted data. We introduce two FHE-friendly convolutional neural networks and a modified version of Lenet-5 [26] that utilize the Avg-Act Swap to demonstrate improvements in encrypted inference speed. Most notably, we improve the encrypted inference speed of Lenet-5 by 28.58% after modifying it with the Avg-Act Swap, with a 90% accuracy.

Plaintext overflow is a plausible problem in deep circuit homomorphic evaluations. We introduce (to our knowledge) the first formalized protocol to detect plaintext overflows in fixed-point arithmetic fully homomorphic encryption schemes that maintains indistinguishability over chosen plaintext attacks. We show that a remote server can homomorphically compute the maximum relative error bound of the client's plaintext only using encrypted inputs from the client. After all operations are done, the client can compare the received relative error bound to the actual error bound in the decrypted plaintext to detect an overflow. Further research to make this work in progress more efficient is encouraged.

KEYWORDS

Fully Homomorphic Encryption; Privacy Preserving Techniques; Machine Learning; Cheon-Kim-Kim-Song

ACM Reference Format:
Ihyun Nam and Honors Thesis in Computer Science, Stanford University, May 2024. 2024. The Avg-Act Swap and Plaintext Overflow Detection in Fully Homomorphic Operations Over Deep Circuits. In *Proceedings of* . ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

, ,

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Fully homomorphic encryption (FHE) offers many privacy advantages as it allows an untrusted third party to compute any arithmetic circuit on encrypted data without seeing the raw data. This is especially valuable in fields like health care, defense, or finance that need to ensure the secrecy of the processed data that may contain sensitive information. In many real-life applications of FHE, it is required or advantageous to compute FHE on deep circuits. For example, in privacy-preserving machine learning algorithms, training the algorithm over deeper circuits often increases the accuracy of the resulting model. Moreover, linear approximations of non-linear functions used in machine learning models become more accurate as they are approximated as higher degree polynomials. All these, however, come at the cost of a higher computational overhead.

There exists a trade-off between accuracy and computational efficiency of machine learning models with FHE because homomorphic operations are extremely expensive and slow. It has long been known that it is impractical to compute any reasonably large circuits using FHE for commercial applications. Against this backdrop, we propose the *Avg-Act Swap* to significantly speed up the encrypted inference speed over deep neural networks and test its performance in classifying encrypted images. Image classification is an adequate task with which to test the Avg-Act Swap as (1) it often involves deep circuit computations and (2) it is often outsourced to a (possibly untrusted) remote server due to the high computational overhead, and therefore images need to be encrypted for privacy. We train our models with *plain* images and employ FHE only during the classification of encrypted images, so computational overhead in this paper always refers to the overhead during inference, as opposed to during training. The Avg-Act Swap may be deployed in any neural network operating homomorphically on encrypted data, not just those specializing in image classifications.

Besides the speed of operation, another problem unique to performing FHE over deep circuits is plaintext overflow. Many FHE schemes define their plaintext and ciphertext spaces on a ring of some integer modulus. If the plaintext or the ciphertext grows larger than the chosen modulus, it "wraps around" and loses the encoded information. Some schemes have measures to prevent ciphertext overflows from happening; namely, 'ModChange' of the Brakerski/Fan-Vercauteren scheme (BFV) [15] and 'Rescaling' of Cheon-Kim-Kim-Song (CKKS) [10]. However, to our knowledge, there is no known method to detect *plaintext* overflows in any fixed-point arithmetic FHE scheme.

In some cases after a plaintext overflow, it is easy for the client to judge whether the decrypted result is meaningful or not, without relying on a special overflow detection mechanism. However, in

many real-life applications, overflown plaintexts may still appear valid, albeit incorrect. For example, in homomorphically classifying an encrypted image as one of 10 categories, an overflown result may indicate that an image is 'category 2' instead of 'category 5'. A client with no ground truth on the classification results has no way to tell that 'category 2' is incorrect. Furthermore, in deep circuits, it is difficult for a client to estimate the exact depth of the circuit and decide on the appropriate plaintext modulus beforehand. This is especially true in cases where the server's operations are not known to the client. Therefore, the state-of-the-art method to prevent plaintext overflows suggested in many papers [9, 11, 35] is to make sure that the plaintext modulus is sufficiently large. Since using larger encryption parameters in FHE schemes increases the computational overhead, such practices are inefficient. More importantly, in case that a plaintext overflow does happen, the client has no way of knowing that. Undetected plaintext overflows are a problem because the client will then take incorrect computation results as answers, which may have serious consequences in applications like in health care or national defense. In this paper, we focus on plaintexts in CKKS and build a plaintext overflow detection mechanism defined through a series of client-server interactions that respects FHE's indistinguishability over chosen plaintext attacks (IND-CPA security).

2 RELATED WORK

Since FHE was first formalized by Gentry [17] in 2009, there have been numerous attempts to apply the principles of FHE to real-world tasks. Some well-studied applications include using FHE in biometric authentication and DNA matching [3, 20, 43], secure banking [32], or contact tracing [39, 40], all without revealing sensitive information about the engaged parties. In particular, encrypting images with homomorphic encryption before processing them has been recognized as a powerful privacy-preserving mechanism in applications like face recognition [28, 33] or medical diagnoses based on patients' images [8, 12, 44].

Image classification tasks often involve developing and training a novel neural network based on the requirements and specifications of the particular image classification task. For example, Cryptonets [18] is a classical feed-forward neural network made of 9 layers including the convolutional layer, fully connected layer, average pooling, and two activation functions: Square and Sigmoid. Cryptonets implements the encryption scheme introduced by Bos et al. in [6], which maps plaintexts in the ring $R_t^n := \mathbb{Z}_t[x]/(x^n+1)$ to ciphertexts in the ring $R_q^n := \mathbb{Z}_q[x]/(x^n+1)$. Our work differs from Cryptonets in that we use the CKKS scheme for the encryption and decryption of images. Our plaintexts are therefore complex numbers, and a possibly indefinite number of homomorphic operations can be supported through the bootstrapping procedure of CKKS. AlexNet [2] is another powerful convolutional neural network with FHE that achieves at least an 80-bit security level and a 99% classification accuracy for encrypted MNIST [13] images.

These deep learning applications of FHE seek a temporary solution to plaintext overflow by making their plaintext modulus sufficiently large or by normalizing their plain data before encrypting them. In 2022, Lee and Shin [27] proposed Overflow-detectable Floating-point Fully Homomorphic Encryption that has an inherent plaintext overflow detection scheme using a separate ciphertext acting as a flag. However, this method is limited to FHE schemes using floating-point arithmetic and therefore is not applicable to fixed-point arithmetic schemes like CKKS, BFV, or the Brakerski-Gentry-Vaikuntanathan scheme [7]. To the best of our knowledge, this paper introduces the first plaintext overflow detection scheme demonstrated on CKKS.

3 BACKGROUND AND DEFINITIONS

In this preliminary section, we formally define FHE and the relevant algorithms of CKKS, mostly following the conventions of [10] and [17]. We also introduce relevant background knowledge in machine learning, especially on the pooling and activation layers that are involved in the Avg-Act Swap.

3.1 Fully Homomorphic Encryption

An FHE scheme has an efficient algorithm Evaluate such that for some valid public key and private key pair (pk, sk), ciphertexts $c_i \leftarrow \mathsf{Encrypt}(pk, \pi_i)$, and any efficient circuit C, it outputs

$$\pi \leftarrow \mathsf{Evaluate}(pk, C, f(\pi_1, \pi_2, \cdots, \pi_n)).$$

The result of evaluation π is such that

$$\mathsf{Decrypt}(sk, \pi) = C(\pi_1, \pi_2, \cdots, \pi_n).$$

In the context of privacy-preserving machine learning image classification, the circuit C is the trained neural network and the ciphertexts are the encrypted pixel values of client's images to classify.

3.1.1 Cheon-Kim-Kim-Song (CKKS). CKKS is a leveled homomorphic encryption scheme that supports approximate fixed-point arithmetic. That is, every CKKS ciphertext has a level (l) associated with it, which begins from a predetermined L and is reduced by 1 after each homomorphic operation on the ciphertext. If the level of a ciphertext reaches 0, no more homomorphic operations can be performed on it. The ciphertext level can be increased at this point using the CKKS bootstrapping scheme, but bootstrapping is not discussed in this paper.

For a positive integer M, let $\Phi_M(X)$ be the M-th cyclotomic polynomial of degree $N = \phi(M)$. Then, for a chosen plaintext modulus q, a CKKS plaintext is defined as a polynomial in $\mathcal{R} = \mathbb{Z}_q[X]/(\Phi_M(X))$. Let $\mathbb{H} = \{(z_j)_{j\in\mathbb{Z}_M^*} : z_{-j} = \bar{z_j}, \forall j \in \mathbb{Z}_M^*\} \subseteq \mathbb{C}^{\Phi(M)}$. Furthermore, T is a subgroup of the multiplicative group $\mathbb{Z}_M^*$ satisfying $\mathbb{Z}_M^*/T = \{\pm 1\}$. Let $\pi : \mathbb{H} \to \mathbb{C}^{\phi(M)/2}$ be the natural projection that sends a plaintext polynomial $m(X) \in \mathcal{R}$ to a vector $(z_j)_{j\in T}$. The ciphertext space is $\mathcal{R}_{q_l}^k$ where l is the level of the ciphertext, q_l is the ciphertext modulus at level l, and k is some fixed integer. A fresh ciphertext that has never been operated on homomorphically is an element of $\mathcal{R}_{q_L}^2$. Conventionally, we set the plaintext modulus q to equal the fresh ciphertext modulus q_L. We may use q_L to refer to both numbers.

Given a $(N/2)$-dimensional vector $\mathbf{z}$ of complex numbers, security parameter λ, scale factor Δ, canonical embedding σ, and ciphertext modulus q_L, the key generation, encoding, decoding, encryption, and decryption algorithms are defined below. The size of a CKKS plaintext m is defined as its canonical embedding norm $||\sigma(m)||_\infty$.

- $\mathsf{Ecd}(\mathbf{z}; \Delta) \to m(X) = \sigma^{-1}(\lfloor \Delta \cdot \pi^{-1}(\mathbf{z}) \rceil_{\sigma(\mathcal{R})}) \in \mathcal{R}$

- $\mathrm{Dcd}(m;\Delta) \rightarrow \mathbf{z} = \pi \circ \sigma(\Delta^{-1} \cdot m)$
- $\mathrm{KeyGen}(1^\lambda) \rightarrow$ sk (secret key), pk (public key), evk (evaluation key)
- $\mathrm{Enc}_{pk}(m) \rightarrow \mathbf{c} \in \mathcal{R}^k_{q_L}$ such that $\langle \mathbf{c}, sk \rangle = m + e (\mathrm{mod} q_L)$ for some small error polynomial e such that $||e||^{can}_\infty \ll ||m||^{can}_\infty$.
- $\mathrm{Dec}_{evk}(\mathbf{c}) \rightarrow m = \langle \mathbf{c}, sk \rangle (\mathrm{mod} q_l)$ for $\mathbf{c}$ at level l.

CKKS requires that $||m||_\infty \ll q_L$ in order to prevent plaintext overflows. However, the scheme provides no inherent mechanism to enforce this requirement. While a plaintext overflow is rarely a problem in the encoding phase, it is possible for $||m||_\infty$ to outgrow q_L deeper into computations [11]. Hence, we introduce the plaintext overflow detection scheme in §7.

3.1.2 Ciphertext multiplication. Multiplication between ciphertexts is one of the most expensive homomorphic operations [42]. Therefore, two natural approaches to speed up CKKS exist: (1) making each multiplication cheaper and (2) doing fewer multiplications in total. The Avg-Act Swap takes the second approach.

For two valid CKKS ciphertexts $\mathbf{c} = (c_0, c_1)$ and $\mathbf{c}' = (c'_0, c'_1)$ that encrypt m and m' respectively,

$$\mathsf{Mult}(\mathbf{c}, \mathbf{c}') \rightarrow (c_0 c'_0, c_0 c'_0 + c'_0 c_1, c_1 c'_1) \tag{1}$$

is such that

$$\mathsf{Decrypt}(sk, \mathsf{Mult}(\mathbf{c}, \mathbf{c}')) \rightarrow m' \cdot m'.$$

However, we see that (1) defines the resulting product ciphertext in terms of 3 polynomials instead of 2. Continuing to operate on this larger ciphertext exponentially increases the size of the ciphertext deeper into the circuit. Therefore, we need to perform the following relinearization and rescaling after every homomorphic multiplication in CKKS, in order to reduce the size of the product back to 2 polynomials.

3.1.3 Relinearization. Relinearization works with the relinearization key RelinKey generated by $\mathsf{KeyGen}(1^\lambda)$ and some 3-polynomial ciphertext (c_0, c_1, c_2) such that

$$\mathsf{Relin}(\mathsf{RelinKey}, (c_0, c_1, c_2)) \rightarrow (d_0, d_1)$$

and

$$\mathsf{Decrypt}(d_0, d_1) \approx \mathsf{Decrypt}(c_0, c_1, c_2).$$

3.1.4 Rescaling. For a ciphertext $\mathbf{c} \in \mathcal{R}^k_{q_l}$ at level l, and a lower level $l' < l$,

$$\left\lfloor \frac{q'_l}{q_l} \mathbf{c} \right\rceil \leftarrow \mathsf{Rescale(c)}$$

such that

$$\langle \mathbf{c}', sk \rangle = \frac{q'_l}{q_l} m + \frac{q'_l}{q_l} e + e_{\mathrm{scale}} (\mathrm{mod}\ q'_l)$$

where $e_{\mathrm{scale}} \leq B_{\mathrm{scale}}$ as introduced in §7.

3.2 Convolutional neural networks

The Avg-Act Swap involves two commonly used layers in deep neural networks: the average pooling layer (AvgPool) and the activation layer (Activation).

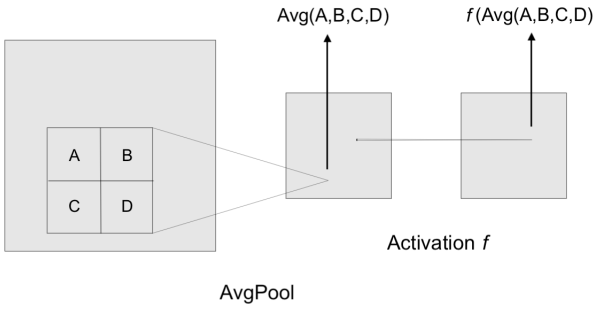

Figure 1: AvgPool and activation layers in the Avg-Act Swap

3.2.1 AvgPool. Pooling reduces the complexity of input data to extract prominent features and is therefore an essential operation in deep neural networks. As shown in Figure 1, AvgPool with a kernel size k replaces k pixels of an image in every kernel location with the average of the values. Through AvgPool, the model learns the relative importances of different regions of an image. There exist other pooling methods like max pooling and global average pooling, but AvgPool is most compatible to be computed homomorphically.

3.2.2 Activation. An activation function in machine learning is simply an arithmetic function that is computed on all pixel values of data equally to extract features from different regions of the data. Activation functions are often critical in neural networks to learn non-linearities in the data.

4 THE AVG-ACT SWAP

This section describes the first main contribution of this paper. We propose the Avg-Act Swap, which is to place AvgPool *before* Activation in neural networks over *encrypted data*. In doing so, we reduce by a factor of k the number of ciphertext multiplications, relinearizations, and rescaling operations the model needs to homomorphically compute. In the implementation of the Avg-Act Swap, Activation is entirely absorbed into the end of AvgPool in such a way that the layers are now collectively considered the new AvgPool layer, as shown in Figure 1. In neural networks over *unencrypted data*, it is conventional to place Activation *before* AvgPool, as seen in many well-known designs [18, 22, 38]. This design choice allows the model to learn more patterns in data before its complexities are reduced through pooling. Since multiplying unencrypted data is not a computationally expensive task, there is no incentive in unencrypted machine learning tasks to swap the order of AvgPool and Activation and reduce the number of multiplications done. To do so would come at the cost of drops in accuracy and only a marginal gain in speed.

However, the multiplication operation in FHE is extremely expensive. For example, one way to compute the linearly approximated Hyperbolic Tangent (Tanh) function homomorphically as $x - 0.333x^3 + 0.133x^5$ requires 3 ciphertext multiplications, 3 relinearization operations, and 5 rescaling operations as shown in Algorithm 1.

Algorithm 1 Computing Tanh homomorphically in CKKS

Require: **Ecd, Relin, RS** are valid functions to encode complex numbers into CKKS ciphertexts, relinearize a CKKS ciphertext, and rescale a CKKS ciphertext, respectively.

```
coeff1 ← Ecd(0.333)
coeff2 ← Ecd(0.133)
power_two ← x*x
Relin(power_two)
RS(power_two)

power_three ← power_two * x
Relin(power_three)
RS(power_three)

power_five ← power_two * power_three
Relin(power_five)
RS(power_five)

term2 ← coeff1 * power_three
RS(term2)
term3 ← coeff2 * power_five
RS(term3)
x ← x - term2 + term3

return x
```

We show in §6 that the Avg-Act Swap helps neural networks to achieve significantly faster encrypted inference speeds while maintaining high accuracies.

5 CONVOLUTIONAL NEURAL NETWORKS

The key to effectively using the Avg-Act Swap is to minimize the drop in accuracy after swapping the order of Activation and AvgPool. We therefore design the following 5- and 8-layer convolutional neural networks (CNN), which utilize the Avg-Act Swap while maintaining accuracies similar to their equivalents without the Avg-Act Swap (see §6 for performance evaluations).

5.1 5-layer CNN

We define $\text{CNN}_{5,\text{trad}}$ as shown below, which is a 5-layer CNN that places Activation before AvgPool like in most traditional neural networks over unencrypted data.

(1) *Convolutional Layer.* Receives an encrypted image of size (1, 28, 28). Kernel size is 5, stride is 5, and number of output channels is 4. Therefore, the output image is of size (4, 24, 24).
(2) *Activation.* Apply activation to every pixel of the input image. Therefore, the output image is of size (4, 24, 24).
(3) *AvgPool Layer.* Kernel size is 4 and stride is 4. Therefore, the output image is of size (4, 6, 6).
(4) *Flatten Layer.* Flatten the input image to one dimension. Therefore, the output image is of size (4*6*6) = (144).
(5) *Linear Layer.* The number of input features is 144 and that of output features is 10. Therefore, the output data is of size (10).

To apply the Avg-Act Swap to $\text{CNN}_{5,\text{trad}}$, we simply switch the order of AvgPool and Activation from $\text{CNN}_{5,\text{trad}}$ and get $\text{CNN}_{5,\text{swap}}$ as shown in Figure 2. We choose $k = 4$ for the kernel size in both $\text{CNN}_{5,\text{trad}}$ and $\text{CNN}_{5,\text{swap}}$, which yields the lowest (12.02%) reduction in encrypted inference time from $\text{CNN}_{5,\text{trad}}$ to $\text{CNN}_{5,\text{swap}}$. Our provided models therefore show the most conservative results, and choosing other kernel sizes ($k = 1, 2, 3, 5$) may lead to even larger increases in speed, as we show empirically in §6. The accuracy of $\text{CNN}_{5,\text{swap}}$ is 99%, which is comparable to that of many existing classifiers of encrypted images [1, 16, 21, 31].

5.2 8-layer CNN

In this section we introduce a deeper neural network that is suitable for more complicated tasks, such as classifying more complicated images. $\text{CNN}_{8,\text{trad}}$ as defined below is made of two convolutional blocks and a final linear layer to format the output as 10 probability classes.

(1) *Convolutional Layer 1.* Receives an encrypted image of size (1, 28, 28). Kernel size is 5, stride is 5, and number of output channels is 4. Therefore, the output image is of size (4, 24, 24).
(2) *Activation 1.* Apply activation to every pixel of the input image. Therefore, the output image is still of size (4, 24, 24).
(3) *AvgPool Layer 1.* Kernel size is 2 and stride is 2. Therefore, the output image is of size (4, 12, 12).
(4) *Convolutional Layer 2.* The number of input channels is 4 and the number of output channels is 12. Kernel size is 5 and stride is 5. Therefore, the output image is of size (12, 8, 8).
(5) *Activation 2.* Apply activation to every pixel of the input image. Therefore, the output image is still of size (12, 8, 8).
(6) *AvgPool Layer 2.* Kernel size is 2 and stride is 2. Therefore, the output image is of size (12, 4, 4).
(7) *Flatten Layer.* Flatten the input image to one dimension. Therefore, the output image is of size (12*4*4) = (192).
(8) *Linear Layer.* The number of input features is 192 and that of output features is 10. Therefore, the output data is of size (10).

Apply the Avg-Act Swap to the second convolutional block to obtain $\text{CNN}_{8,\text{swap}}$, whose structure is shown in Figure 3. We do not apply the Avg-Act Swap to the first convolutional block, because doing so drops the accuracy to an unreasonable value. Experimental results in §6 show that even one Avg-Act Swap in $\text{CNN}_{8,\text{trad}}$ reduces the encrypted inference time by up to 39.44% for the Square activation function.

5.3 Making CNNs FHE-friendly

We take a few measures to make the CNNs work homomorphically on encrypted data.

5.3.1 Quantization. FHE is only compatible with integers. Therefore, we use the following linear approximations of two of the most widely used polynomial activation functions in the FHE-friendly versions of our CNNs. Most extensive tests are done using the

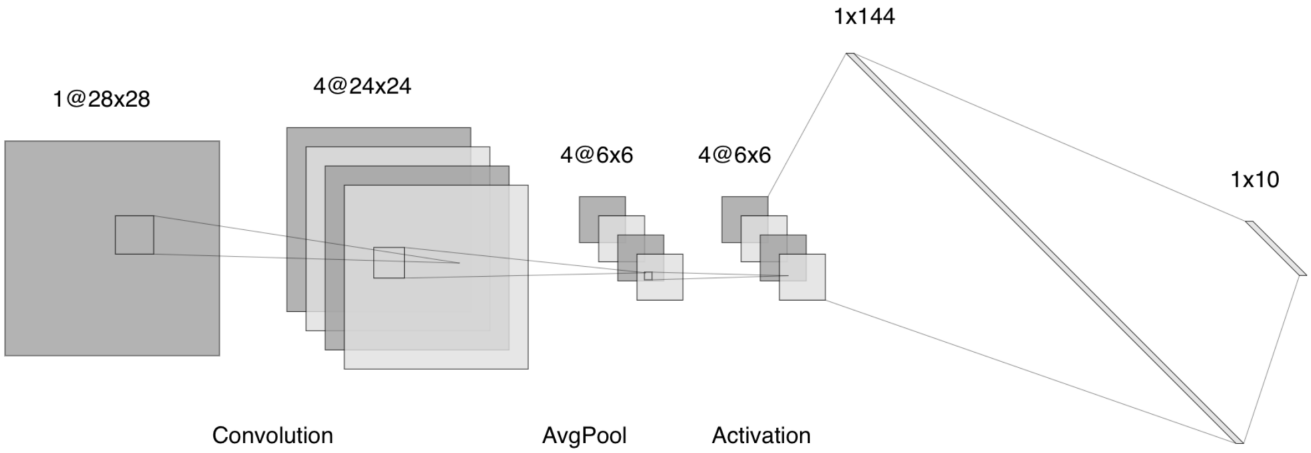

Figure 2: The structure of $\mathbf{CNN_{5,swap}}$

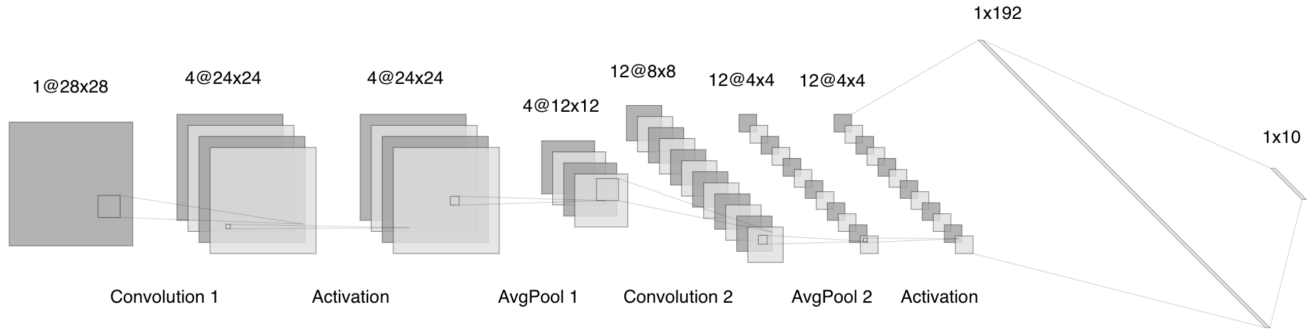

Figure 3: The structure of $\mathbf{CNN_{8,swap}}$

Square activation function, since it is used in multiple CNN encrypted inference tasks [2, 19, 24] in place of more complex activation functions that may be too expensive to compute homomorphically.

- The Square activation function is used as it is, as
$$x^2.$$
- The Tanh function is
$$\frac{e^{2x}-1}{e^{2x}+1},$$
bounded in the range (-1,1). To train both our models and for encrypted inference, we linearly approximate the Tanh function as
$$x - 0.333x^3 + 0.133x^5.$$

5.3.2 Client-Server interaction. There are two parties involved in our FHE computation: a client and a server. Like in many practical applications of FHE, we assume the client (the data owner) wants to outsource the processing of their data to an untrusted server without revealing the content of their data. In our implementations, the client generates its CKKS key pair. It uses the keys with CKKS encoding and encryption schemes to turn its images represented as complex numbers into ciphertexts. The server obtains the ciphertexts, homomorphically classifies them using one of the FHE-friendly CNNs, and returns the encrypted classification results to the client. The client decrypts the results with its CKKS private key. Notice that the server does not have access to the client's private key and therefore at no point during the processing of the encrypted images is the server able to decrypt the images being processed.

6 AVG-ACT SWAP PERFORMANCE EVALUATION

In this section, we report results of experiments that test the performance of the 5- and 8-layer CNNs. Accuracy is measured based on classifications of 100 encrypted MNIST test images and encrypted inference time is the average time needed to classify 10 encrypted

Table 1: CKKS encryption parameters for FHE-friendly CNNs with Square activation

CKKS Parameter	Value
Scale factor	2^{30}
Modulus chain	{31, 30[*8] 31}
Polynomial modulus	2^{14}
Number of plaintexts one ciphertext encodes	2^13
Number of possible multiplications	9

MNIST test images, one image at a time. Unless otherwise specified, all experiments were conducted on an e2-standard-32 machine with 32 vCPUs and 128GB of memory. The CPU platform is Intel Broadwell.

6.0.1 Machine learning implementation. We use PyTorch [34] to implement the neural networks and Pyfhel [23] to translate the neural network layers to their FHE-friendly equivalents. Pyfhel uses Microsoft SEAL [30] and OpenFHE [4] for backend support.

6.0.2 FHE implementation. We implement CKKS using PyCrCNN [14]. PyCrCNN depends on Pyfhel library v2.0.1, Laurent (SAP) and Onen (EURECOM) in the backend. We use the following CKKS encryption parameters for all instances of the FHE-friendly CNNs using the Square activation function.

The 30[*8] in the modulus chain means 30 repeated 8 times. Using 2^{30} as the scale factor is common in many toy implementations of CKKS [11, 41]. The modulus chain begins and ends with a factor slightly larger than the exponent of the scale factor, while the intermediate factors, totaling the maximum number of multiplications to be performed, are set to equal the scale factor exponent, as recommended in [10]. The actual maximum number of multiplications needed is less than 9 in most instances of our CNNs. For example, both the 8-layer CNN with Square activation and the 5-layer CNN with Tanh activation require a maximum of 7 homomorphic operations. However, we assign a few more moduli in the chain to account for any variables during computation and ensure an uninterrupted processing, as is the practice in many real-world FHE applications. Using a longer modulus chain than what is absolutely required slows down the encrypted inference; therefore, the proposed CNNs working with a tighter CKKS parameters could potentially lead to an increased speed.

6.0.3 Training and testing. We train the CNNs using 60,000 unencrypted MNIST training images. We then test the models using 10,000 unencrypted MNIST test images to check the models' accuracy without FHE. To test the accuracy of the models with FHE, we make them FHE-friendly using PyCrCNN and classify 100 encrypted MNIST test images with them.

In the FHE-friendly models, various parameters for neural layers, such as weights and bias for the convolutional layer, are first retrieved from the model trained on unencrypted images and then encrypted using the client's public key. The parameters can then be computed with encrypted data homomorphically as needed in the layers. The server must use the public key generated by the client to do so, because homomorphic operations can only be done on ciphertexts generated in the same cryptographic context.

We measure two real times for testing: (1) the sum of time taken in each neural layer except for the flatten layer (the total encrypted inference time) and (2) the time needed to complete AvgPool and Activation involved in the Avg-Act Swap (the total Avg-Act time). For both (1) and (2), we report the average time for processing 10 encrypted MNIST images, one at a time.

6.1 Performance of the 5-Layer CNN

In this section, we test how deploying the Avg-Act Swap in the 5-layer FHE-friendly CNNs affects the accuracy and the encrypted inference time of the models. Stride is always set equal to the kernel size, which is the default option from PyTorch. As seen in Table 2, the Avg-Act Swap used with the Square activation function ('Swap') achieves a 25.67% shorter encrypted inference time with a 98% accuracy, compared to its companion model ('Trad') without the Swap when $k = 2$. Furthermore, the combined time of AvgPool and Activation is reduced by up to 95.70%. We also see that the FHE-friendly CNN has only a small difference in accuracy from the plain model ('Plain') classifying unencrypted images. These results demonstrate that the Avg-Act Swap achieves significant speed improvements with marginal or no drops in accuracy. Performance evaluations using the Tanh activation function instead of the Square activation function are reported in Appendix A. Most notably, the Tanh activation used with AvgPool of kernel size 3 achieves a 47.63% increase in encrypted inference speed with a 96% accuracy. This shows that the Avg-Act Swap can support high-degree polynomial functions with high accuracy.

6.2 Performance of the 8-Layer CNNs

The test results from this section show that the Avg-Act Swap is able to support deeper neural networks. The 8-layer model achieves similar speed improvements as the 5-layer CNN with the Avg-Act Swap and with at most a 3% drop in accuracy.

6.3 Applying the Avg-Act Swap to Lenet-5

In this section, we demonstrate the usefulness of the Avg-Act Swap as a deployable tool by showing that the technique reduces the encrypted inference time of Lenet-5 [26] (or in short, Lenet) by 28.6%. Lenet is made of 9 layers of convolutional, AvgPool, Tanh activation, and linear layers. Because Lenet remains manageable even with expensive homomorphic operations and all its layers can be implemented with FHE, Lenet makes a good case study to test the performance of the Avg-Act Swap. The structure of Lenet is reproduced below.

(1) *Image preprocessing.* MNIST images of size 28×28 are padded to become 32×32.
(2) *Convolutional layer 1.* Receives an encrypted image of size (1,32,32). Kernel size is 5, stride is 1, and the number of output channels is 6. Therefore, the output image is of size (6, 28, 28).
(3) *Tanh activation 1.* Apply Tanh activation to every pixel of the input image. Therefore, the output image is of size (6,28,28).
(4) *Average pool 1.* Kernel size is 2 and stride is 2. Therefore, the output image is of size (6, 14, 14).

Table 2: Encrypted inference times (sec) of 5-layer CNNs using Square activation

Kernel size	Total encrypted inference time			Total Avg-Act time		
	Trad.	Swap	% decrease	Trad.	Swap	% decrease
2	555.61	413.00	25.67	105.71	4.55	95.70
3	524.55	423.63	19.24	105.75	15.06	85.76
4	493.61	432.38	12.40	111.80	9.77	91.26
5	509.70	411.13	19.34	104.28	4.58	95.60

Table 3: Accuracies (%) of the 5-layer CNNs using Square activation

Kernel size	Traditional accuracy			Swap accuracy		
	Plain	FHE	% change	Plain	FHE	% change
2	99	99	0	97.64	98	0
3	98.17	99	1	97.13	100	3
4	97.88	100	2	96.36	99	3
5	96.96	98	1	94.18	98	4

(5) *Convolutional layer 2.* The number of input channels is 6 and the number of output channels is 16. Kernel size is 5 and stride is 1. Therefore, the output image is of size (16,10,10).
(6) *Tanh activation 2.* Apply Tanh activation to every pixel of the input image. Therefore, the output image is of size (16,10,10).
(7) *Average pool 2.* Kernel size is 2 and stride is 2. Therefore, the output image is of size (16,5,5).
(8) *Convolutional layer 3.* The number of input channels is 16 and the number of output channels is 120. Kernel size is 5 and stride is 1. Therefore, the output image is of size (120,1,1).
(9) *Flatten layer.* Flatten the input image to one dimension. Therefore, the output image is of size (120*1*1)=(120).
(10) *Linear layer 1.* The number of input features is 120 and that of output features is 84. Therefore, the output image is of size (84).
(11) *Linear layer 2.* The number of input features is 84 and that of output features is 10. Therefore, the output image is of size (10).

We apply the Avg-Act Swap to Lenet by swapping *Tanh activation 2* and *average pool 2*, and call the new model $\text{Lenet}_{\text{swap}}$. Similarly to our design of $\text{CNN}_{8,\text{swap}}$, we Avg-Act Swap only the later occurrence of activation and AvgPool to maintain a reasonable accuracy.

6.3.1 FHE implementations. To support the increased circuit depth and subsequently more homomorphic operations, we choose increased CKKS encryption parameters as shown in Table 6 to implement FHE-friendly Lenet and $\text{Lenet}_{\text{swap}}$.

6.3.2 Performance. Below, we compare the performance (in terms of encrypted inference speed) of $\text{Lenet}_{\text{swap}}$ to that of Lenet. The experiments reported in these section were conducted on a c3-standard-88 machine with 88 vCPUs and 352GB of memory. The CPU platform is Intel Sapphire Rapids.

We see from Figure 4 that $\text{Lenet}_{\text{swap}}$ achieves a 28.58% reduction in encrypted inference time compared to Lenet, and a 76.06% reduction in the total combined time for the second occurrence of AvgPool and Activation. While Lenet achieves a 100% accuracy, $\text{Lenet}_{\text{swap}}$ achieves a 90% accuracy in classifying 100 encrypted MNIST images. While $\text{Lenet}_{\text{swap}}$ is less accurate than our 5- and 8-layer models, its accuracy is still comparable to that of many privacy-preserving machine learning models [5, 36, 37]. This demonstrates that the Avg-Act Swap may be deployed in existing deep learning models to significantly speed up encrypted inference when processing encrypted data.

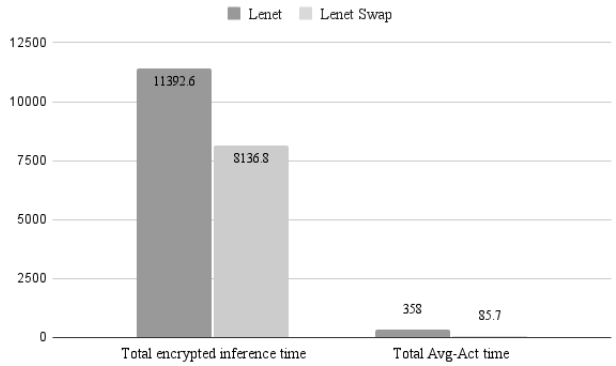

Figure 4: Lenet and $\text{Lenet}_{\text{swap}}$ time measurements

7 PLAINTEXT OVERFLOW DETECTION IN CKKS

In this final section of the paper, we introduce the first formalized method to detect plaintext overflow after homomorphic operations. In CKKS, one plaintext encodes $N/2$ complex numbers. If any one of the numbers grows larger than the plaintext modulus, all $N/2$ numbers are wiped out to lose meaning. While only a few homomorphic operations are not likely to cause a plaintext overflow since the modulus is often a large power of 2, overflow is a possibility in deep circuits where a single plaintext goes through many operations. We consider the following use case for our method.

- A remote *server* has a series of private homomorphic operations that is not known beforehand to the client.
- A *client* wishes to outsource its encrypted data to the server and retrieve the encrypted result of the server's operations performed on its data.

In addition to the CKKS preliminaries introduced in §3, we reproduce here the formal definitions of plaintext errors as introduced in [10]. A full CKKS ciphertext is defined as $(\mathbf{c}, l, v, B)$ where

Table 4: Encrypted inference times (sec) of 8-layer CNNs using Square activation

Kernel size	Total encrypted inference time			Total Avg-Act time		
	Trad.	Swap	% decrease	Trad.	Swap	% decrease
2	945.47	572.60	39.44	11.25	1.53	86.40
3	854.57	572.19	33.04	11.21	1.48	86.78
4	860.62	572.81	33.44	11.06	1.00	90.95
5	848.22	567.75	33.06	11.12	0.29	97.37

Table 5: Accuracies (%) of the 8-layer CNNs using Square activation

Kernel size	Traditional accuracy			Swap accuracy		
	Plain	FHE	% change	Plain	FHE	% change
2	99	99	0	98.78	98	-1
3	98.02	98	0	97.43	98	0
4	98.50	99	0	97.63	96	-2
5	96.41	97	0	94.63	97	2

Table 6: CKKS encryption parameters for $\text{Lenet}_{\text{swap}}$

CKKS Parameter	Value
Scale factor	2^{30}
Modulus chain	{31, 30[*15] 31}
Polynomial modulus	2^{15}
Number of plaintexts one ciphertext encodes	2^{14}
Number of possible multiplications	16

$\mathbf{c} = (\mathbf{c_0}, \mathbf{c_1})$ is the polynomial ciphertext, l is the level of the ciphertext, v is some upper bound of the underlying plaintext m such that $||m||_\infty^{can} \leq v$, and B is the error bound of $\mathbf{c}$. The error bound of a fresh ciphertext is equivalent to the encryption noise and is computed as $B_{\text{clean}} = 8\sqrt{2}\sigma N + 6\sigma\sqrt{N} + 16\sigma\sqrt{hN}$ where $h = h(1^\lambda, q_L)$ is an integer chosen while the client generates its keys, and σ is the square root of the variance of each coefficient of m. The relative error of a ciphertext is defined as $\beta = B/v$. After every homomorphic operation, the value of B is changed in a predetermined way introduced in [10], depending on the type of the operation. Therefore, it is possible to estimate the resulting value of β after a known series of homomorphic operations on the ciphertext.

7.1 Overflow Detection

We propose a method that the client can use to detect overflows in their plaintexts. This protocol has IND-CPA security as required by FHE, but does not guarantee server privacy, as is also the case in FHE. In our protocol, the server has three sets of homomorphic operations defined as follows.

- **The actual operations.** These are the operations that the server wishes to perform on the client's data. For example, this can be the functions involved in image classification.
- **The maximum error bound operations.** These are the operations to compute the final error bound B_{result} of a ciphertext after performing the *actual operations* on it. For example, if the actual operation is rescaling a ciphertext $(\mathbf{c}_1, l, v_1, B_1)$ from level l to l', the rescaled ciphertext becomes $(\mathbf{c}_2, l', p^{l-l'} \cdot v, p^{l-l'})$ [10].
- **The relative error operations.** These are the operations to compute the theoretical relative error of a ciphertext after performing the *actual operations* on it. For example, if the actual operation is multiplying $(\mathbf{c}_1, l, v_1, B_1)$ and $(\mathbf{c}_2, l, v_2, B_1)$, then the error bound operation computes

$$\beta' = \beta_1 + \beta_2 + \beta_1\beta_2 + \frac{B_{\text{mult}}(l) + p^{l-l'} \cdot B_{\text{scale}}}{v_1 v_2}.$$

The server homomorphically computes the maximum error bound and relative error of the ciphertext after it goes through many homomorphic operations. These values have to be computed separately from the actual ciphertext because the tag information (l, v, B) are not encoded into the ciphertext, but are instead conceptual features that describe the ciphertext. Computing three separate sets of homomorphic operations is extremely expensive for the server. Parallelizing these three operations, by means such as making the tag information an inherent part of the CKKS ciphertext through some packing, is a natural extension to this work.

Specifically, the client and the server engages in the following interaction to facilitate the overflow detection on the client's side.

(1) The client encrypts its message m as ciphertext c. The client also encrypts B_{clean}, σ, N, h, v_1, and v_2 as $C_{B_{\text{clean}}}$, c_σ, c_N, c_h, c_{v_1}, and c_{v_2} respectively. The client sends all encrypted values to the server.
(2) The server computes Algorithm 2.
(3) The client detects a potential overflow in its plaintext through Algorithm 3.

Relinearization is considered a part of multiplication and the errors accumulating after the two operations are computed together.

Algorithm 3 describes the actions of the client to detect the plaintext overflow, beginning from when the client receives the homomorphically computed ciphertext, error bound, and relative error bound from the server.

We assume that the inherent noise handling in CKKS works correctly, and therefore the condition $\beta' > \beta$ is only triggered by a plaintext overflow, and not an actual noise explosion in the plaintext. A correct canonical embedding for CKKS encoding ensures that the small errors inserted in plaintexts do not blow up during encoding and decoding. Furthermore, homomorphic additions do not incur a significant increase in error. Homomorphic multiplications cause a noticeable raise in plaintext noise, but the rescaling operation reduces the noise back to a manageable level by dividing it by a scale factor.

Algorithm 2 Homomorphic computations by the server

Require: Receive $\mathbf{c_1}, \mathbf{c_2}, C_{B_{\text{clean}}}, c_\sigma, c_N, c_h, c_{v_1}$, and c_{v_2} from the client.

1. Actual homomorphic operations

$\mathbf{c}_{\text{mult}} \leftarrow \text{Mult}_{evk}(\mathbf{c_1}, \mathbf{c_2})$

$\mathbf{c}_{\text{result}} \leftarrow \text{RS}_{l \to l'}(\mathbf{c}_{\text{mult}})$

return $\mathbf{c}_{\text{result}}$

2. Maximum error bound operations

$B_{\text{ks}} \leftarrow 8\sigma N/\sqrt{3}$

$B_{\text{scale}} \leftarrow \sqrt{N/3} \cdot (3 + 8\sqrt{h})$

$B_{\text{mult}}(L-1) \leftarrow p^{l'-l} \cdot q_l \cdot B_{\text{ks}} + B_{\text{scale}}$

$B_{\text{result}} \leftarrow v_1 B_2 + v_2 B_1 + B_1 B_2 + B_{\text{mult}}(L-1)$

return B_{result}

3. Relative error operations

$\beta_{\text{result}} \leftarrow \beta_1 + \beta_2 + 2^{-L-2}$

return β_{result}

Algorithm 3 Plaintext overflow detection by the client

Require: Client receives $\mathbf{c}_{\text{result}}, B_{\text{result}}$, and β_{result} from the server, all computed through algorithm 2.

$m + e \leftarrow \text{Dec}(\mathbf{c}_{\text{result}})$

$B \leftarrow \text{Dec}(c_{B_{\text{result}}})$

$\beta \leftarrow \text{Dec}(c_{\beta_{\text{result}}})$

$\beta' \leftarrow B/v$

if $\beta' > \beta$ **then**

  Plaintext overflow detected in $\mathbf{c}_{\text{result}}$

else

  No plaintext overflow detected in $\mathbf{c}_{\text{result}}$

end if

7.1.1 Possible overflow in maximum error bounds. The initial error bound B_{clean} is encrypted in the same cryptographic context as the actual message m and therefore is subject to the same overflow problem. If the error bound overflows during the maximum error bound operations of the server, then the overflow detection fails. However, we show that the probability of this happening is negligible, without formal proofs. Because $||e||_\infty^{can} \ll ||m||_\infty^{can} \ll q$ and B_{clean} are true, it follows, with some abuse of notation, that $B_{\text{clean}} \ll ||m||_\infty^{can}$. Therefore, the probability of B growing larger than q insignificant in all implementations of CKKS that correctly follow the recommended parameter setup of the original scheme.

We pay a closer attention to the error bound of a product of two ciphertexts. Multiplying $(\mathbf{c_1}, l, v_1, B_1)$ and $(\mathbf{c_2}, l, v_2, B_2)$ gives $(\mathbf{c_{mult}}, l, v_1 v_2, v_1 B_2 + v_2 B_1 + B_1 B_2 + B_{\text{mult}}(l))$ where $B_{\text{mult}}(l)$ is defined as $P^{-1} \cdot q_l \cdot B_{ks} + B_{\text{scale}}$. Since in CKKS, $q_i \approx \frac{q_{i+1}}{\Delta}$ and $q_l \neq q_L$, we conclude that $q_l < q_L$. Furthermore, in the implementation of CKKS in [10], it is stated that $P \cdot q_L$ is the largest modulus to generate an evaluation key and it suffices to assume that P is approximately equal to q_L. Therefore, $P^{-1} \cdot q_l \ll q_L$ and the possibility of B_{scale} overflowing q_L in all practical applications of our scheme is negligieble.

7.1.2 Possible overflow in relative error bounds. The relative error bounds are defined for addition and multiplication. The addition of ciphertexts with relative error bounds β_i results in a ciphertext whose error is bounded by $\max_i \beta_i$. Since the v_i of fresh ciphertexts are less than q_L, it is easy to see that additions do not produce ciphertexts with incorrect relative error bounds due to overflown v_i. The $\max_i \beta_i = \frac{B_i}{v_i}$ chosen for any sums would have $v_i < q_L$ and therefore be correct with no overflow.

The product of ciphertexts with relative error bounds β_1 and β_2 produces a new relative error bound

$$\beta' = \beta_1 + \beta_2 + \beta_1\beta_2 + \frac{B_{\text{mult}}(l) + p^{l-l'} \cdot B_{\text{scale}}}{v_1 v_2}.$$

We take a result derived in [10], which states that

$$\beta_1\beta_2 + \frac{B_{\text{mult}}(l) + p^{l-l'} \cdot B_{\text{scale}}}{v_1 v_2} \leq \beta*$$

for some constant $\beta* \leq 2^{-L-2}$. This condition ensures that the ciphertext remains decryptable after evaluation of circuits of depth less than $L-1$. Therefore, we approximate the relative error bound of a product as

$$\beta' \leq \beta_1 + \beta_2 + 2^{-L-2}$$

in the homomorphic relative error operations of the server. Doing so ensures that the computed relative error bound is correct without a possible overflow in computing $v_1 v_2$ and B_{scale}.

7.1.3 IND-CPA security. A homomorphic encryption scheme ensures IND-CPA security. That is, a probabilistic polynomial time adversary has only negligible advantage over random guessing to distinguish between ciphertexts of two distinct plaintexts (see §B for a formal definition). Our proposed overflow detection protocol respects IND-CPA security, because all additional parameters provided by the client to the server for the detection protocol are encrypted just like the client's plaintext. The server learns no additional information about the client's plaintext, compared to in an FHE scheme without our protocol. This is true even when the server learns whether the client detected a plaintext overflow or not through, for example, the client's subsequent behaviors.

7.1.4 Extensibility. Our plaintext overflow detection protocol can be modified to be used in other fixed-point arithmetic FHE schemes with notions of error similar to those of CKKS. For example, in the encryption schemes for BGV and BFV, the client samples two error terms e_1 and e_2 from a known error distribution χ. To keep the noise under control, BFV and BGV define the size of noise resulting from each type of homomorphic operation. While BFV and BGV do not explicitly define relative error like CKKS does, it is possible for the server to compute the ratio of maximum error to maximum plaintext size, using the encrypted error terms and plaintext provided by the client. We may use this ratio as the equivalent of CKKS's relative error. Furthermore, unlike in CKKS, the error terms in BFV and BGV can be separated from the plaintext after decryption. Therefore, the server only needs to perform two series of homomorphic operations: the actual operations on the plaintext and the relative error computations. The client's task to check for a plaintext overflow is now reduced to checking whether the ratio of the separated error term to the retrieved plaintext is larger than the relative error it received from the server.

8 DISCUSSION

In this section, we discuss methods to strengthen the applicability of the Avg-Act Swap and suggest a possible experimental methodology to verify and improve our plaintext overflow detection protocol. In this paper, we tested the speed improvements of using the Avg-Act Swap by computing the average time taken to classify 10 encrypted MNIST images. While this decision was made considering the slowness of encrypted inference using FHE and our limited resources, the Swap can be more thoroughly evaluated with more robust test sets, such as CIFAR-10 [25] or the full MNIST test set consisting of 10,000 images. Our plaintext overflow detection protocol is a work in progress. In particular, it has not been implemented and tested on real plaintext overflows. To test the protocol's applicability to real systems, we suggest implementing a CKKS scheme with a small modulus and applying our protocol while squaring a known integer repeatedly until it grows larger than the chosen modulus. A particularly desirable efficiency improvement for our protocol would be to avoid computing three separate sets of FHE operations.

9 CONCLUSION

In this work, we presented two separate techniques that improve the performance and integrity of computing FHE over deep circuits. The Avg-Act Swap brings us a step closer towards faster machine learning applications of FHE. While it is a common practice to perform Activation before AvgPool in neural networks over unencrypted data, we showed with experimental results that swapping the order of the two operations can significantly improve the speed of encrypted inference at the cost of a small drop in accuracy in carefully designed neural networks. Furthermore, by using the Avg-Act Swap in Lenet-5 and achieving an over 28% reduction in encrypted inference speed, we showed the potential of the Avg-Act Swap as a deployable tool in existing neural networks.

Furthermore, we introduced the first formalized plaintext overflow detection method in CKKS. Prior to this work, there was no protocol for a client to find out whether their plaintext overflowed during homomorphic operations and became meaningless. Therefore, FHE implementations resorted to setting the plaintext modulus very large in order to prevent plaintext overflows. While our plaintext detection scheme does not prevent an overflow or recover the correct result after an overflow happened, it allows the client to discern between valid and invalid decryption results. This technique is particularly useful in ensuring the integrity of data processed with FHE over deep circuits where an overflow is more likely and it is difficult to estimate the circuit depth.

REFERENCES

[1] Anish Acharya, Abolfazl Hashemi, Prateek Jain, Sujay Sanghavi, Inderjit S. Dhillon, and Ufuk Topcu. 2021. Robust training in high dimensions via block coordinate geometric median descent. https://arxiv.org/abs/2106.08882

[2] Ahmad Al Badawi, Chao Jin, Jie Lin, Chan Fook Mun, Sim Jun Jie, Benjamin Hong Meng Tan, Xiao Nan, Khin Mi Mi Aung, and Vijay Ramaseshan Chandrasekhar. 2020. 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 (2020), 1330–1343.

[3] Wilson Abel Alberto Torres, Nandita Bhattacharjee, and Bala Srinivasan. 2015. Privacy-preserving biometrics authentication systems using fully homomorphic encryption. *International Journal of Pervasive Computing and Communications* 11, 2 (2015), 151–168. https://doi.org/10.1108/ijpcc-02-2015-0012

[4] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. 2022. OpenFHE: Open-Source Fully Homomorphic Encryption Library. Cryptology ePrint Archive, Paper 2022/915. https://eprint.iacr.org/2022/915 https://eprint.iacr.org/2022/915.

[5] Ahmad Al Badawi, Jin Chao, Jie Lin, Chan Fook Mun, Jun Jie Sim, Benjamin Hong Meng Tan, Xiao Nan, Khin Mi Mi Aung, and Vijay Ramaseshan Chandrasekhar. 2018. Towards the AlexNet Moment for Homomorphic Encryption: HCNN, the First Homomorphic CNN on Encrypted Data with GPUs. Cryptology ePrint Archive, Paper 2018/1056. https://doi.org/10.1109/TETC.2020.3014636 https://eprint.iacr.org/2018/1056.

[6] Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. 2013. Improved security for a ring-based fully homomorphic encryption scheme. *Cryptography and Coding* 8308 (2013), 45–64. https://doi.org/10.1007/978-3-642-45239-0_4

[7] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory* 6, 3 (2014), 1–36. https://doi.org/10.1145/2633600

[8] Sergiu Carpov, Thanh Hai Nguyen, Renaud Sirdey, Gianpiero Constantino, and Fabio Martinelli. 2016. Practical privacy-preserving medical diagnosis using homomorphic encryption. *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)* (Jun 2016). https://doi.org/10.1109/cloud.2016.0084

[9] Jung Hee Cheon, Anamaria Costache, Radames Cruz Moreno, Wei Dai, Nicolas Gama, Mariya Georgieva, Shai Halevi, Miran Kim, Sunwoong Kim, Kim Laine, and et al. 1970. Introduction to homomorphic encryption and Schemes. https://link.springer.com/chapter/10.1007/978-3-030-77287-1_1

[10] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic Encryption for Arithmetic of Approximate Numbers. In *Advances in Cryptology – ASIACRYPT 2017*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer International Publishing, Cham, 409–437.

[11] Sangeeta Chowdhary, Wei Dai, Kim Laine, and Olli Saarikivi. 2021. EVA Improved: Compiler and Extension Library for CKKS. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography* (Virtual Event, Republic of Korea) *(WAHC '21)*. Association for Computing Machinery, New York, NY, USA, 43–55. https://doi.org/10.1145/3474366.3486929

[12] Xu Cui, Hao Liu, Min Tang, and Yihong Ma. 2023. A Medical Pre-diagnosis Scheme Based on Neural Network and Inner Product Function Encryption. In *2023 4th International Conference on Electronic Communication and Artificial Intelligence (ICECAI)*. IEEE, IEEE, Guangzhou, China, 93–97.

[13] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[14] Simone Disabato, Alessandro Falcetta, Alessio Mongelluzzo, and Manuel Roveri. 2020. A Privacy-Preserving Distributed Architecture for Deep-Learning-as-a-Service. In *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–8.

[15] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2012/144. https://eprint.iacr.org/2012/144

[16] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional sequence to sequence learning. https://arxiv.org/abs/1705.03122

[17] Craig Gentry. 2009. *A fully homomorphic encryption scheme.* Ph. D. Dissertation. Stanford University.

[18] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 48)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.). PMLR, New York, New York, USA, 201–210. https://proceedings.mlr.press/v48/gilad-bachrach16.html

[19] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 48)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.). PMLR, New York, New York, USA, 201–210. https://proceedings.mlr.press/v48/gilad-bachrach16.html

[20] Marta Gomez-Barrero, Emanuele Maiorana, Javier Galbally, Patrizio Campisi, and Julian Fierrez. 2017. Multi-biometric template protection based on homomorphic encryption. *Pattern Recognition* 67 (2017), 149–163. https://doi.org/10.1016/j.patcog.2017.01.024

[21] Ole-Christoffer Granmo, Sondre Glimsdal, Lei Jiao, Morten Goodwin, Christian W. Omlin, and Geir Thore Berge. 2019. The convolutional Tsetlin Machine. https://arxiv.org/abs/1905.09688

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV]

[23] Alberto Ibarrondo and Alexander Viand. 2021. Pyfhel: Python for homomorphic encryption libraries. In *Proceedings of the 9th on Workshop on Encrypted*

Computing & Applied Homomorphic Cryptography. Association for Computing Machinery, New York, NY, USA, 11–16.

[24] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. 2018. Secure Outsourced Matrix Computation and Application to Neural Networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 1209–1222. https://doi.org/10.1145/3243734.3243837

[25] Alex Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images. (2009), 32–33. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. https://doi.org/10.1109/5.726791

[27] Seunghwan Lee and Dong-Joon Shin. 2022. Overflow-detectable Floating-point Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2022/186. https://eprint.iacr.org/2022/186 https://eprint.iacr.org/2022/186.

[28] Xiaodong Li, Qing Han, and Xin Jin. 2018. A Secure and Efficient Face-Recognition Scheme Based on Deep Neural Network and Homomorphic Encryption. In *2018 International Conference on Virtual Reality and Visualization (ICVRV)*. IEEE, Qingdao, China, 53–57. https://doi.org/10.1109/ICVRV.2018.00017

[29] Zhen Liu, Yanbin Pan, and Tianyuan Xie. 2018. Breaking the hardness assumption and ind-CPA security of HQC submitted to NIST PQC Project. *Cryptology and Network Security* (2018), 344–356. https://doi.org/10.1007/978-3-030-00434-7_17

[30] Microsoft. 2023. Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA..

[31] Maryam Mirsadeghi, Majid Shalchian, Saeed Reza Kheradpisheh, and Timothée Masquelier. 2021. Spike time displacement based error backpropagation in convolutional spiking neural networks. https://doi.org/10.48550/arXiv.2108.13621

[32] Sonam Mittal, Priya Jindal, and K. R. Ramkumar. 2021. Data Privacy and System Security for Banking on Clouds using Homomorphic Encryption. In *2021 2nd International Conference for Emerging Technology (INCET)*. IEEE, 1–6. https://doi.org/10.1109/INCET51464.2021.9456345

[33] Vishnu Naresh Boddeti. 2018. Secure Face Matching Using Fully Homomorphic Encryption. In *2018 IEEE 9th International Conference on Biometrics Theory, Applications and Systems (BTAS)*. IEEE, Redondo Beach, CA, USA, 1–10. https://doi.org/10.1109/BTAS.2018.8698601

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., Vancouver, Canada, 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[35] Robert Podschwadt, Daniel Takabi, Peizhao Hu, Mohammad H. Rafiei, and Zhipeng Cai. 2022. A Survey of Deep Learning Architectures for Privacy-Preserving Machine Learning With Fully Homomorphic Encryption. *IEEE Access* 10 (2022), 117477–117500. https://doi.org/10.1109/ACCESS.2022.3219049

[36] Theo Ryffel, Andrew Trask, Morten Dahl, Jonathan Passerat-Palmbach, Daniel Rueckert, Jason Mancuso, and Bobby Wagner. 2018. A generic framework for privacy preserving deep learning. *(work in progress)* (2018).

[37] Mingjia Shi, Yuhao Zhou, Qing Ye, and Jiancheng Lv. 2022. Personalized Federated Learning with Hidden Information on Personalized Prior. arXiv:2211.10684 [cs.LG]

[38] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. https://arxiv.org/abs/1409.1556

[39] Koushik Sinha, Pratham Majumder, and Subhas K. Ghosh. 2020. Fully Homomorphic Encryption based Privacy-Preserving Data Acquisition and Computation for Contact Tracing. In *2020 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*. 1–6. https://doi.org/10.1109/ANTS50601.2020.9342834

[40] Shahzaib Tahir, Hasan Tahir, Ali Sajjad, Muttukrishnan Rajarajan, and Fawad Khan. 2021. Privacy-preserving COVID-19 contact tracing using blockchain. *Journal of Communications and Networks* 23, 5 (2021), 360–373. https://doi.org/10.23919/JCN.2021.000031

[41] Nattaset Tanabodee, Kalika Suksomboon, Chaveee Issariyapat, Sophon Mongkolluksamee, Aimaschana Niruntasukrat, Natapon Tansangworn, and Sukumal Kitisin. 2022. CipherFlow: A playground for developing privacy-preserving IOT in node-red. *Proceedings of the 17th Asian Internet Engineering Conference* (2022). https://doi.org/10.1145/3570748.3570752

[42] Wei Wang, Yin Hu, Lianmu Chen, Xinming Huang, and Berk Sunar. 2015. Exploring the Feasibility of Fully Homomorphic Encryption. *IEEE Trans. Comput.* 64, 3 (2015), 698–706. https://doi.org/10.1109/TC.2013.154

[43] Masaya Yasuda, Takeshi Shimoyama, Jun Kogure, Kazuhiro Yokoyama, and Takeshi Koshiba. 2013. Secure pattern matching using somewhat homomorphic encryption. *Proceedings of the 2013 ACM workshop on Cloud computing security workshop* (2013). https://doi.org/10.1145/2517488.2517497

[44] Sufang Zhou, Jianing Fan, Xiaoyu Du, Baojun Qiao, and Zhi Qiao. 2022. Efficient Multi-disease Privacy-Preserving Medical Pre-Diagnosis Based on Partial Homomorphic Encryption. In *2022 12th International Conference on Information Science and Technology (ICIST)*. 248–254. https://doi.org/10.1109/ICIST55546.2022.9926857

A PERFORMANCE EVALUATION OF TANH ACTIVATION

Below, we report the encrypted inference speed and accuracy of the Tanh activation function used in the FHE-friendly 5-layer model with the Avg-Act Swap, compared to the equivalent models without the Swap. The total encrypted inference time is the average time of processing 100 encrypted MNIST images, measured from the beginning of the first convolutional layer to the end of the linear layer.

Table 7: Accuracies (%) of FHE-friendly 5-layer CNN using Tanh activation

Kernel size	Traditional accuracy			Swap accuracy		
	Plain	FHE	% change	Plain	FHE	% change
2	97.59	99	1	96.23	93	-3
3	97.53	98	0	94.92	96	-1
4	97.35	98	1	96.20	89	-7
5	96.33	100	4	89.22	11	-78

B IND-CPA SECURITY

Given the PKE scheme $\Pi = (KGen, Enc, Dec)$ and an adversary $\mathcal{A}$, we define the following indistinguishability game.

Algorithm 4 Indistinguishablility game $PubK^{cpa}_{\mathcal{A},\Pi}(\lambda)$

$(pk, sk) \leftarrow KGen(1^\lambda)$
$(m_0, m_1) \leftarrow \mathcal{A}(1^\lambda, pk)$
$b \xleftarrow{\$} \{0, 1\}$
$\mathbf{c} \leftarrow Enc(pk, m_b)$
$b' \leftarrow \mathcal{A}(1^\lambda, pk, c)$
return 1 if $b' = b$ and 0 otherwise.

Definition B.1. [29] A PKE scheme Π is *IND-CPA secure* if for all probabilistic polynomial time adversaries $\mathcal{A}$ there is a negligible function $negl(\lambda)$ such that

$$Pr[PubK^{cpa}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \frac{1}{2} + negl(\lambda).$$

Table 8: Encrypted inference times (sec) of 5-layer CNNs using Tanh activation

Kernel size	Total encrypted inference time			Total Avg-Act time		
	Trad.	Swap	% decrease	Trad.	Swap	% decrease
2	506.47	329.59	34.92	246.57	59.85	75.73
3	508.03	266.06	47.63	245.84	26.97	89.03
4	500.88	244.44	51.20	246.55	15.21	93.83
5	510.55	242.28	52.54	244.92	7.22	97.05