future internet

Article

HeFUN: Homomorphic Encryption for Unconstrained Secure Neural Network Inference

Duy Tung Khanh Nguyen [1], Dung Hoang Duong [1,*], Willy Susilo [1], Yang-Wai Chow [1] and The Anh Ta [2]

1 Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, University of Wollongong, Northfields Avenue, Wollongong, NSW 2522, Australia; dtkn354@uowmail.edu.au (D.T.K.N.); wsusilo@uow.edu.au (W.S.); caseyc@uow.edu.au (Y.-W.C.)
2 CSIRO's Data61, Corner Vimiera & Pembroke Roads, Marsfield, NSW 2122, Australia; theanh.ta@csiro.au
* Correspondence: hduong@uow.edu.au

Abstract: Homomorphic encryption (HE) has emerged as a pivotal technology for secure neural network inference (SNNI), offering privacy-preserving computations on encrypted data. Despite active developments in this field, HE-based SNNI frameworks are impeded by three inherent limitations. Firstly, they cannot evaluate non-linear functions such as ReLU, the most widely adopted activation function in neural networks. Secondly, the permitted number of homomorphic operations on ciphertexts is bounded, consequently limiting the depth of neural networks that can be evaluated. Thirdly, the computational overhead associated with HE is prohibitively high, particularly for deep neural networks. In this paper, we introduce a novel paradigm designed to address the three limitations of HE-based SNNI. Our approach is an interactive approach that is *solely* based on HE, called iLHE. Utilizing the idea of iLHE, we present two protocols: ReLU, which facilitates the direct evaluation of the ReLU function on encrypted data, tackling the first limitation, and HeRefresh, which extends the feasible depth of neural network computations and mitigates the computational overhead, thereby addressing the second and third limitations. Based on HeReLU and HeRefresh protocols, we build a new framework for SNNI, named HeFUN. We prove that our protocols and the HeFUN framework are secure in the semi-honest security model. Empirical evaluations demonstrate that HeFUN surpasses current HE-based SNNI frameworks in multiple aspects, including security, accuracy, the number of communication rounds, and inference latency. Specifically, for a convolutional neural network with four layers on the MNIST dataset, HeFUN achieves 99.16% accuracy with an inference latency of 1.501 s, surpassing the popular HE-based framework CryptoNets proposed by Gilad-Bachrach, which achieves 98.52% accuracy with an inference latency of 3.479 s.

Keywords: privacy-preserving machine learning; secure neural network inference; homomorphic encryption

check for updates

Citation: Nguyen, D.T.K.; Duong, D.H.; Susilo, W.; Chow, Y.-W.; Ta, T.A. HeFUN: Homomorphic Encryption for Unconstrained Secure Neural Network Inference. *Future Internet* **2023**, *15*, 407. https://doi.org/10.3390/fi15120407

Academic Editor: Ivan Serina

Received: 14 November 2023
Revised: 7 December 2023
Accepted: 13 December 2023
Published: 18 December 2023

1. Introduction

Over the last decade, the field of machine learning (ML), particularly deep learning (DL), has witnessed a tremendous rise, achieving superhuman-level performance in many areas, such as image recognition [1,2], natural language processing [3,4], and speech recognition [5,6]. The success of deep neural networks (DNNs) relies on training large-scale models encompassing billions of parameters and fed with extensive datasets. For instance, ChatGPT [7], a famous large language model, was trained on the diverse and wide-ranging dataset derived from books, websites, and other texts, containing a total of 175 billion parameters. Therefore, training DNNs can be time-consuming and requires huge training samples. For such reasons, it is challenging for individual users to build DNNs of this scale. In response to this challenge, Machine Learning as a Service (MLaaS) has been proposed, enabling individual users to perform inference on their data without the necessity of local model training.

The MLaaS paradigm can be described in more detail as follows. A pre-trained neural network (NN), denoted by F, is deployed on a remote cloud server. In this setup, a client transmits a data sample, denoted by x, to the server for inference. After receiving x, the cloud server performs neural network inference on x to obtain the output $F(x)$, which will be returned to the user as the prediction result. Although MLaaS has clear benefits, it also raises serious privacy concerns for the client and the server. Firstly, from the client's perspective, the data x and the prediction result $F(x)$ may contain sensitive personal information, e.g., in healthcare applications, data x can be a medical record and the prediction $F(x)$ is whether the client has contracted a certain disease. Therefore, the client may hesitate to share x with the server and want to hide $F(x)$ from the server. Secondly, from the server's perspective, the pre-trained NN, F, represents the service provider's intellectual property and has great commercial advantages. Hence, the server does not want to share F with clients. Furthermore, the NN may leak information about the training data [8]. Consequently, it is very desirable and of great practical importance to design secure MLaaS frameworks, where the client should learn $F(x)$ but nothing else about the server's model F, while the server should not learn anything about the client's input x or the prediction result $F(x)$. Throughout this paper, we refer to this goal as *secure neural network inference* (SNNI) (other names sometimes used in the literature include *privacy-preserving inference* and *oblivious inference*).

Existing approaches to the SNNI problem naturally rely on privacy-preserving techniques of modern cryptography, in particular, multi-party computation (MPC) and fully homomorphic encryption (HE). In secure two-party computation (2PC), a special case of MPC with two parties, the client and the server interact with each other to securely compute the prediction result without revealing any individual values. After three decades of theoretical and applied work improving and optimizing 2PC protocols, we now have very efficient implementations, e.g., refs. [9–13]. The main advantage of 2PC protocols is that they are computationally inexpensive and can evaluate arbitrary operators. However, 2PC protocols require the structure (e.g., the boolean circuit) of the NN to be public and involve multiple rounds of interaction between the client and the server. On the other hand, HE-based SNNI is a non-interactive approach (the client and the server are not required to involve any communication during the inference process) and keeps all NNs' information secret from the client. The main bottlenecks of HE-based SNNI frameworks are their computational overhead and they cannot evaluate non-linear operators, e.g., the comparison operator. If prioritizing accuracy and inference time, 2PC is the better candidate. For prioritizing security and optimizing communication, HE is the better candidate. Our motivation is to design a novel framework that achieves the best of both worlds: the security and low communication cost characteristic of the HE-based approach, coupled with the computational efficiency and ability to evaluate non-linear functions inherent in the 2PC-based approach.

In this paper, we focus on the HE-based approach to SNNI. HE is a cryptographic technique that allows computations to be performed on encrypted data without decrypting it. HE-based SNNI can be described as follows. The client encrypts x and sends the encrypted data $[x]$ to the server. The server evaluates F on $[x]$ and sends the result $F([x])$ to the client, where $F([x]) = [F(x)]$ (homomorphic property). Finally, the client decrypts the ciphertext and obtains $F(x)$. The main contribution of our paper is to propose a novel interactive approach for SNNI *solely* based on HE that overcomes the aforementioned issues in 2PC and HE. The novelty of our approach is that it is only based on HE, while the existing interactive approach combines HE with 2PC protocols. Hence, we reduce communication complexity caused by 2PC protocols.

Existing (non-interactive) HE-based SNNI frameworks face three inherent drawbacks, which our approach will address.

- Firstly, existing HE schemes only support addition and multiplication, while other operators, such as comparison, are not readily available and are usually replaced by evaluating expensive (high-degree) polynomial approximation [9,10]. Hence, current HE-based SNNI frameworks **cannot practically evaluate non-linear activation functions**, e.g., ReLU function, which are widely adopted in NNs.

The solution for this issue is to replace these activation functions with (low-degree) polynomial approximations, e.g., square function. However, this replacement degrades the accuracy of the NN [11] and requires a costly re-training of the NN. Moreover, training NNs using the square activation function can lead to strange behavior when running the gradient descent algorithm, especially for deep NNs. For example, gradient explosion or overfitting phenomena may occur [12]. This is because the derivative of the square function, unlike the ReLU function, is unbounded. This paper proposes a novel protocol to evaluate the ReLU function, HeReLU, which addresses the challenge of evaluating ReLU in current HE-based SNNI frameworks.

- Secondly, in all existing HE schemes, ciphertexts contain noise, which will be accumulated after running homomorphic operations, and at some point, the noise present in a ciphertext may become too large, and the ciphertext is no longer decryptable. A family of HE supporting a predetermined number of homomorphic operators is called leveled HE (LHE). LHE is widely adopted in SNNI frameworks (LHE-based SNNI frameworks). Therefore, an inherent drawback of existing LHE-based SNNI frameworks is that they **limit the depth of NNs**.

The bootstrapping technique, an innovative technique proposed by Gentry [13], can address this issue. The bootstrapping technique produces a new ciphertext that encrypts the same message with a lower noise level so that more homomorphic operations can be performed on the ciphertext. Roughly speaking, bootstrapping homomorphically decrypts the ciphertext using encryption of the secret key, called the bootstrapping key, then (re-)encrypts the message with the bootstrapping key. This requires an extra security prerequisite, termed the circular security assumption. The assumption implies that disclosing the encryption of the secret key is presumed to be secure.

The circular security assumption remains poorly studied and poorly understood. Proving circular security for the HE schemes remains an open problem. Furthermore, bootstrapping is generally considered an expensive operation, so one usually tries to avoid it as much as possible [14]. This paper proposes a novel protocol to refresh ciphertext in HE schemes, HeRefresh, to address the noise growth issue in HE schemes. Our protocol is much faster than bootstrapping (the empirical comparison results are shown in Section 5.2) and does not require circular security.

- Thirdly, LHE-based SNNI suffers from **huge computational overhead**. The computational cost of LHE, for example, grows dramatically with the number of levels of multiplication that the scheme needs to support. This means LHE-based SNNI is impractical in the case of deep NNs (equivalent to a very large multiplication level). Interestingly, HeRefresh naturally reduces the size of LHE schemes' parameters, hence significantly reducing computational overhead (details in Section 5.3).

The non-interactive LHE-based SNNI approach cannot efficiently address the three above challenges. A natural approach that appears in the literature is interactive LHE-based approaches [11,15,16]. There are two branches in interactive LHE-based approaches. The first branch is *solely* based on LHE, which we call iLHE, and the second branch combines HE and 2PC protocols, known as the HE-MPC hybrid approach in the literature.

n-Graph-HE2 [11] is the only iLHE-based SNNI framework. n-Graph-HE2 adopted client-aided architecture where the server sends encrypted values (before the non-linear activation function) of intermediate layers to the client, and the client decrypts those encrypted values and then evaluates the activation function on intermediate layers' values; finally, the client encrypts those intermediate layers' values (after non-linear activation function) and sends it to the server. It is worth noting that the process of decryption-then-encryption can be considered as a noise refresher. This solution addresses the three mentioned issues of non-interactive LHE-based SNNI. However, nGraph-HE2 leaks the network's information, namely NNs' architecture and all intermediate layers' values.

Recent studies have illuminated the potential risks of this information leakage; a malicious client could feasibly reconstruct the entire neural network parameters by exploiting these intermediate values [17,18]. nGraph-HE2 is an ad hoc solution to address issues of LHE-based SNNI frameworks and is not considered as a separate approach to SNNI in the literature. This paper proposes a novel iLHE-based SNNI framework that addresses the drawbacks of existing non-interactive LHE-based SNNI frameworks while preserving NNs' architecture and not leaking information about intermediate layers' values. We argue that the iLHE-based approach deserves more attention from the research community.

The second branch of interactive LHE-based SNNI is the HE-MPC hybrid approach, which wisely combines HE and MPC. In particular, HE schemes were used to compute linear layers, i.e., fully connected layer and convolution layer, while 2PC protocols, e.g., garbled circuit (GC) [19] and ABY [20], were used to compute an *exact* non-linear activation function. The major bottleneck of the approach is the communication complexity caused by 2PC protocols. On the other hand, our framework is an iLHE-based approach, which is *solely* based on HE. Hence, we naturally avoid expensive 2PC protocols.

To address the three challenges faced by non-interactive LHE-based SNNI, this paper designs two novel iLHE-based protocols. Firstly, we design the HeReLU protocol to evaluate the ReLU activation function, an essential non-linear function widely adopted in NNs. This protocol addresses the first challenge of non-linear function evaluation in HE-based SNNI. Compared to current LHE-based SNNI frameworks, our protocol can *exactly* evaluate the ReLU function. Compared to existing HE-MPC hybrid frameworks, i.e., Gazelle [15] and MP2ML [16], our protocol achieves fewer communication rounds (we detail this in Section 5.1). Secondly, we design HeRefresh to *refresh* HE ciphertexts, which enable further homomorphic operators. This protocol addresses the second challenge of noise accumulation in HE-based SNNI. By using HeRefresh, rather than selecting HE parameters large enough to support the entire NN, the HE's parameters must now only be large enough to support the computation on linear layers between ReLU activation function layers. For instance, to perform secure inference on a seven-layer NN, which contains convolution layers, fully connected layers followed by ReLU activation function layers, our framework (with HeRefresh) requires $L = 3 \ll 7$, while current non-interactive LHE-based SNNI frameworks (without HeRefresh) require $L = 7$ (details in Section 5.3). Therefore, HeRefresh can address the third challenge of computational overhead due to large multiplication depth in LHE-based SNNI. In the HE scheme, HeRefresh plays the role of a bootstrapping procedure, which aims to reduce the noise in ciphertexts. Our experiment showed that HeRefresh outperforms bootstrapping by $300\times$ in running time (we detail this in Section 5.2). Interestingly, HeRefresh and HeReLU can run in parallel. Hence, we can save communication rounds.

Our contribution. In this paper, we first design two novel protocols, i.e., HeReLU and HeRefresh, to address the drawbacks of existing LHE-based SNNI frameworks. Then, we leverage HeReLU and HeRefresh to build a new framework for SNNI called HeFUN. The idea of HeFUN is to use HeReLU to evaluate the ReLU activation function and use HeRefresh to refresh intermediate neural network layers. The benefits of our proposed method are twofold. Firstly, compared to current HE-based frameworks, our approach significantly accelerates inference time while achieving superior accuracy. Secondly, compared to current 2PC-based frameworks, our methodology reduces communication rounds and safeguards circuit privacy for service providers. We highlight the superiority of our approach compared to current approaches in Section 2. A comparison summary is shown in Table 1. The contribution of this paper is as follows:

- We proposed a novel iLHE-based protocol, HeReLU, that can exactly evaluate the ReLU function *solely* based on HE, which achieves better communication rounds than current HE-MPC hybrid frameworks. The analysis is presented in Section 5.1.
- We proposed a novel iLHE-based protocol, HeRefresh, that refreshes noise in ciphertexts to enable arbitrary depth of the circuit. HeRefresh also reduces HE parameters' size, hence significantly reducing computation time. Considering the ciphertext

refreshing purpose, HeRefresh outperforms bootstrapping 300× in relation to computation time (we detail this in Section 5.2). Furthermore, our protocol deviates from Gentry's bootstrapping technique, so we bypass the circular security requirement.

- We build a new iLHE-based framework for SNNI, named HeFUN, which uses HeReLU to evaluate the ReLU activation function and use HeRefresh to refresh intermediate neural network layers.
- We provide security proofs for the proposed protocols. All proposed protocols, i.e., HeReLU and HeRefresh, are proven to be secure in the semi-honest model using the simulation paradigm [21]. Our framework, HeFUN, is created by composing sequential protocols. The security proof of the HeFUN framework is provided based on modular sequential composition [22].
- Experiments show that HeFUN outperform previous HE-based SNNI frameworks. Particularly, we achieve higher accuracy and better inference time.

Table 1. Comparison of SNNI frameworks.

Approach	Framework	Security	Accuracy			Efficiency		
		Circuit Privacy	Comparable Accuracy [a]	Non-Linear	Unbounded	Non-Interactive [b]	SIMD	Small HE Params [c]
LHE	CryptoNets	✓	✓	✗	✗	✓	✓	✗
	CryptoDL	✓	✓	✗	✗	✓	✓	✗
	BNormCrypt	✓	✓	✗	✗	✓	✓	✗
	Faster-CryptoNets	✓	✓	✗	✗	✓	✓	✗
	HCNN	✓	✓	✗	✗	✓	✓	✗
	E2DM	✓	✓	✗	✗	✓	✓	✗
	LoLa	✓	✓	✗	✗	✓	✓	✗
	CHET	✓	✓	✗	✗	✓	✓	✗
	SEALion	✓	✓	✗	✗	✓	✓	✗
	nGraph-HE	✓	✓	✗	✗	✓	✓	✗
FHE	FHE-DiNN	✓	✗	✓	✓	✓	✗	✓
	TAPAS	✓	✗	✓	✓	✓	✗	✓
	SHE	✓	✗	✓	✓	✓	✗	✓
MPC	DeepSecure	✗	✗	✓	✓	✗	-	-
	XONN	✗	✗	✓	✓	✗	-	-
	GarbledNN	✗	✗	✓	✓	✗	-	-
	QUOTIENT	✗	✓	✓	✓	✗	-	-
	Chameleon	✗	✓	✓	✓	✗	-	-
	ABY3	✗	✓	✓	✓	✗	-	-
	SecureNN	✗	✓	✓	✓	✗	-	-
	FalconN	✗	✓	✓	✓	✗	-	-
	CrypTFlow	✗	✓	✓	✓	✗	-	-
	Crypten	✗	✓	✓	✓	✗	-	-
HE-MPC hybrid	Gazelle	✓	✓	✓	✓	✗	✓	✓
	MP2ML	✓	✓	✓	✓	✗	✓	✓
iLHE	nGraph-HE2	✗	✓	✓	✓	✗	✓	✓
	HeFUN (this work)	✓	✓	✓	✓	✗	✓	✓

[a] Among frameworks considered having comparable accuracy with plain model, LHE-based frameworks are less accurate than others because they approximate non-linear function by polynomial, while MPC-based, HE-MPC hybrid-based, and iLHE-based approaches achieve the same accuracy as the plain model (details in Section 5.3).
[b] HeFUN requires less communication rounds than HE-MPC hybrid-based approach (details in Section 5.1).
[c] The size of HE parameters directly impacts the inference time of SNNI frameworks. HeFUN uses smaller HE parameters than current LHE-based frameworks, hence significantly accelerating inference time (details in Section 5.3).

The rest of the paper is organized as follows. Section 2 reviews current SNNI frameworks, Section 3 provides the necessary cryptographic background, Section 4 presents our proposed approach, Section 5 presents our implementation and evaluation results, Section 6 provides some discussions and potential future works, and Section 7 concludes our work.

2. Related Works

This section provides a summary of current approaches for SNNI. Existing solutions for SNNI can be categorized into four main approaches: LHE-based (Section 2.1), TFHE-based (Section 2.2), MPC-based (Section 2.3), and HE-MPC hybrid approach (Section 2.4). We now overview those approaches. Table 1 summarizes the comparison between existing SNNI approaches.

2.1. LHE-Based SNNI

This approach utilizes leveled homomorphic encryption (LHE) schemes, e.g., BFV [23] and CKKS [24]. Such schemes allow a limited and predefined number of operations (e.g., additions and multiplications) that can be performed on encrypted data. The computational cost of LHE-based SNNI frameworks grows dramatically with the number of layers in NNs. CryptoNets [12] by Microsoft Research is the initial work for LHE-based SNNI. In this paper, the authors first presented an SNNI framework for a five-layer NN on the MNIST dataset. CryptoNets triggered significant interest in using LHE for SNNI. Based on the idea of CryptoNets, many subsequent works were proposed to improve the performance, scaled to deeper networks, and integrated with deep NN compiler technologies, e.g., CryptoDL [25], BNormCrypt [26], Faster-CryptoNets [27], HCNN [28], E2DM [29], LoLa [30], CHET [31], SEALion [32], nGraph-HE [33], and nGraph-HE2 [11]. An essential feature of LHE schemes is that they offer the Single-Instruction-Multiple-Data (SIMD) technique [34], where we can pack multiple plain data into a single ciphertext and performing homomorphic operations on the ciphertext corresponds to simultaneously performing operations on multiple plain data. This feature helps amortize the cost of homomorphic operations, so it increases the throughput of the inference process. However, LHE-based SNNI frameworks have three inherent shortcomings.

Firstly, LHE-based SNNI frameworks cannot evaluate the ReLU function, commonly used in deep learning. This is because existing LHE schemes do not support the comparison operation, which is required in computing the ReLU function. One solution widely adopted to address the issue is to replace the ReLU function with a polynomial approximation, e.g., CryptoNets replaces the ReLU function with the square function ($f(x) = x^2$). However, this solution compromises the accuracy of the NN [11]. Additionally, it requires re-training the entire NN, which is a costly procedure. Furthermore, training an NN with polynomial approximation, e.g., square function, unlike the ReLU function, may lead to strange behavior, as the derivative of the square function is unbounded.

Secondly, LHE-based SNNI restricts the depth of the NNs, as LHE schemes allow only a limited number of operations that can be performed on the encrypted data. Thus, this method is not scalable and suitable for deep learning, where NNs can contain tens, hundreds, or sometimes thousands of layers [35,36]. These issues can be addressed by using the bootstrapping technique [13]. However, the bootstrapping technique requires circular security and is a costly procedure. So one usually tries to avoid it as much as possible [14].

Thirdly, and perhaps more importantly, even with many efforts to improve the performance, the computational cost of LHE-based SNNI frameworks is prohibitively large. This is because inference on deep NNs requires a large multiplicative level, leading to large LHE schemes' parameters.

HeFUN addresses all three mentioned drawbacks of LHE-based SNNI thanks to two novel protocols. That is, the HeReLU protocol can exactly evaluate the ReLU function, which addresses the first drawback, and HeRefresh can refresh ciphertexts, which addresses the second and third drawbacks.

2.2. TFHE-Based SNNI

TFHE [37] is an HE scheme that represents ciphertexts in the torus modulo 1. TFHE can execute rapid binary operations on encrypted binary bits. Furthermore, TFHE is equipped with a fast bootstrapping technique. Compared to LHE schemes, TFHE has two advantages. Firstly, unlike the LHE schemes approximating the ReLU by expensive polynomial operations, TFHE can exactly evaluate the ReLU function by boolean operations. Secondly, TFHE allows unbounded homomorphic operators that can be performed on ciphertext, thanks to the fast bootstrapping technique. However, TFHE only works on binary gates, and current TFHE-based SNNI frameworks work with binarized neural networks (BNNs) [38]. BNNs restrict the weights and inputs of their linear layers to the set $\{-1, 1\}$. It employs the activation function sign, which is straightforward to implement in boolean circuits. FHE-DiNN [39] is the first TFHE-based SNNI framework. While FHE-DiNN performs inference on MNIST faster than CryptoNets, its accuracy is notably lower than CryptoNets because it is limited to supporting smaller BNNs.

Subsequent works, i.e., TAPAS [40] and SHE [41], improved the efficiency of FHE-DiNN, but their core idea was merely based on FHE-DiNN, i.e., using TFHE over BNN. Due to the limitation of operating with BNN, the accuracy of TFHE-based frameworks falls short when compared to the plain NN (running inference on the original NN). For example, ref. [42] evaluated the most prevalent HE schemes for SNNI, namely CKKS, BFV, and TFHE; while CKKS and BFV achieve comparable accuracy with the plain NN (96.34%), TFHE's accuracy was significantly lower (82.70%). As shown in Table 1, while other approaches achieve comparable accuracy with the plain NN, the TFHE-based approach falls short in accuracy. Furthermore, since the packing technique does not easily align with the TFHE scheme, it might introduce extra inefficiencies in terms of running time and memory overhead when we want to process large amounts of data concurrently.

Compared to the TFHE-based approach, HeFUN is more accurate (we achieve the same accuracy with the plain NN) and efficient (supports SIMD). Furthermore, HeFUN also achieves unbounded depth and can exactly evaluate the ReLU activation function.

2.3. MPC-Based SNNI

MPC is a cryptographic technique that enables two or more parties to jointly compute a function f over their respective secret inputs without revealing their secret inputs to each other. SNNI is a special case of MPC, i.e., secure two-party computation (2PC), where the client and the server jointly evaluate the prediction function on private data (held by the client) and the NN's parameters (held by the server). GC is an MPC protocol commonly used in MPC-based SNNI frameworks. In the context of SNNI, when the bit length of the numbers in use is fixed, all the operations in an NN can be represented using a boolean circuit. This means the entire NN can be translated into one (likely extensive) boolean circuit. The GC protocol [19] provides a method to evaluate such a boolean circuit securely. DeepSecure [43] was pioneered as the first SNNI framework that purely relied on GC. Following this, XONN [44] and GarbledNN [45] are other frameworks that follow this purely GC approach. Such purely GC frameworks suffer from three drawbacks. Firstly, the boolean circuits are unsuitable for performing arithmetic operations (used in NN). Hence, all these frameworks require some modification on NN to make them compatible with boolean circuits, e.g., XONN works on BNNs. This modification significantly drops the accuracy of the NN. Secondly, the purely GC-based approach is bulky and incurs a large overhead [46]. This is considered as the major hurdle of the purely GC-based approach. Thirdly, it leaks the entire NN architecture to the client, i.e., the number of layers, type of layers, and the number of neurons in each layer.

Secret Sharing (SS) [47] and Boolean SS (also known as GMW) [48] are other common MPC protocols used in MPC-based SNNI. Many SS/GMW-based frameworks have been proposed to address SNNI, e.g., SecureML [49], ABY2 [50], QUOTIENT [51], Chameleon [52], ABY3 [20], SecureNN [53], FalconN [54], CrypTFlow [55], and Crypten [56]. These frameworks are highly efficient. Typically, they exhibit superior performance compared to existing SNNI approaches [46]. It is worth noting that many of them support private training (not only SNNI). Although SS/GMW-based frameworks offer many benefits, such as the fact that they can evaluate any function, including ReLU, and they are very efficient, they suffer from a serious security issue similar to the purely GC-based approach, namely, they leak the entire network architecture to the client.

Overall, MPC-based SNNI frameworks provide a lower level of security than other SNNI approaches (including HeFUN), as they leak the entire NN's architecture.

2.4. HE-MPC Hybrid SNNI

HE-MPC Hybrid SNNI judiciously combines HE and MPC. In particular, HE-MPC Hybrid SNNI frameworks compute linear layers, e.g., fully connected and convolutional layers, using HE and non-linear activation functions, e.g., ReLU using MPC. Gazelle [15] was the first work to combine an LHE scheme, i.e., BFV, to compute linear layers with GC to compute non-linear layers. Gazelle [15] opened a new approach, which outperformed other approaches, i.e., LHE-based, TFHE-based, and MPC-based, at the time of publication (2018). Based on the idea of Gazelle, MP2ML [16] was proposed, which combines the CKKS scheme with ABY [57] (BFV was replaced by CKKS and GC was replaced by ABY). Although Gazelle and MP2ML can evaluate the ReLU function, offer efficient solutions, and keep the NN architecture secret from the client, the main bottleneck of these frameworks is the GC/ABY block, where the client and the server have to interact to jointly compute non-linear functions. Additionally, HE-MPC hybrid frameworks require converting between HE ciphertexts and MPC values.

On the other hand, our framework, HeFUN, is an MPC-free interactive SNNI framework where we use HE only as the cryptographic primitive. Compared to HE-MPC hybrid frameworks, HeFUN is simpler and requires fewer communication rounds, as HeFUN escapes from the main bottleneck block, i.e., GC/ABY. We detail this in Section 5.1.

Our protocol, HeFUN, overcomes the shortcomings of previous approaches by introducing two novel protocols for evaluating the ReLU function (HeReLU) and refreshing ciphertext (HeRefresh). Table 1 shows a comparison between HeFUN and existing approaches. HeFUN satisfies security, accuracy, and efficiency criteria. Notably, HeFUN is the first framework that meets all criteria only using LHE. The idea is that we allow the client and the server to interact *solely* based on HE (without the need for any MPC protocols). We call this approach interactive LHE (iLHE).

3. Cryptographic Preliminaries

3.1. Notation

We denote a scalar value in lower-case letters, e.g., x. We denote vectors in lower-case bold letters, e.g., $\boldsymbol{x}$, and we refer to the ith element of a vector as $\boldsymbol{x}_i$, with indexing starting from one. For any two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$, $\boldsymbol{u} + \boldsymbol{v}$ and $\boldsymbol{u} \times \boldsymbol{v}$ signify component-wise addition and multiplication, respectively, while $\boldsymbol{u} \cdot \boldsymbol{v}$ denotes the dot product between $\boldsymbol{u}$ and $\boldsymbol{v}$. We use $\langle \boldsymbol{u} \rangle_r$ to denote the cyclically leftward-shifted (rotation) vector of $\boldsymbol{u}$ by r positions, with the understanding that a negative r corresponds to a rightward shift. We denote a matrix in capital bold letters, e.g., $\boldsymbol{W}$, and we refer to the ith column of a vector as $\boldsymbol{W}_i$. We let $\boldsymbol{x} \cdot \boldsymbol{W}$ denote vector–matrix multiplication. Additionally, we use $\mathcal{R}$ to denote the cyclotomic ring $\mathbb{Z}[X]/\langle X^N + 1 \rangle$, where N is a power of two, and $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$.

3.2. Homomorphic Encryption

Homomorphic encryption (HE) is a distinctive form of encryption that permits computation on ciphertexts without the need for decryption, thereby ensuring data confidentiality, even within untrusted environments. Current HE schemes vary, particularly concerning the data types they accommodate. For instance, integer-based HE schemes facilitate operations on integer vectors, whereas real number-based HE schemes cater to computations involving real vectors. Although our proposed frameworks are compatible with both types, our primary focus is on the CKKS scheme [10], which is prevalent in SNNI due to its inherent compatibility with real numbers.

In CKKS, a ciphertext takes the form of $(b, a) \in \mathcal{R}_q^2$, with q being a prime number. The scheme enables the encryption of a vector consisting of $N/2$ real numbers into $N/2$ slots of a single ciphertext, thanks to the batching technique [10]. CKKS facilitates homomorphic operations, applying identical computations to each slot concurrently. In particular, if we represent the ciphertext of a vector $\boldsymbol{x} \in \mathbb{R}^{N/2}$ as $[\boldsymbol{x}]$, we can describe homomorphic operators as follows: $[\boldsymbol{x}] \oplus [\boldsymbol{y}] = [\boldsymbol{x}+\boldsymbol{y}]$, $[\boldsymbol{x}] \oplus \boldsymbol{y} = [\boldsymbol{x}+\boldsymbol{y}]$, $[\boldsymbol{x}] \otimes [\boldsymbol{y}] = [\boldsymbol{x} \times \boldsymbol{y}]$, $[\boldsymbol{x}] \otimes \boldsymbol{y} = [\boldsymbol{x} \times \boldsymbol{y}]$, and $\mathsf{Rotate}([\boldsymbol{x}], r) = [\langle \boldsymbol{x} \rangle_r]$.

LHE-based SNNI. Leveraging the operators supported by homomorphic encryption, numerous LHE-based SNNI frameworks have been developed (as can be seen in Table 1). In NNs, two common operators that can be efficiently evaluated using HE schemes are matrix multiplication and convolution. A comprehensive explanation of these methods is beyond the scope of this paper; for detailed insights, we direct readers to [58–61]. We briefly overview the operators that we used in this paper.

Given a vector $\boldsymbol{x}$ and a matrix $\boldsymbol{W}$, the vector–matrix multiplication can be efficiently evaluated in the HE scheme, i.e., $[\boldsymbol{x}] \odot \boldsymbol{W} = [\boldsymbol{x} \cdot \boldsymbol{W}]$, where $\odot$ denotes homomorphic (encrypted) vector and (plain) matrix multiplication. In the context of convolution, given an input image $\boldsymbol{x}$ (we repurpose the symbol $\boldsymbol{x}$, typically used for vectors, to represent images in this context) and a filter $\boldsymbol{W}$ (for clarity, we presume the filter $\boldsymbol{W}$ possesses a singular map; in scenarios with multiple maps, the operation can be repeated accordingly), we have $[\boldsymbol{x}] \circledast \boldsymbol{W} = [\boldsymbol{x} * \boldsymbol{W}]$, where $*$ and $\circledast$ denote convolution and homomorphic convolution, respectively. In this paper, we employ convolution based on [60], in which convolution between an encrypted vector and a plain matrix can be computed based on element-wise operators in HE schemes. Additionally, it is noteworthy that the dot product is essentially a specific instance of matrix multiplication, illustrated as $[\boldsymbol{x}] \odot \boldsymbol{y} = [\boldsymbol{x}.\boldsymbol{y}]$.

LHE. In CKKS, each ciphertext has a non-negative integer, termed as level, representing the capacity for homomorphic multiplication operations. After each homomorphic multiplication, the level is reduced by one through the rescaling procedure. After several multiplications, if the level ℓ reduces to zero, the ciphertext can no longer support further multiplication operations. The initial setting of ℓ is crucial and is determined based on the expected number of multiplications. For deep NN, ℓ is large, resulting in a corresponding increase in the CKKS parameters, which subsequently leads to computational overheads. Another solution to expend ciphertext's computational capacity is the bootstrapping procedure [13] that turns the zero-level ciphertext into a higher-level ciphertext, thereby enabling further homomorphic multiplications. However, bootstrapping is commonly considered as a costly procedure, so one usually tries to avoid it as much as possible [14].

Security of HE. An essential attribute of the CKKS is its semantic security. This ensures that a computationally bounded adversary cannot distinguish whether a given ciphertext corresponds to the encryption of one of two different messages. Intriguingly, even if two messages are identical (i.e., $\boldsymbol{x} = \boldsymbol{y}$), their respective ciphertexts will differ (i.e., $[\boldsymbol{x}] \neq [\boldsymbol{y}]$). This is due to random noise introduced during the encryption process. The robustness of CKKS's semantic security is based on the Ring Learning with Errors (RLWE) assumption [62]. The RLWE problem has undergone extensive scrutiny within the cryptographic community. Significantly, it is deemed a computationally hard problem, resistant to quantum attacks, thereby positing RLWE-based schemes like CKKS as viable options in the quantum era.

3.3. Threat Model and Security

We prove the security of our proposed protocols against a semi-honest adversary by following the simulation paradigm in [21]. We consider two parties, $\mathcal{C}$ and $\mathcal{S}$, and jointly compute a function $f = (f_C, f_S)$ by running a protocol Π. A computationally bounded adversary $\mathcal{A}$ corrupts either $\mathcal{C}$ or $\mathcal{S}$ at the beginning of the protocol Π and follows the protocol specification honestly but attempts to infer additional sensitive information (e.g., information about the input of the other party) from the observed messages during protocol execution. We now give the formal definition of security within the semi-honest model. Intuitively, the definition guarantees that everything a party observes from the received message of a legitimate execution is essentially inferred from its output (coupled with its corresponding input). In other words, any knowledge the party acquires from the protocol execution is fundamentally implied in the output itself, leaving the party with only the output as new knowledge.

Definition 1. *A protocol Π between a party $\mathcal{C}$ and a party $\mathcal{S}$ is said to securely compute a function f in semi-honest model if it satisfies the following guarantees:*

- ***Correctness:*** *For any $\mathcal{C}$'s input $\boldsymbol{x}$ and $\mathcal{S}$'s input $\boldsymbol{y}$, the probability that, at the end of the protocol, $\mathcal{C}$ outputs $f_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})$ and $\mathcal{S}$ outputs $f_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y})$ is 1.*
- ***Security:***
  - ***Corrupted $\mathcal{C}$:*** *For $\mathcal{C}$ that follows the protocol, there exists a simulator $\mathsf{Sim}_{\mathcal{C}}$ such that $\mathsf{View}_{\mathcal{C}} \approx_c \mathsf{Sim}_{\mathcal{C}}(\boldsymbol{x}, f_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y}))$ where $\mathsf{View}_{\mathcal{C}}$ denotes the view of $\mathcal{C}$ in the execution of Π (the view includes the $\mathcal{C}$'s input, randomness, and the message $\mathcal{C}$ received during the protocol), and $f_{\mathcal{C}}(\boldsymbol{x}, \boldsymbol{y})$ denotes the ultimate output $\mathcal{C}$ received at the end of Π.*
  - ***Corrupted $\mathcal{S}$:*** *For $\mathcal{S}$ that follows the protocol, there exists a simulator $\mathsf{Sim}_{\mathcal{S}}$ such that $\mathsf{View}_{\mathcal{S}} \approx_c \mathsf{Sim}_{\mathcal{S}}(\boldsymbol{x}, f_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y}))$ where $\mathsf{View}_{\mathcal{S}}$ denotes the view of $\mathcal{S}$ in the execution of Π, and $f_{\mathcal{S}}(\boldsymbol{x}, \boldsymbol{y})$ denotes the ultimate output $\mathcal{S}$ received at the end of Π.*

Our framework (Section 4) is created by composing sequential secure protocols. To prove the security of our framework, we invoke modular sequential composition, as introduced in [22]. The details of the modular sequential composition are out of the scope of this work. Please refer to [22] for details. We now briefly describe the idea of modular sequential composition, which we used to prove security for our framework. The idea is that two parties $\mathcal{C}$ and $\mathcal{S}$ run a protocol Π and use calls to an ideal functionality $\mathcal{F}$ to compute a function f (e.g., $\mathcal{C}$ and $\mathcal{S}$ may privately compute f by transmitting their inputs to a trusted third party and subsequently receiving the output) in Π. If we can prove that Π is secure in the semi-honest model and if we have a protocol ρ that privately computes f in the same model, then we can replace the ideal calls to $\mathcal{F}$ by the execution of ρ in Π; the new protocol denoted Π_ρ is then secure in the semi-honest model.

4. HeFUN: An iLHE-Based SNNI Framework

This section provides a detailed description of our proposed framework. We first state the problem we try to solve throughout this section (Section 4.1). Secondly, we present a protocol to homomorphically evaluate the ReLU function, i.e., HeReLU (Section 4.2). Thirdly, we present a protocol to refresh ciphertexts, i.e., HeRefresh (Section 4.3). Fourthly, we present a naive framework for SNNI, i.e., $\mathsf{HeFUN}_{\mathsf{naive}}$, that straightforwardly applies the HeReLU and HeRefresh, but it leaks some unexpected information to the client (Section 4.4). Then, we revise $\mathsf{HeFUN}_{\mathsf{naive}}$ to prevent the potential information leakage, which results in our "ultimate" framework, i.e., HeFUN (Section 4.5). Throughout this section, we denote the client by $\mathcal{C}$ and the server by $\mathcal{S}$.

4.1. Problem Statement

We consider a standard scenario for cloud-based prediction services. In this context, $\mathcal{S}$ possesses an NN, and $\mathcal{C}$ sends the data to $\mathcal{S}$ and subsequently obtains the corresponding prediction. The problem is formally defined as:

$$\boldsymbol{z} := a_L\Big(f_L\Big(\boldsymbol{W}^{(L)}, a_{L-1}\Big(f_{L-1}\Big(\dots a_1\Big(f_1\Big(\boldsymbol{W}^{(1)}, \boldsymbol{x}, \boldsymbol{b}^{(1)}\Big)\Big)\Big)\Big), \boldsymbol{b}^{(L)}\Big)\Big)$$

where $\boldsymbol{x}$ is the input data, f_i is the linear transformation applied in the ith layer, a_i is (usually non-linear) activation functions, and $\boldsymbol{W}^{(i)}$ and $\boldsymbol{b}^{(i)}$ are the weight and bias in the i-th layer of the NN. The problem we address is SNNI. That is, after each prediction, $\mathcal{C}$ obtains the prediction $\boldsymbol{z}$, while $\mathcal{S}$ learns nothing about $\boldsymbol{x}$ and $\boldsymbol{z}$, and $\mathcal{C}$ learns nothing about $\boldsymbol{W}^{(i)}$ and $\boldsymbol{b}^{(i)}$ with $i = \overline{1, n}$ except $\boldsymbol{z}$. In this paper, we focus on the ReLU function, i.e., $a_i = \mathsf{ReLU}()$. The final activation function, i.e., a_L, can be omitted in the inference stage without compromising the accuracy of predictions. This is because NN predictions rely on the index with the maximum value in its output vector, and since the a_L is monotone increasing, whether or not we apply it will not affect the prediction. The following sections elaborate on how to homomorphically evaluate layers in NNs in the HeFUN framework.

Linear layers (f_i). These layers apply a linear transformation on input. In NNs, there are two common linear transformations, i.e., fully connected layers and convolution layers.

Fully connected layer. The fully connected layer can be considered as the multiplication between an encrypted vector ($[x]$) and a plain weight matrix ($[\boldsymbol{W}]$) and addition with a bias vector ($\boldsymbol{b}$). Based on operators supported by the CKKS scheme (as described in Section 3.2), the fully connected layer can be efficiently evaluated in the ciphertext domain. In particular, $[x] \odot \boldsymbol{W} \oplus \boldsymbol{b} = [\boldsymbol{x} \cdot \boldsymbol{W}] \oplus \boldsymbol{b} = [\boldsymbol{x} \cdot \boldsymbol{W} + \boldsymbol{b}]$.

Convolution layer. A convolutional layer consists of filters that act on input values. Convolutional layers aim to extract features from the given image. Every filter is an $n \times n$ square that moves with a certain stride. By convolving the image pixels, we compute the dot product between filter values and the values adjacent to a particular pixel. Similar to the fully connected layer, the convolution layer can be efficiently evaluated in the ciphertext domain, i.e., $[x] \circledast \boldsymbol{W} \oplus \boldsymbol{b} = [\boldsymbol{x} * \boldsymbol{W} + b]$.

After linear layers, a non-linear activation function (a_i) is usually applied. Unfortunately, the activation function cannot be evaluated in the ciphertext domain. The next section presents our method to address this problem.

4.2. HeReLU: *Homomorphic* ReLU *Evaluation Protocol*

After a linear layer, a non-linear activation function is applied to introduce non-linearity into the NN and allow it to capture complex patterns and relationships in the data. In this paper, we focus on the ReLU function.

$$\mathsf{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Evaluating ReLU requires the comparison operator, which is not natively supported by existing HE schemes. Hence, the ReLU activation function layer cannot be directly computed in the ciphertext domain. To address this issue, we propose a protocol, HeReLU, in which $\mathcal{S}$ interacts with $\mathcal{C}$ to perform ReLU computation.

Homomorphic ReLU evaluation problem. $\mathcal{C}$ holds secret key sk, the server has public key pk but can not access to sk. $\mathcal{S}$ holds ciphertext $[\boldsymbol{x}]$ ($\boldsymbol{x} \in \mathbb{R}^n$) encrypted under sk. $\mathcal{S}$ obtains $[\mathsf{ReLU}(\boldsymbol{x})]$ without leaking $\boldsymbol{x}$ to $\mathcal{C}$ and $\mathcal{S}$.

We now present our protocol that solves the homomorphic ReLU evaluation problem described above. Here, $\mathsf{ReLU}(\boldsymbol{x})$ means applying the ReLU function on each element of $\boldsymbol{x}$, i.e., $\mathsf{ReLU}(\boldsymbol{x}) = (\mathsf{ReLU}(\boldsymbol{x}_1), \ldots, \mathsf{ReLU}(\boldsymbol{x}_n))$. Our protocol (HeReLU) is described in Protocol 1. We now give the intuition behind the protocol and then present the proof for the correctness and security of the protocol (as illustrated in Lemma 1).

Intuition. We based this on the fact that evaluating the ReLU function is essentially equivalent to evaluating the sign function. In particular, given $\mathsf{sign}(x)$, $\mathsf{ReLU}(x)$ can be implemented as follows:

$$\mathsf{ReLU}(x) = \frac{1}{2}(x + x.\mathsf{sign}(x)) \tag{2}$$

where $\mathsf{sign}(x) = 1$ if $x > 0$, $= 0$ if $x = 0$, and $= -1$ if $x < 0$. It is noteworthy that the sign function as defined in this paper is consistent with the standard interpretation found in the literature [9,10,63,64]. This sign is widely employed in various real-world applications, including machine learning algorithms, such as support vector machines [65], cluster analysis [66], and gradient boosting [67].

Therefore, the fundamental question now is as follows: *How can $\mathcal{S}$ (holds $[\boldsymbol{x}]$) securely evaluate* $[\mathsf{sign}(\boldsymbol{x})]$*?* We start from a trivial (non-secure) solution to evaluate $[\mathsf{sign}(\boldsymbol{x})]$ as follows: $\mathcal{S}$ sends $[\boldsymbol{x}]$ to $\mathcal{C}$; $\mathcal{C}$ decrypts the ciphertext (using sk) and obtains $\boldsymbol{x}$, then obtains $sign(\boldsymbol{x})$; $\mathcal{C}$ encrypts the result and sends the ciphertext $[\mathsf{sign}(\boldsymbol{x})]$ to $\mathcal{S}$. Obviously, this trivial solution reveals the entire $\boldsymbol{x}$ to $\mathcal{C}$. To hide $\boldsymbol{x}$ from $\mathcal{C}$, $\mathcal{S}$ first homomorphically multiplies $[\boldsymbol{x}]$ with a random vector $\boldsymbol{r} \neq \boldsymbol{0}$ (line 2), then sends $[\boldsymbol{x} \times \boldsymbol{r}]$ to $\mathcal{C}$ (line 3). After that, $\mathcal{C}$ can decrypt $[\boldsymbol{x} \times \boldsymbol{r}]$ and obtain $\boldsymbol{x} \times \boldsymbol{r}$, which hides $\boldsymbol{x}$ from $\mathcal{C}$ in an information-theoretic way (it is a one-time pad). At $\mathcal{S}$'s side, based on the sign of $\boldsymbol{x} \times \boldsymbol{r}$ and sign of $\boldsymbol{r}$, this allows $\mathcal{S}$ to obtain the sign of $\boldsymbol{x}$. In particular, as $\boldsymbol{r} \neq \boldsymbol{0}$, $\mathsf{sign}(\boldsymbol{x}) = \mathsf{sign}(\boldsymbol{x} \times \boldsymbol{r}) \times \mathsf{sign}(\boldsymbol{r})$ (illustrated in Table 2). Using $[\mathsf{sign}(\boldsymbol{x})]$, $\mathcal{C}$ can homomorphically compute $[\mathsf{ReLU}(\boldsymbol{h})]$ based on Formula (2). The correctness and security of Protocol 1 is shown in Lemma 1.

Protocol 1 HeReLU: Homomorphic ReLU evaluation protocol

```
Input $\mathcal{C}$: Secret key $sk$
Input $\mathcal{S}$: Public key $pk$, ciphertext $[\boldsymbol{x}]$
Output $\mathcal{S}$: $[\mathsf{ReLU}(\boldsymbol{x})]$
    $\mathcal{S}$ runs following steps:
 1: Picks $\boldsymbol{r} \leftarrow \mathbb{R}^n/\{\boldsymbol{0}\}$
 2: $[\boldsymbol{x} \times \boldsymbol{r}] \leftarrow [\boldsymbol{x}] \otimes \boldsymbol{r}$
 3: Sends to $\mathcal{C}$: $[\boldsymbol{x} \times \boldsymbol{r}]$
    $\mathcal{C}$ runs following steps:
 4: $\boldsymbol{x} \times \boldsymbol{r} \leftarrow Decrypt([\boldsymbol{x} \times \boldsymbol{r}], sk)$
 5: Sends to $\mathcal{S}$: $[\mathsf{sign}(\boldsymbol{x} \times \boldsymbol{r})]$
    $\mathcal{S}$ runs following steps:
 6: $[\mathsf{sign}(\boldsymbol{x})] = [\mathsf{sign}(\boldsymbol{x} \times \boldsymbol{r})] \otimes \mathsf{sign}(\boldsymbol{r})$
 7: $[\mathsf{ReLU}(\boldsymbol{x})] = \frac{1}{2} \otimes ([\boldsymbol{x}] \oplus [\boldsymbol{x}] \otimes [\mathsf{sign}(\boldsymbol{x})])$
```

Lemma 1. HeReLU *protocol is secure in the semi-honest model.*

Table 2. Logic table for $\mathsf{sign}()$ function.

$\mathsf{sign}(r)$	$\mathsf{sign}(x \times r)$	$\mathsf{sign}(x)$
1	1	1
1	0	0
1	−1	−1
−1	1	−1
−1	0	0
−1	−1	1

Proof. We prove that the security of our protocol follows Definition 1. We first prove the correctness and then analyze the security of the protocol.

Correctness. The correctness of the protocol is straightforward. We first prove the correctness of computing $[\mathsf{sign}(\boldsymbol{x})]$. At line 6, we have:

$[\mathsf{sign}(\boldsymbol{x} \times \boldsymbol{r})] \otimes \mathsf{sign}(\boldsymbol{r}) = [\mathsf{sign}(\boldsymbol{x} \times \boldsymbol{r}) \times \mathsf{sign}(\boldsymbol{r})] = [\mathsf{sign}(\boldsymbol{x})]$.

Now, we prove the correctness of computing $[\mathsf{ReLU}(\boldsymbol{x})]$. At line 7, we have:

$\frac{1}{2} \otimes ([\boldsymbol{x}] \oplus [\boldsymbol{x}] \otimes [\mathsf{sign}(\boldsymbol{x})]) = \left[\frac{1}{2}(\boldsymbol{x} + \boldsymbol{x} \times \mathsf{sign}(\boldsymbol{x}))\right] = [\mathsf{ReLU}(\boldsymbol{x})]$ (Formula (2)).

Security. We now consider two cases as following Definition 1, i.e., corrupted $\mathcal{C}$ and corrupted $\mathcal{S}$.

Corrupted $\mathcal{C}$: $\mathcal{C}$'s view is $\mathsf{View}_{\mathcal{C}} = (sk, [\boldsymbol{x} \times \boldsymbol{r}])$. Given (sk), we build a simulator $\mathsf{Sim}_{\mathcal{C}}$ as below:

1. Pick $\boldsymbol{x}' \leftarrow \mathbb{R}^n$;
2. Output $(sk, [\boldsymbol{x}'])$.

Even $\mathcal{C}$ can decrypt $[\boldsymbol{x} \times \boldsymbol{r}]$ and obtain $\boldsymbol{x} \times \boldsymbol{r}$, because $\boldsymbol{r}$ was randomly chosen by $\mathcal{S}$; hence, from the view of $\mathcal{C}$: $\boldsymbol{x} \times \boldsymbol{r} \approx_c \boldsymbol{x}' \Leftrightarrow (sk, [\boldsymbol{x} \times \boldsymbol{r}]) \approx_c (sk, [\boldsymbol{x}']) \Leftrightarrow \mathsf{View}_{\mathcal{C}} \approx_c \mathsf{Sim}_{\mathcal{C}}$.

Corrupted $\mathcal{S}$: $\mathcal{S}$'s view is $\mathsf{View}_{\mathcal{S}} = (pk, [\boldsymbol{x}], [\mathsf{sign}(\boldsymbol{x} \times \boldsymbol{r})])$. Given $(pk, [\boldsymbol{x}], [\mathsf{ReLU}(\boldsymbol{x})])$, we build a simulator $\mathsf{Sim}_{\mathcal{S}}$ as below:

1. Pick $\boldsymbol{s} \leftarrow \{-1, 0, 1\}^n$;

2. Output $(pk, [\boldsymbol{x}], [\boldsymbol{s}])$.

Based on the semantic security of HE schemes, it is obvious that $(pk, [\boldsymbol{x}], [\mathsf{sign}(\boldsymbol{x} \times \boldsymbol{r})]) \approx_c (pk, [\boldsymbol{x}], [\boldsymbol{s}]) \Leftrightarrow \mathsf{View}_{\mathcal{S}} \approx_c \mathsf{Sim}_{\mathcal{S}}$. $\square$

The idea of computing the ReLU function is based on Equation (2), i.e., $\mathrm{ReLU}(x) = \frac{1}{2}(x + x.\mathsf{sign}(x))$. This equation requires a division by 2, which is equivalent to multiplying by 0.5 in CKKS. However, the division is not supported by almost all other schemes that work on integers in the ring $\mathbf{Z}_n$, such as BFV [23] and BGV [68]. In those schemes, the division by 2 can be replaced by multiplying by 2^{-1}, where 2^{-1} is the inverse of 2 in $\mathbb{Z}_n$. By doing so, our protocol can be compatible with both CKKS and other integer HE schemes.

HeReLU allows $\mathcal{S}$ to obtain $[\mathrm{ReLU}(\boldsymbol{x})]$ without leaking $\boldsymbol{x}$ to $\mathcal{S}$ and C. However, it leaks some unexpected information about $\boldsymbol{x}$ to $\mathcal{C}$, namely, it leaks whether the ith slot in $\boldsymbol{x}$ is zero or not (we denote it by $\boldsymbol{x} \overset{?}{=} 0$). This leakage occurs at line 4 in Protocol 1. In particular, $\mathcal{C}$ knows a slot $\boldsymbol{x}_i = 0$ if the ith slot in $\boldsymbol{x} \times \boldsymbol{r}$ is 0 (as $\boldsymbol{r} \neq 0$). We address the information leakage by proposing a novel NN permutation. We detail this in Section 4.5.

4.3. HeRefresh: *Refreshing Ciphertexts*

LHE schemes limit the depth of the circuit that can be evaluated. Furthermore, in the case of deep NNs, the multiplicative depth of the HE scheme must be a big value that causes computation overhead (as mentioned in Section 3.2). To address these issues, we design a simple but efficient protocol to regularly refresh intermediate layers in NNs, HeRefresh. In this section, we use $[\boldsymbol{x}, \ell]$ to denote the ciphertext $[\boldsymbol{x}]$ is at the level ℓ, i.e., ℓ multiplications can be performed on this ciphertext).

Ciphertext refreshing problem. $\mathcal{C}$ holds secret key sk, the server has public key pk but can not access to sk. $\mathcal{S}$ holds ciphertext $[\boldsymbol{x}, \ell]$ ($\boldsymbol{x} \in \mathbb{R}^n$) encrypted under sk. $\mathcal{S}$ obtains $[\boldsymbol{x}, L]$, s.t. $L > \ell$, without leaking $\boldsymbol{x}$ to $\mathcal{C}$ and $\mathcal{S}$.

We now present our protocol that solves the ciphertext refreshing problem described above. The protocol is shown in Protocol 2. We now give the intuition behind the protocol and then present the proof for the correctness and security of the protocol (as illustrated in Lemma 2.

Protocol 2 HeRefresh: Ciphertext refreshing protocol

Input $\mathcal{C}$**:** Secret key sk
Input $\mathcal{S}$**:** Public key pk, $[\boldsymbol{x}, \ell]$
Output $\mathcal{S}$**:** $[\boldsymbol{x}, L]$

```
   $\mathcal{S}$ runs following steps:
1: Picks $\boldsymbol{r} \leftarrow \mathbb{R}^n$
2: $[\boldsymbol{x}+\boldsymbol{r}, \ell] \leftarrow [\boldsymbol{x}, \ell] \oplus \boldsymbol{r}$
3: Sends to $\mathcal{C}$: $[\boldsymbol{x}+\boldsymbol{r}, \ell]$
   $\mathcal{C}$ runs following steps:
4: $\boldsymbol{x}+\boldsymbol{r} \leftarrow Decrypt([\boldsymbol{x}+\boldsymbol{r}, \ell], sk)$
5: Sends to $\mathcal{S}$: $[\boldsymbol{x}+\boldsymbol{r}, L]$
   $\mathcal{S}$ runs following steps:
6: $[\boldsymbol{x}, L] = [\boldsymbol{x}+\boldsymbol{r}, L] \ominus \boldsymbol{r}$
```

Lemma 2. HeRefresh *protocol is secure in the semi-honest model.*

Intuition. The idea of the protocol is as follows. First, $\mathcal{S}$ additively blinds $[\boldsymbol{x}]$ with a random mask $\boldsymbol{r}$ (line 2) and sends the masked ciphertext $[\boldsymbol{x}+\boldsymbol{r}]$ to $\mathcal{C}$ (line 3). Then, $\mathcal{C}$ decrypts the ciphertext and obtains masked message $\boldsymbol{x}+\boldsymbol{r}$, which perfectly hides $\boldsymbol{x}$ from $\mathcal{C}$ in an information-theoretic way (it is a one-time pad). Then, $\mathcal{C}$ encrypts the masked message the $[\boldsymbol{x}+\boldsymbol{r}]$. This ciphertext is at the highest level L, as it has not undergone any multiplication. Obviously, the encryption–decryption procedure can be considered as the ciphertext refresher. Finally, $\mathcal{S}$ homomorphically subtracts $\boldsymbol{r}$ and obtains a new ciphertext, which is an L-level ciphertext. HeRefresh is essential to enable continued computation without increasing the encryption parameters. Rather than selecting encryption parameters large enough to support the entire NN, we must now only be large enough to support the linear layers and computation in the HeReLU protocol.

Proof. We first prove the correctness and then analyze the security of the protocol based on Definition 1.

Correctness. The correctness of the protocol is straightforward. We first consider the level of the ciphertext. At line 5, $\mathcal{C}$ encrypts the plain message $\boldsymbol{x} + \boldsymbol{r}$ into $[\boldsymbol{x} + \boldsymbol{r}]$. This *new* ciphertext has not undergone any computation; hence, it is at the highest level of the HE scheme, i.e., L. In line 6, the ciphertext involves one homomorphic subtraction; hence, the resulting ciphertext remains at the level L.

From now on, for the sake of simplicity, we remove the level notion in ciphertexts. At line 6 of the protocol, $[\boldsymbol{x} + \boldsymbol{r}] \ominus \boldsymbol{r} = [\boldsymbol{x} + \boldsymbol{r} - \boldsymbol{r}] = [\boldsymbol{x}]$.

Security. We now consider two cases as following Definition 1, i.e., corrupted $\mathcal{C}$ and corrupted $\mathcal{S}$.

Corrupted $\mathcal{C}$: $\mathcal{C}$'s view is $\mathsf{View}_{\mathcal{C}} = (sk, [\boldsymbol{h} \times \boldsymbol{r}])$. Given (sk), we build a simulator $\mathsf{Sim}_{\mathcal{C}}$ as below:

1. Pick $\boldsymbol{h'} \leftarrow \mathbb{R}^n$;
2. Output $(sk, [\boldsymbol{h'}])$.

Even $\mathcal{C}$ can decrypt $[\boldsymbol{h} \times \boldsymbol{r}]$, and obtain $\boldsymbol{h} \times \boldsymbol{r}$, because $\boldsymbol{r}$ was randomly chosen by $\mathcal{S}$; hence, from the view of $\mathcal{C}$: $\boldsymbol{h} \times \boldsymbol{r} \approx_c \boldsymbol{h'} \Leftrightarrow (sk, [\boldsymbol{h} \times \boldsymbol{r}]) \approx_c (sk, [\boldsymbol{h'}]) \Leftrightarrow \mathsf{View}_{\mathcal{C}} \approx_c \mathsf{Sim}_{\mathcal{C}}$.

Corrupted $\mathcal{S}$: $\mathcal{S}$'s view is $\mathsf{View}_{\mathcal{S}} = (pk, [\boldsymbol{h}], [\mathsf{sign}(\boldsymbol{h} \times \boldsymbol{r})])$. Given $(pk, [\boldsymbol{h}], [\mathsf{ReLU}(\boldsymbol{h})])$, we build a simulator $\mathsf{Sim}_{\mathcal{S}}$ as below:

1. Pick $\boldsymbol{s} \leftarrow \{-1, 0, 1\}^n$;
2. Output $(pk, [\boldsymbol{h}], [\boldsymbol{s}])$.

Based on the semantic security of HE schemes, it is obvious that $(pk, [\boldsymbol{h}], [\mathsf{sign}(\boldsymbol{h} \times \boldsymbol{r})]) \approx_c (pk, [\boldsymbol{h}], [\boldsymbol{s}]) \Leftrightarrow \mathsf{View}_{\mathcal{S}} \approx_c \mathsf{Sim}_{\mathcal{S}}$. □

We now leverage HeReLU and HeRefresh to design an SNNI framework. We start from a simple framework, $\mathsf{HeFUN}_{\mathsf{naive}}$, which straightforwardly applies HeReLU in SNNI (Section 4.4). $\mathsf{HeFUN}_{\mathsf{naive}}$ *almost* satisfies the requirements of SNNI, except that it leaks $\boldsymbol{m} \overset{?}{=} 0$ to $\mathcal{C}$. We then improve to $\mathsf{HeFUN}_{\mathsf{naive}}$ to prevent such information leakage, which results in the HeFUN protocol (Section 4.5).

4.4. $\mathsf{HeFUN}_{\mathsf{naive}}$ *Protocol*

This section presents our method to solve the SNNI problem, which uses HE to evaluate linear layers (as described in Section 4.1), uses the HeReLU protocol to evaluate the ReLU activation function, and uses the HeRefresh protocol to refresh intermediate layers. Interestingly, HeRefresh and HeReLU can be run in parallel on the same input $\boldsymbol{x}^{(i)}$. By doing so, we preserve communication rounds. The protocol is shown in Protocol 3. HeFUN_{naive} straightforwardly applies the HeReLU protocol to compute the ReLU activation function. As shown in Section 4.2, it leaks $\boldsymbol{x}^{(i)} \overset{?}{=} 0$ to $\mathcal{C}$ when $\mathcal{C}$ and $\mathcal{S}$ run HeReLU (line 4). In the next section, we present a *mature* version of HeFUN_{naive}, i.e., HeFUN, which overcomes such data leakage.

Protocol 3 HeFUN$_{naive}$ framework

Input $\mathcal{C}$**:** Data $\boldsymbol{x}$, secret key sk
Input $\mathcal{S}$**:** public key pk, trained weights and biases $(\boldsymbol{W}_i, \boldsymbol{b}_i)$ with $i = \overline{1,n}$
Output $\mathcal{C}$**:** $z := f_L(\boldsymbol{W}_L, a_{L-1}(f_{L-1}(\dots a_1(f_1(\boldsymbol{W}_1, \boldsymbol{x}, \boldsymbol{b}_1)))), \boldsymbol{b}_L)$, where f_i is ReLU function.

```
1: $\mathcal{C}$ encrypts data $\boldsymbol{x}$ and send $[\boldsymbol{x}]$ to server
2: $\mathcal{S}$ computes $\left[\boldsymbol{x}^{(1)}\right] = f_1([\boldsymbol{x}], \boldsymbol{W}^{(1)}, \boldsymbol{b}^{(1)})$
3: for $i = 1$ to $L-1$ do
4:     $\mathcal{S}$ obtains refreshed $\left[\boldsymbol{x}^{(i)}\right]$        ▷ $\mathcal{C}$ and $\mathcal{S}$ run Protocol 2 on $\left[\boldsymbol{x}^{(i)}\right]$
5:     $\mathcal{S}$ obtains $\left[\boldsymbol{h}^{(i)}\right] = \left[\text{ReLU}(\boldsymbol{x}^{(i)})\right]$        ▷ $\mathcal{C}$ and $\mathcal{S}$ run Protocol 1 on $\left[\boldsymbol{x}^{(i)}\right]$
6:     $\mathcal{S}$ compute $\left[\boldsymbol{x}^{(i+1)}\right] = f_{i+1}(\left[\boldsymbol{h}^{(i)}\right], \boldsymbol{W}^{(i+1)}, \boldsymbol{b}^{(i+1)})$
7: $\mathcal{S}$ sends $\left[\boldsymbol{x}^{(L)}\right]$ to the $\mathcal{C}$
8: $\mathcal{C}$ obtains $\boldsymbol{z} \leftarrow Decrypt(\left[\boldsymbol{x}^{(L)}\right], sk)$
```

4.5. HeFUN

We first present the intuition behind how can we hide $\boldsymbol{x}^{(i)} \overset{?}{=} 0$ from $\mathcal{C}$.

Intuition. To tackle the issues of leaking $\boldsymbol{x}^{(i)} \overset{?}{=} 0$ to $\mathcal{C}$, $\mathcal{S}$ implements a random permutation on $\boldsymbol{x}^{(i)}$ prior to executing the HeReLU protocol. Then, $\boldsymbol{x}^{(i)}$ cannot determine the original position of a value, concealing whether the actual value of a particular slot is zero. Thus, $\mathcal{C}$ only learns the number of zero values, which can be obscured by adding dummy elements. However, in SNNI, $\boldsymbol{x}^{(i)}$ is in encrypted form, i.e., $\left[\boldsymbol{x}^{(i)}\right]$. Unfortunately, permutation on $\left[\boldsymbol{x}^{(i)}\right]$ is a challenging task. This is because the permutation requires swapping slots in the ciphertext, which requires numerous homomorphic rotations—a costly operator [11,15].

This approach is not practical when the number of neurons in NN layers is large. The section below describes how we can overcome this challenge.

Given a vector $\boldsymbol{x} \in \mathbb{R}^n$, and π is a permutation over $\{1, \dots, n\}$, let $\pi(\boldsymbol{x})$ denote the permutation $\boldsymbol{x}$'s slots according to π, i.e., the ith slot in the permuted vector $(\pi(\boldsymbol{x}))$ is $\boldsymbol{x}_{\pi(i)}$ (instead of $\boldsymbol{x}_i$). Given a matrix $\boldsymbol{W}$, we use $\pi(\boldsymbol{W})$ to denote permutation of $\boldsymbol{W}$ by columns, i.e., the ith column in the permuted matrix $(\pi(\boldsymbol{W}))$ is $\boldsymbol{W}_{\pi(i)}$ (instead of $\boldsymbol{W}_i$). Here are some obvious facts:

- Permuting two vectors with the same permutation preserves element-wise addition:

$$\pi(\boldsymbol{x}) + \pi(\boldsymbol{y}) = \pi(\boldsymbol{x} + \boldsymbol{y}) \tag{3}$$

- Permute two vectors with the same permutation and preserve their dot product:

$$\pi(\boldsymbol{x}) \cdot \pi(\boldsymbol{y}) = \boldsymbol{x} \cdot \boldsymbol{y} \tag{4}$$

- Permute a vector, and every column in a matrix with the same permutation preserves the vector–matrix product:

$$\pi(\boldsymbol{x}).\pi(\boldsymbol{W}, col) = \boldsymbol{x}.\boldsymbol{W} \tag{5}$$

  where $\pi(\boldsymbol{W}, col)$ denotes that we apply the permutation π to every column of $\boldsymbol{W}$.
- In vector–matrix multiplication, (only) permutation of a matrix by column leads to the same permutation on the result.

$$\boldsymbol{x}.\pi(\boldsymbol{W}) = \pi(\boldsymbol{x}.\boldsymbol{W}) \tag{6}$$

Proof. Proof of Property (3): $\pi(\boldsymbol{x}) + \pi(\boldsymbol{y}) = (\boldsymbol{x}_{\pi(1)} + \boldsymbol{y}_{\pi(n)}, \dots, \boldsymbol{x}_{\pi(n)} + \boldsymbol{y}_{\pi(n)}) = \pi(\boldsymbol{x} + \boldsymbol{y})$

Proof of Property (4):

$$\begin{aligned}\pi(\boldsymbol{x})\cdot\pi(\boldsymbol{y}) &= \sum_{i=1}^{n}\boldsymbol{x}_{\pi(i)}\cdot\boldsymbol{y}_{\pi(i)}\\ &= \sum_{i=1}^{n}\boldsymbol{x}_i\cdot\boldsymbol{y}_i\\ &= \boldsymbol{x}\cdot\boldsymbol{y}\end{aligned}$$

Proof of Property (5):

$$\pi(\boldsymbol{x}).\pi(\boldsymbol{W},col) = [\text{—}\ \pi(\boldsymbol{x})\ \text{—}]\cdot\begin{bmatrix} | & & | \\ \pi(\boldsymbol{W}_1) & \vdots & \pi(\boldsymbol{W}_n) \\ | & & | \end{bmatrix}$$

$$\begin{aligned}&= (\pi(\boldsymbol{x})\cdot\pi(\boldsymbol{W}_1),\ldots,\pi\boldsymbol{x}\cdot\pi(\boldsymbol{W}_n))\\ &= (\boldsymbol{x}\cdot\boldsymbol{W}_1,\ldots,\boldsymbol{x}\cdot\boldsymbol{W}_n) \qquad \text{(Property (4))}\\ &= \boldsymbol{x}\cdot\boldsymbol{W}\end{aligned}$$

Proof of Property (6):

$$\begin{aligned}\boldsymbol{x}\cdot\pi(\boldsymbol{W}) &= \boldsymbol{x}\left(\boldsymbol{W}^T_{\pi(1)},\ldots,\boldsymbol{W}^T_{\pi(n)}\right)\\ &= \left(\boldsymbol{x}\cdot\boldsymbol{W}^T_{\pi(1)},\ldots,\boldsymbol{x}\cdot\boldsymbol{W}^T_{\pi(n)}\right)\\ &= \pi\left(\boldsymbol{x}\cdot\boldsymbol{W}^T_1,\ldots,\boldsymbol{x}\cdot\boldsymbol{W}^T_n\right)\\ &= \pi(\boldsymbol{x}\cdot\boldsymbol{W})\end{aligned}$$

□

Based on Property (6) and the fact that $\boldsymbol{x} = \boldsymbol{x}\cdot\boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix, we can achieve permutation of $\boldsymbol{x}$ by just permuting identity matrix $\boldsymbol{I}$, namely $\boldsymbol{x}\cdot\pi(\boldsymbol{I}) = \pi(\boldsymbol{x}\cdot\boldsymbol{I}) = \pi(\boldsymbol{x})$. Based on this observation, we propose a novel algorithm, i.e., HePerm, to permute slots inside a ciphertext without the rotation operator. The algorithm is shown in Protocol 4. The correctness of the protocol is straightforward, as below.

$$\begin{aligned}[\boldsymbol{x}]\odot\pi(\boldsymbol{I}) &= [\boldsymbol{x}\cdot\pi(\boldsymbol{I})] \qquad \text{(Homomorphic property)}\\ &= [\pi(\boldsymbol{x}\cdot\boldsymbol{I})] \qquad \text{(Property (6))}\\ &= [\pi(\boldsymbol{x})]\end{aligned}$$

Protocol 4 HePerm: Homomorphic permutation

Input: ciphertext $[\boldsymbol{x}]$, permutation π
Output: $[\pi(\boldsymbol{x})]$

```
1: $[\pi(\boldsymbol{x})] = [\boldsymbol{x}] \odot \pi(\boldsymbol{I})$
2: return $\boldsymbol{x}$
```

The HePerm algorithm allows us to permute any (encrypted) intermediate layers, i.e., $\left[\boldsymbol{x}^{(i)}\right]$, prior to executing the HeReLU protocol. Hence, we can hide $\boldsymbol{x}^{(i)} \overset{?}{=} 0$ from $\mathcal{C}$. Suppose each intermediate layer $\left[\boldsymbol{x}^{(i)}\right]$ (before the ReLU layer) is permuted by a permutation $\pi^{(i)}$. Then, the value obtained by $\mathcal{S}$ (after running HeReLU) is $\left[\pi^{i}(\boldsymbol{h}^{(i)})\right]$ (not $\left[\boldsymbol{h}^{(i)}\right]$). This value will be the input for the next linear layer. Therefore, to compute the next linear layer, $\mathcal{S}$ first needs to *unpermute* $\left[\pi^{i}(\boldsymbol{h}^{(i)})\right]$. To do so, $\mathcal{S}$ runs HePerm on $\left[\pi^{i}(\boldsymbol{h}^{(i)})\right]$ and ${\pi^{(i)}}^{-1}$, which results in $\left[{\pi^{(i)}}^{-1}(\pi^{(i)}(\boldsymbol{h}^{(i)}))\right] = \left[\boldsymbol{h}^{(i)}\right]$. Then, $\mathcal{S}$ can homomorphically evaluate the next linear layer on $\left[\boldsymbol{h}^{(i)}\right]$, which results in $\left[\boldsymbol{x}^{(i+1)}\right]$. Finally, $\mathcal{S}$ can run HePerm to permute $\left[\boldsymbol{x}^{(i+1)}\right]$ using permutation $\pi^{(i+1)}$. Interestingly, for a fully connected layer, we can achieve permutation without the need of HePerm. This is because the fully connected layer can be homomorphically evaluated directly on $\left[\pi^{(i)}(\boldsymbol{h}^{(i)})\right]$ by permuting each column in $\boldsymbol{W}^{(i+1)}$ by $\pi^{(i)}$ (based on Property (5)), namely $\left[\pi^{(i)}(\boldsymbol{h}^{(i)})\right] \odot \pi^{(i)}(\boldsymbol{W}^{(i+1)}, col) \oplus \boldsymbol{b}^{(i+1)} = \left[\pi^{(i)}(\boldsymbol{h}^{(i)}) \cdot \pi^{(i)}(\boldsymbol{W}^{(i+1)}, col) + \boldsymbol{b}^{(i+1)}\right] = \left[\boldsymbol{h}^{(i)} \cdot \boldsymbol{W}^{(i+1)} + \boldsymbol{b}^{(i+1)}\right]$. Therefore, we achieve permutation *for free* in fully connected layers.

Assume that $\mathcal{S}$ chooses L random permutations $\pi^{(1)}, \ldots, \pi^{(L)}$ corresponding to layer 1 to layer L, respectively. Notably, $\pi^{(1)}, \ldots, \pi^{(L)}$ can be chosen offline by $\mathcal{S}$ before the inference process. Hence, we can save the inference time. The details of HeFUN are shown in Protocol 5, in which *FC* means fully connected layer and *Conv* means convolution layer. HeFUN's security follows Lemma 3. We now provide an analysis of the correctness and security of the protocol.

Protocol 5 HeFUN framework

```
Input C: Data $\boldsymbol{x}$, secret key $sk$
Input S: public key $pk$, trained weights and biases $(\boldsymbol{W}_i, \boldsymbol{b}_i)$ with $i = \overline{1, n}$
Output C: $z = f_L(\boldsymbol{W}_L, f_{L-1}(\ldots f_1(\boldsymbol{W}_1, \boldsymbol{x}, \boldsymbol{b}_1)), \boldsymbol{b}_L)$, where $f_i$ is ReLU function.
 1: $\mathcal{C}$ encrypts data $\boldsymbol{x}$ and send $[\boldsymbol{x}]$ to server
    $\mathcal{S}$ runs following steps:
 2: $\mathcal{S}$ chose $\pi^{(1)}, \ldots, \pi^{(L)}$                          ▷ Can be done offline
 3: if $f_1 = FC$ then
 4:     $\left[\pi^{(1)}(\boldsymbol{x}^{(1)})\right] = [\boldsymbol{x}] \odot \pi^{(1)}(\boldsymbol{W}^{(1)}) \oplus \pi^{(1)}(\boldsymbol{b}^{(1)})$
 5: else if $f_1 = Conv$ then
 6:     $\left[\boldsymbol{x}^{(1)}\right] = [\boldsymbol{x}] \circledast \boldsymbol{W}^{(1)} \oplus \boldsymbol{b}^{(1)}$
 7:     $\left[\pi^{(1)}(\boldsymbol{x}^{(1)})\right] \leftarrow \mathsf{HePerm}(\left[\boldsymbol{x}^{(1)}\right], \pi^{(1)})$
 8: for $i = 1$ to $L-1$ do
 9:     $\mathcal{S}$ obtains refreshed $\left[\pi^{(i)}(\boldsymbol{x}^{(i)})\right]$   ▷ $\mathcal{C}$ and $\mathcal{S}$ run Protocol 2 on $\left[\pi^{(i)}(\boldsymbol{x}^{(i)})\right]$
10:     $\mathcal{S}$ obtains $\left[\pi^{(i)}(\boldsymbol{h}^{(i)})\right] = \left[\mathsf{ReLU}(\pi^{(i)}(\boldsymbol{x}^{(i)}))\right]$   ▷ $\mathcal{C}$ and $\mathcal{S}$ run Protocol 1 on
    $\left[\pi^{(i)}(\boldsymbol{x}^{(i)})\right]$
11:     if $f_{i+1} = FC$ then
12:         $\left[\pi^{(i+1)}(\boldsymbol{x}^{(i+1)})\right] = \left[\pi^{(i)}(\boldsymbol{h}^{(i)})\right] \odot \pi^{(i+1)}(\pi^{(i)}(\boldsymbol{W}^{(i+1)}, col)) \oplus \pi^{(i+1)}(\boldsymbol{b}^{(i+1)})$
13:     else if $f_{i+1} = Conv$ then
14:         $\left[\boldsymbol{h}^{(i)}\right] \leftarrow \mathsf{HePerm}(\left[\pi^{(i)}(\boldsymbol{h}^{(i)})\right], {\pi^{(i)}}^{-1})$
15:         $\left[\boldsymbol{x}^{(i+1)}\right] = \left[\boldsymbol{h}^{(i)}\right] \circledast \boldsymbol{W}^{(i+1)} \oplus \boldsymbol{b}^{(i+1)}$
16:         $\left[\pi^{(i+1)}(\boldsymbol{x}^{(i+1)})\right] \leftarrow \mathsf{HePerm}(\left[\boldsymbol{x}^{(i+1)}\right], \pi^{(i+1)})$
17: $\mathcal{S}$ sends $\pi^{(L)}$ and $\left[\pi^{(L)}(\boldsymbol{x}^{(L)})\right]$ to the $\mathcal{C}$
    $\mathcal{C}$ runs following steps:
18: $\pi^{(L)}(\boldsymbol{x}^{(L)}) \leftarrow Decrypt(\left[\pi^{(L)}(\boldsymbol{x}^{(L)})\right], sk)$
19: $\mathcal{C}$ obtains $\boldsymbol{x}^{(L)} = {\pi^{(L)}}^{-1}(\pi^{(L)}(\boldsymbol{x}^{(L)}))$
```

Lemma 3. HeFUN *protocol is secure in the semi-honest model.*

Proof. Correctness. We first consider the fully connected layers. At line 12, we have:

$$\begin{aligned}
&\left[\pi^{(i)}(\boldsymbol{h}^{(i)})\right] \odot \pi^{(i+1)}(\pi^{(i)}(\boldsymbol{W}^{(i+1)}, col)) \oplus \pi^{(i+1)}(\boldsymbol{b}^{(i+1)}) \\
&= \left[\pi^{(i)}(\boldsymbol{h}^{(i)}) \cdot \pi^{(i+1)}(\pi^{(i)}(\boldsymbol{W}^{(i+1)}, col)) + \pi^{(i+1)}(\boldsymbol{b}^{(i+1)})\right] \qquad \text{(Homomorphic property)} \\
&= \left[\pi^{(i+1)}(\pi^{(i)}(\boldsymbol{h}^{(i)}) \cdot \pi^{(i)}(\boldsymbol{W}^{(i+1)}, col)) + \pi^{(1)}(\boldsymbol{b}^{(1)})\right] \qquad \text{(Property (6))} \\
&= \left[\pi^{(i+1)}(\boldsymbol{h}^{(i)} \cdot \boldsymbol{W}^{(i+1)}) + \pi^{(i+1)}(\boldsymbol{b}^{(i+1)})\right] \qquad \text{(Property (5))} \\
&= \left[\pi^{(i+1)}(\boldsymbol{h}^{(i)} \cdot \boldsymbol{W}^{(i+1)} + \boldsymbol{b}^{(i+1)})\right] \qquad \text{(Property (3))} \\
&= \left[\pi^{(i+1)}(\boldsymbol{x}^{(i+1)})\right]
\end{aligned}$$

We now consider convolution layers, i.e., lines 14, 15, and 16. The correctness of convolution layers is straightforward based on the correctness of the HePerm algorithm and the homomorphic property.

The above process, i.e., computing the linear layer then computing the ReLU activation function, is repeated through layers. At the end, $\mathcal{C}$ obtains $\pi^{(L)}(\boldsymbol{x}^{(L)})$ (line 18). Because $\mathcal{S}$ sends to $\mathcal{C}$ the permutation of the last layer, i.e., $\pi^{(L)}$ (line 17), $\mathcal{C}$ can unpermute $\pi^{(L)}(\boldsymbol{x}^{(L)})$, and obtains $\boldsymbol{x}^{(L)} = \boldsymbol{z}$.

Security. We prove the security using modular sequential composition introduced in [22]. We first construct framework HeFUN′, in which $\mathcal{C}$ and $\mathcal{S}$ call ideal functionalities $\mathcal{F}_{Refresh}$ and $\mathcal{F}_{ReLU}$ to refresh ciphertext and evaluate the ReLU function. We have HeRefresh and HeReLU such that they are already secure in the semi-honest model (according to Lemma 2 and Lemma 1, respectively). Now, we need to prove that HeFUN′ (with the calls to $\mathcal{F}_{Refresh}$ and $\mathcal{F}_{ReLU}$) is secure in the semi-honest model. Then, we can replace the calls to ideal functionalities $\mathcal{F}_{Refresh}$ and $\mathcal{F}_{ReLU}$ by protocols HeRefresh and HeReLU, respectively, and conclude the security of HeFUN by invoking modular sequential composition (as presented in Section 3.3). We now consider two cases as follows Definition 1, i.e., corrupted $\mathcal{C}$ and corrupted S. Considering corrupted $\mathcal{C}$, it is obvious that $\mathcal{C}$ receives nothing from ideal calls $\mathcal{F}_{Refresh}$ and $\mathcal{F}_{ReLU}$. Hence, we just need to consider the case of corrupted $\mathcal{S}$, as below:

Corrupted $\mathcal{S}$: $\mathcal{S}$'s view is

$$\mathsf{View}_{\mathcal{S}} = \left(pk, f_1, \ldots, f_n, \boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(n)}, \boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(n)}; \left[\boldsymbol{x}^{(1)}\right], \ldots, \left[\boldsymbol{x}^{(L)}\right]\right).$$

Given $(pk, f_1, \ldots, f_n, \boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(n)}, \boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(n)})$, we build a simulator $\mathsf{Sim}_{\mathcal{S}}$ as below:

1. Pick $\boldsymbol{s_i} \leftarrow \{-1, 0, 1\}^n$ with $i = \overline{1, n}$;
2. Output $(pk, f_1, \ldots, f_n, W_1, \ldots, W_n, B_1, \ldots, B_n; [\boldsymbol{s_1}], \ldots, [\boldsymbol{s_L}])$.

Given the semantic security guaranteed by the HE scheme, coupled with the entropic uncertainty introduced by our homomorphic permutation algorithm, it naturally follows that $\left[\boldsymbol{x}^{(i)}\right] \approx_c [\boldsymbol{s_i}]$. Hence, $\mathsf{View}_{\mathcal{S}} \approx_c \mathsf{Sim}_{\mathcal{S}}$. □

5. Experiments

In the evaluation of SNNI, we consider three critical criteria: security, accuracy, and efficiency. The comparative analysis outlined in Table 1 leads to several observations. (1) MPC-based frameworks fail to meet the security criterion, as they disclose the entire architecture of NNs; (2) TFHE-based frameworks fall short on accuracy, since they are limited to BNNs; and (3) LHE-based frameworks, HE-MPC hybrid frameworks, and our proposed HeFUN framework appear to fulfill all three criteria (it should be noted that the fulfillment of these requirements is not absolute, and some issues still persist, as outlined in Table 1). Hence, in this section, we compare our proposed approach, HeFUN, with HE-MPC hybrid-based frameworks and LHE-based frameworks. The core architecture of both HeFUN and prevailing HE-MPC hybrid frameworks encompasses two fundamental components: linear layer evaluation and the evaluation of the non-linear function, specifically the ReLU function. While the linear layer evaluation in these frameworks uniformly employs homomorphic encryption, the difference arises in the treatment of non-linear layer evaluation, namely, HeFUN implements HeReLU, whereas HE-MPC hybrid-based frameworks opt for either the GC or ABY protocol. This section delves into the communication complexity inherent in the ReLU evaluation within HeFUN as compared to that within existing HE-MPC hybrid frameworks (Section 5.1). HeReLU serves as a potential alternative to the GC/ABY protocols in HE-MPC hybrid-based frameworks, which reduces the communication complexity in SNNI. To compare HeFUN with LHE-based frameworks, we first compare the ciphertext refreshing component. In particular, we compare HeRefresh with the bootstrapping technique in the existing HE schemes (Section 5.2). Subsequently, we benchmark HeFUN alongside the prevalent LHE-based frameworks using a real-world dataset (Section 5.3). All experiments are run on a PC with a single Intel i9-13900K running at 5.80 GHz and 64 GB of RAM, running Ubuntu 22.04. Overall, the results presented below show that:

- Compared to current HE-MPC frameworks, HeFUN requires less communication rounds in evaluating the ReLU activation function (Section 5.1).
- Compared to existing bootstrapping procedures, HeRefresh is much faster (Section 5.2).
- Compared to current LHE-based frameworks, HeFUN outperforms the current LHE-based approach in terms of accuracy and inference time (Section 5.3).

5.1. Comparison with Hybrid HE-MPC Approach

The current state-of-the-art HE-MPC hybrid-based approach is Gazelle [15]. Within Gazelle's framework, the server directly computes linear layers using HE operators of the LHE scheme in an offline phase. To evaluate non-linear layers, Gazelle leverages GC [19] to handle the bitwise operations required by ReLU (online phase). Finally, because each layer in an NN consists of alternating linear and non-linear layers, Gazelle also elaborates on an efficient method to switch between the two aforementioned primitives using a novel technique based on additive secret sharing. Gazelle's primary bottleneck is the cost of evaluating GC for the ReLU activation function [15,69]. Compared to Gazelle, HeFUN can evaluate the ReLU function *solely* based on an HE scheme, hence avoiding the expensive GC protocol that significantly reduces communication rounds and communication cost. We now compare the procedure of evaluating the ReLU function between HeFUN and Gazelle regarding communication rounds and communication cost. In both frameworks, i.e., HeFUN and Gazelle, after computing the linear layers homomorphically or receiving the client's encrypted input, the server holds an encrypted vector $[\boldsymbol{x}]$. The objective for both HeFUN and Gazelle is to enable the server to obtain $[\mathsf{ReLU}(\boldsymbol{x})]$. In Gazelle, the server and the client perform the following steps:

- *Conversion from HE to MPC:* The server additively blinds $[\boldsymbol{x}]$ with a random mask $\boldsymbol{r}_1$ and sends the masked ciphertext $[\boldsymbol{x}+\boldsymbol{r}_1]$ to the client.
- *MPC circuit evaluation:* The client decrypts $[\boldsymbol{x}+\boldsymbol{r}_1]$. Then, the client (holds $\boldsymbol{x}+\boldsymbol{r}_1$) and the server (holds $\boldsymbol{r}_1$ and a randomness $\boldsymbol{r}_2$) run GC to compute $\mathsf{ReLU}(\boldsymbol{x}+\boldsymbol{r}_1-\boldsymbol{r}_1)+\boldsymbol{r}_2 = \mathsf{ReLU}(\boldsymbol{x})+\boldsymbol{r}_2$ without leaking $\boldsymbol{x}+\boldsymbol{r}_1$ to the server and $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ to the client. Finally, the client sends $[\mathsf{ReLU}(\boldsymbol{x})+\boldsymbol{r}_2]$ to the server.
- *Conversion from MPC to HE:* The server homomorphically subtracts $\boldsymbol{r}_2$ and obtains $[\mathsf{ReLU}(\boldsymbol{x})]$.

The main bottleneck of Gazelle is step 2, i.e., the client and the server run GC, which requires a huge communication [15]. The comparison shown in Figure 1 illustrates that HeFUN surpasses Gazelle by reducing both communication rounds and costs. Specifically, HeFUN performs ReLU evaluation in just two rounds, whereas Gazelle requires more communication rounds, namely two rounds (to convert between HE and MPC and vice versa) and communication rounds required by GC. In HeFUN, two ciphertexts are transmitted between the client and the server, while the number of messages exchanged in Gazelle is larger, namely two ciphertexts ($[x + r_1]$ and $[\text{ReLU}(x) + r_2]$) and a huge number of messages exchanged in the GC protocol.

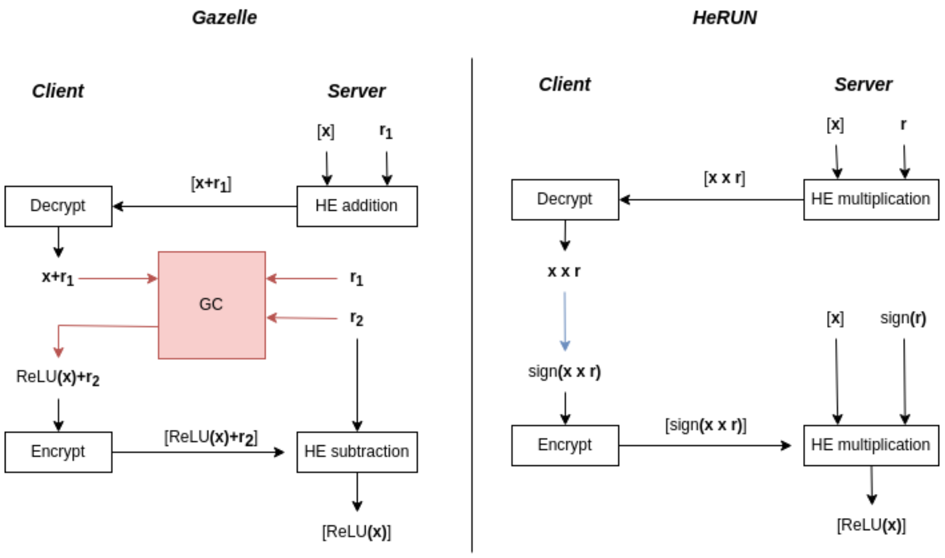

Figure 1. Comparison between Gazelle [15] and HeFUN. The main bottleneck of Gazelle is GC. HeFUN is *solely* based on HE, hence avoiding expensive GC procedure. HeFUN outperforms Gazelle in terms of communication rounds and communication cost.

In Gazelle, the client needs to involve the GC procedure, which is undesirable in SNNI. In contrast, for HeFUN, the client only needs to perform decryption and encryption, which are simple algorithms in the HE scheme, while other heavy computation takes place in the server.

One should acknowledge that in Gazelle, the server's operations include homomorphic addition and subtraction; conversely, in HeFUN, the server performs homomorphic multiplication (as delineated in line 7, Protocol 1), which necessitates an LHE scheme with larger parameters due to the additional multiplications required. Where Gazelle operates with a multiplicative depth of 1, HeFUN requires a depth of 3.

Another HE-MPC hybrid-based SNNI framework is MP2ML [16], which incorporates the ABY protocol [20] for the evaluation of the ReLU function, as opposed to the utilization of GC. In the comparative analysis (see Figure 1), the GC component is substituted with an ABY component. It is obvious that HeFUN demonstrates superior performance over MP2ML in terms of both the number of communication rounds and communication costs. Overall, HeFUN outperforms the current HE-MPC hybrid-based approach in evaluating ReLU in terms of communication complexity. This is primarily attributed to HeFUN's elimination of the necessitated communication-intensive GC or ABY protocols. However, HeFUN presents an elevated multiplicative depth requirement of 3, in contrast to the multiplicative depth of 1 required by Gazelle and MP2ML.

5.2. HeRefresh *Experimental Results*

In the HE scheme, HeRefresh is equivalent to the bootstrapping procedure, which both aim to reduce the noise in ciphertexts. We compare HeRefresh with the bootstrapping technique for CKKS [70], implemented in OpenFHE. We report the result for refreshing noise for one ciphertext. To ensure the reliability of the results, we repeat the experiment 1000 times and take the average result. The comparison results are shown in Table 3. The depth is the number of expected homomorphic multiplications performed before noise refreshing, N is the degree of the polynomial used in the CKKS scheme, and the level is the multiplicative level required for noise refreshing. Bootstrapping homomorphically computes the decryption equation in the encrypted domain. The procedure consumes an amount of homomorphic multiplication, so the level of the CKKS scheme needs to be big enough for this procedure. The details of the bootstrapping procedure are out of the scope of this work. Please refer to [70] for details. While DoubleR (Protocol 2) only requires simple computation, i.e., encryption, decryption, addition, and subtraction, we do not need to increase the level of CKKS. The larger level of CKKS leads to a bigger value of N to meet the security requirement. With simpler operators and smaller parameters, DoubleR is much faster than bootstrapping. For example, with a depth of 1, DoubleR is nearly 300 times faster than bootstrapping. For a depth of 4, with the same N, HeRefresh is nearly 225 times faster than bootstrapping.

Table 3. Comparison between Double-CKKS protocol and bootstrapping.

Depth	Runtime (s)		N		Level	
	DoubleR-CKKS	**Bootstrapping**	**DoubleR-CKKS**	**Bootstrapping**	**DoubleR-CKKS**	**Bootstrapping**
1	0.04	11.85	16,384	65,536	1	22
2	0.05	12.71	16,384	65,536	2	23
4	0.07	14.69	16,384	65,536	4	25
8	0.10	17.39	32,768	65,536	8	29
16	0.19	22.51	65,536	65,536	16	37

5.3. *Comparison between* HeFUN *and LHE (CryptoNets)*

This section aims to compare HeFUN with the LHE-based approaches. The foundational concept for LHE-based approaches originates from the landmark study presented in CryptoNets [12], which utilizes an LHE scheme with a predetermined depth that aligns with the NN's architecture. For the purposes of this analysis, the term 'CryptoNets' will be used to refer to the conventional LHE-based approach.

5.3.1. Experimental Setup

The NNs in this section were trained using PyTorch [71]. To implement HE, we employed TenSEAL [60], with CKKS as instantiation. All parameters in the following sections are chosen to comply with the recommendations on HE standards [72], which satisfies 128 bits of security.

5.3.2. Dataset and NN Setup

We evaluated the performance of NN inference on two distinct datasets, i.e., MNIST [73] and AT&T faces datasets [74]. The MNIST dataset has a standard split containing 28 × 28 grayscale images of Arabic numerals 0 to 9 of 50,000 training images and 10,000 test images. MNIST is the standard benchmark for homomorphic inference tasks [11,12,25,33]. The AT&T faces dataset includes 92 × 112 grayscale images of 40 individuals, with 10 different images of each individual. The dataset is considered a classic dataset in computer vision for experimenting with techniques for face recognition. It offers a more realistic scenario for SNNI, allowing for the recognition of faces while maintaining the confidentiality of the individual images.

The neural networks (NNs) under consideration are composed of convolutional layers, activation functions, and fully connected layers. For the HeFUN framework, the activation function employed is the ReLU function, while LHE-based secure neural network inference (SNNI) typically utilizes the square function as the activation mechanism, as extensively documented in the literature [12,25,26]. To introduce diversity into our experimental evaluation, we have selected two distinct NN architectures, i.e., a small NN and a large NN (the term "large NN" refers to an NN with more complexity than our "small NN", but its size is tailored to stay within the operational limits of LHE-based frameworks for meaningful comparisons). The details of the NNs' architectures for the MNIST dataset are detailed below.

- Small NN:

  - Convolution layer: The input image is 28 $\times$ 28. 5 kernels, each 3 $\times$ 3 in size, with a stride of 2 and no padding. The output is a 5 $\times$ 13 $\times$ 13 tensor.
  - Activation function: This layer applies the approximate activation function to each input value. It is a square function in CryptoNets, and the ReLU function in HeFUN.
  - Fully connected layer: It connects the 845 incoming nodes to 100 outgoing nodes.
  - Activation function: It is the square function in CryptoNets, and the ReLU function in HeFUN.
  - Fully connected layer: It connects the 100 incoming nodes to 10 outgoing nodes (corresponding to 10 classes in the MNIST dataset).
  - Activation function: It is the sigmoid activation function.
- Large NN:
  - Convolution layer: It contains 5 kernels, each 33 in size, with a stride of 2 and no padding. The output is a 5 $\times$ 13 $\times$ 13 tensor.
  - Activation function: It is the square function in CryptoNets, and the ReLU function in HeFUN.
  - Fully connected layer: It connects the 845 incoming nodes to 300 outgoing nodes.
  - Activation function: It is the square function in CryptoNets, and the ReLU function in HeFUN.
  - Fully connected layer: It connects the 300 incoming nodes to 100 outgoing nodes.
  - Activation function: It is the square function in CryptoNets, and the ReLU function in HeFUN.
  - Fully connected layer: connects the 100 incoming nodes to 10 outgoing nodes.
  - Activation function: It is the sigmoid activation function.

In the case of the AT&T faces dataset, the neural network retains a similar layer structure as used with the MNIST dataset. However, adjustments are made to the neuron count per layer to accommodate the dataset's structure. For instance, the output layer features 40 neurons, corresponding to the 40 distinct classes represented in the dataset, in contrast to the 10-neuron configuration used for datasets like MNIST.

As we mention in Section 4.1, the last sigmoid activation function can be removed in the inference phase without interfering with prediction accuracy.

5.3.3. HE Parameter

The main parameters defining the CKKS scheme [24] are the degree N of the polynomial modulus $X^N + 1$, the coefficient modulus q, and multiplicative depth L. The multiplicative depth L of the CKKS scheme was chosen to align with the NN's depth. Subsequently, the modulus q is carefully chosen to meet the security criterion of 128 bits, taking into account the specified multiplicative depth L. Within the HeFUN framework, the HeRefresh protocol is employed to refresh the outputs of intermediate layers, thereby facilitating additional multiplications. Table 4 details the parameters. CryptoNets$_s$ (HeFUN$_s$) and CryptoNets$_l$ (HeFUN$_l$) stand for CryptoNets (HeFUN) on small NN and large NN, respectively. We choose N = 16,384 in both frameworks. Notably, in the case of CryptoNets, an escalation in NN depth from a small to a large model necessitates an increase in L (from 5 to 7), consequently requiring an increase in modulus q to preserve the desired security level. Conversely, HeFUN maintains a constant and reduced multiplicative depth regardless of NN depth increments—specifically 3 as per Table 4—thanks to its intermediate ciphertext refreshing mechanism. It is imperative to recognize that elevated values of L and q slow down inference time.

Table 4. LHE's parameters of CryptoNets and HeFUN. The best results are in bold font.

	Security Level	N	L	q (Bit Length)
CryptoNets_s	128	16,384	5	420
HeFUN_s	128	16,384	**3**	**320**
CryptoNets_l	128	16,384	7	520
HeFUN_l	128	16,384	**3**	**320**

5.3.4. Experimental Results

Tables 5 and 6 present the experimental results on the MNIST and AT&T faces datasets, respectively, upon which we base our analysis of the accuracy and inference time associated with HeFUN and CryptoNets.

Table 5. Comparison between HeFUN frameworks and CryptoNets1 on MNIST dataset. The best results are in bold font.

		CryptoNets_s	HeFUN_s	CryptoNets_l	HeFUN_l
	Accuracy (%)	98.15	**98.31**	98.52	**99.16**
Inference times (s)	Encoding+Encryption	0.008	0.007	0.011	0.007
	Convolution	0.503	0.395	0.829	0.395
	Activation 1 [a]	0.008	0.028	0.012	0.028
	Fully connected 1	1.121	0.812	2.138	0.812
	Activation 2	0.008	0.029	0.012	0.029
	Fully connected 2	0.065	0.102	0.408	0.102
	Activation 3	-	-	0.011	0.029
	Fully connected 3	-	-	0.073	0.098
	Decryption + Decoding	0.002	0.001	0.003	0.001
	Total	1.715	**1.374**	3.497	**1.501**

[a] Activation function is the square function in CryptoNets and is ReLU function in HeFUN.

Table 6. Comparison between HeFUN frameworks and CryptoNets1 on AT&T faces dataset. The best results are in bold font.

		CryptoNets_s	HeFUN_s	CryptoNets_l	HeFUN_l
	Accuracy (%)	95.19	**96.66**	96.87	**97.43**
Inference times (s)	Encoding+Encryption	0.009	0.007	0.013	0.007
	Convolution	0.695	0.425	0.991	0.435
	Activation 1 [a]	0.008	0.027	0.012	0.028
	Fully connected 1	1.583	1.115	3.171	1.115
	Activation 2	0.009	0.030	0.012	0.030
	Fully connected 2	0.065	0.102	0.408	0.102
	Activation 3	-	-	0.011	0.029
	Fully connected 3	-	-	0.081	0.102
	Decryption + Decoding	0.002	0.001	0.003	0.001
	Total	2.361	**1.707**	5.248	**1.849**

[a] Activation function is the square function in CryptoNets and is ReLU function in HeFUN.

Accuracy. It can be seen that HeFUN outperforms CryptoNets in accuracy. For the MNIST dataset, HeFUN_l and HeFUN_s achieve accuracy of 99.16% and 98.31%, respectively, while CryptoNet_l and CryptoNet_s achieve accuracy of 98.52% and 98.15%, respectively. The difference is caused by the activation used in the frameworks, namely, HeFUN used the ReLU activation function, whereas CryptoNets used the square function. Similarly, for the AT&T faces dataset, HeFUN_l and HeFUN_s achieve accuracy of 97.43% and 96.66%, respectively, while CryptoNet_l and CryptoNet_s achieve accuracy of 96.87% and 95.19%, respectively.

Inference. The timing results reported in Tables 5 and 6 are the total running time at the client and server. As shown in Tables 5 and 6, HeFUN is faster than CryptoNets for both the small NN and large NN. This acceleration is because LHE parameters in HeFUN are smaller than CryptoNets. For instance, in Table 5, with an increase in NN depth from 5 to 7, CryptoNets shows a significant jump in inference time—from 1.715 s to 3.497 s. This increase is due to the escalated complexity of LHE parameters necessitated by a deeper network. Conversely, HeFUN benefits from the HeRefresh protocol, which mitigates noise accumulation at intermediate layers and allows for consistent LHE parameters regardless of NN depth. Consequently, in HeFUN, a similar increase in NN depth results in a modest increase in inference time, from 1.374 s to 1.501 s. Interestingly, in HeFUN, the computation time at the same layers almost remains unchanged, regardless of the depth of NN, due to its invariable LHE parameter requirements. On the other hand, in CryptoNets, deeper NNs require an increase in the LHE's parameters, which slows down the computation time. For instance, the layer Fully connected 1 takes 0.812 s in both HeFUN_s and HeFUN_l, while it takes 1.121 s in CryptoNet_s, and takes 2.138 s in CryptoNet_l. This observation reinforces the scalability of HeFUN in terms of computation time when compared to CryptoNets as NN complexity increases.

We now analyze the running time at the activation function layers. The evaluation of activation function layers' running times, as shown in Table 1, reveals that HeFUN exhibits slower performance compared to CryptoNets. This difference stems from the intrinsic complexity of the activation functions utilized; CryptoNets employs a square function as the activation, which is a single homomorphic multiplication of two ciphertexts, whereas HeFUN utilizes the more complex HeReLU protocol for evaluating the ReLU function, as detailed in Table 7.

Table 7. Comparison between HeReLU protocol and square function evaluation in CryptoNets.

	Depth	Encryption	Decryption	Multiplication		Addition	
				Ciphert-Plain	**Cipher-Cipher**	**Ciphert-Plain**	**Cipher-Cipher**
CryptoNets	L [a]	0	0	0	1	0	0
HeReLU	3	1	1	3	1	0	1

[a] Depend on the NN's depth.

Despite this, HeFUN has an advantage in terms of its independence from the neural network's (NN) depth. Specifically, HeFUN maintains a *constant* multiplicative depth of 3, regardless of the NN's depth, which avoids the escalation of LHE parameters—and consequently, inference times—that is seen with CryptoNets as the NN becomes deeper. To illustrate, the Activation 1 layer in HeFUN consistently requires 0.028 s for both small and large NNs (HeFUN_s and HeFUN_l), in contrast to CryptoNets, which records 0.008 s for a small NN (CryptoNet_s) and increases to 0.012 s for a large NN (CryptoNet_l). This observation reinforces the scalability of HeFUN in terms of computation time when compared to CryptoNets as NN complexity increases.

6. Discussion on Security of HeFUN

6.1. Model Extraction Attack

While HeFUN ensures security against semi-honest adversaries by protecting client data and the model's parameters, it remains vulnerable to model extraction attacks. Adversaries can potentially exploit a series of inference queries to reverse-engineer the model's parameters [75,76], reconstruct training samples [77,78], or deduce the presence of specific samples in the training set [8]. Indeed, all the frameworks in Table 1 remain vulnerable to such attacks. We acknowledge that addressing model extraction is a distinct challenge from SNNI and is not the primary focus of our current work.

6.2. Fully Private SNNI

HeFUN hides all the NN's parameters from the client and hides clients' data from the server. However, HeFUN does not completely hide the network architecture. Our protocol leaks the number of layers and the number of neurons in each layer to the client. To further enhance privacy, we can adopt the padding technique used in the Gazelle framework, which involves adding dummy layers and neurons to conceal the network's true architecture at the cost of additional computational resources. Furthermore, our protocol obviously leaks the type of activation function used in the NN to the client. It should be noted that all frameworks in the MPC-based and HE-MPC hybrid-based approaches (in Table 1) leak such information. Among current SNNI approaches, LHE-based and FHE-based approaches do not expose any details about the NN, thereby providing a superior level of security. Despite this, such methods are constrained by their inability to efficiently compute certain functions like ReLU, their limitation in handling deep NNs, and the significant computational overhead involved. These limitations reduce their scalability and practical applicability in complex SNNI tasks.

7. Conclusions and Future Works

This paper introduces iLHE, a novel interactive LHE-based approach for SNNI. Building upon iLHE, we developed two innovative protocols that address the limitations of current HE-based SNNI methods. Our HeReLU protocol enables the evaluation of the ReLU activation function with reduced communication rounds compared to existing HE-MPC hybrid solutions. We also introduce HeRefresh, a refreshing framework that facilitates deep computation by allowing additional homomorphic operations without increasing HE parameters, thereby accelerating HE scheme computations. Uniquely, HeRefresh sidesteps the need for Gentry's bootstrapping and its associated circular security assumptions. Leveraging HeReLU and HeRefresh, we propose a comprehensive framework for SNNI that offers enhanced security, improved accuracy, fewer communication rounds, and faster inference times, as demonstrated by our experimental results. This framework has been rigorously validated to be secure within the semi-honest model.

This work introduces an iLHE-based method for efficiently evaluating the ReLU activation in NNs, circumventing the communication overhead associated with MPC protocols. An avenue for future research lies in extending iLHE-based techniques to other non-linear functions, such as max-pooling and softmax. Such advancements could significantly enhance the performance of existing SNNI frameworks.

Author Contributions: Conceptualization, D.T.K.N.; Methodology, D.T.K.N.; Validation, D.T.K.N., D.H.D. and T.A.T.; Formal analysis, D.T.K.N. and D.H.D.; Investigation, D.T.K.N.; Writing–original draft, D.T.K.N.; Writing–review & editing, D.H.D., W.S., Y.-W.C. and T.A.T.; Supervision, D.H.D., W.S. and Y.-W.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data presented in this study are available in this article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Tan, M.; Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In Proceedings of the International Conference on Machine Learning, PMLR, Long Beach, CA, USA, 9–15 June 2019; pp. 6105–6114.
2. Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. An image is worth 16 × 16 words: Transformers for image recognition at scale. *arXiv* **2020**, arXiv:2010.11929.
3. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language models are few-shot learners. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 1877–1901.
4. Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P.J. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.* **2020**, *21*, 5485–5551.
5. Park, D.S.; Chan, W.; Zhang, Y.; Chiu, C.C.; Zoph, B.; Cubuk, E.D.; Le, Q.V. Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv* **2019**, arXiv:1904.08779.
6. Gulati, A.; Qin, J.; Chiu, C.C.; Parmar, N.; Zhang, Y.; Yu, J.; Han, W.; Wang, S.; Zhang, Z.; Wu, Y.; et al. Conformer: Convolution-augmented transformer for speech recognition. *arXiv* **2020**, arXiv:2005.08100.
7. OpenAI. ChatGPT. 2023. Available online: https://chat.openai.com (accessed on 3 November 2023).
8. Shokri, R.; Stronati, M.; Song, C.; Shmatikov, V. Membership inference attacks against machine learning models. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2017; pp. 3–18.

9. Lee, E.; Lee, J.W.; No, J.S.; Kim, Y.S. Minimax approximation of sign function by composite polynomial for homomorphic comparison. *IEEE Trans. Dependable Secur. Comput.* **2021**, *19*, 3711–3727. [CrossRef]
10. Cheon, J.H.; Kim, D.; Kim, D. Efficient homomorphic comparison methods with optimal complexity. In *Advances in Cryptology–ASIACRYPT 2020, Proceedings of the 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, Republic of Korea, 7–11 December 2020*; Proceedings, Part II 26; Springer: Cham, Switzerland, 2020; pp. 221–256.
11. Boemer, F.; Costache, A.; Cammarota, R.; Wierzynski, C. 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, London, UK, 11 November 2019; pp. 45–56.
12. Gilad-Bachrach, R.; Dowlin, N.; Laine, K.; Lauter, K.; Naehrig, M.; Wernsing, J. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In Proceedings of the International Conference on Machine Learning, PMLR, New York, NY, USA, 20–22 June 2016; pp. 201–210.
13. Gentry, C. Fully homomorphic encryption using ideal lattices. In Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, Bethesda, MD, USA, 31 May–2 June 2009; pp. 169–178.
14. Podschwadt, R.; Takabi, D.; Hu, P.; Rafiei, M.H.; Cai, Z. A survey of deep learning architectures for privacy-preserving machine learning with fully homomorphic encryption. *IEEE Access* **2022**, *10*, 117477–117500. [CrossRef]
15. Juvekar, C.; Vaikuntanathan, V.; Chandrakasan, A. GAZELLE: A low latency framework for secure neural network inference. In Proceedings of the 27th USENIX Security Symposium (USENIX Security 18), Baltimore, MD, USA, 15–17 August 2018; pp. 1651–1669.
16. Boemer, F.; Cammarota, R.; Demmler, D.; Schneider, T.; Yalame, H. MP2ML: A mixed-protocol machine learning framework for private inference. In Proceedings of the 15th International Conference on Availability, Reliability and Security, Virtual Event, 25–28 August 2020; pp. 1–10.
17. Lehmkuhl, R.; Mishra, P.; Srinivasan, A.; Popa, R.A. Muse: Secure inference resilient to malicious clients. In Proceedings of the 30th USENIX Security Symposium (USENIX Security 21), Virtual Event, 11–13 August 2021; pp. 2201–2218.
18. Chen, S.; Fan, J. SEEK: Model extraction attack against hybrid secure inference protocols. *arXiv* **2022**, arXiv:2209.06373.
19. Yao, A.C.C. How to generate and exchange secrets. In Proceedings of the 27th Annual Symposium on Foundations of Computer Science (Sfcs 1986), Toronto, ON, Canada, 27–29 October 1986; pp. 162–167.
20. Mohassel, P.; Rindal, P. ABY3: A mixed protocol framework for machine learning. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October 2018; pp. 35–52.
21. Goldreich, O. *Foundations of Cryptography: Volume 2, Basic Applications*; Cambridge University Press: Cambridge, UK, 2009.
22. Canetti, R. Security and composition of multiparty cryptographic protocols. *J. Cryptol.* **2000**, *13*, 143–202. [CrossRef]
23. Fan, J.; Vercauteren, F. Somewhat Practical Fully Homomorphic Encryption. *Cryptology ePrint Archive*. 2012. Available online: https://eprint.iacr.org/2012/144 (accessed on 13 November 2023).
24. Cheon, J.H.; Kim, A.; Kim, M.; Song, Y. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017, Proceedings of the 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, 3–7 December 2017*; Proceedings, Part I 23; Springer: Cham, Switzerland, 2017; pp. 409–437.
25. Hesamifard, E.; Takabi, H.; Ghasemi, M. CryptoDL: Deep neural networks over encrypted data. *arXiv* **2017**, arXiv:1711.05189.
26. Chabanne, H.; De Wargny, A.; Milgram, J.; Morel, C.; Prouff, E. Privacy-Preserving Classification on Deep Neural Network. *Cryptology ePrint Archive*. 2017. Available online: https://eprint.iacr.org/2017/1114 (accessed on 13 November 2023).
27. Chou, E.; Beal, J.; Levy, D.; Yeung, S.; Haque, A.; Fei-Fei, L. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *arXiv* **2018**, arXiv:1811.09953.
28. Al Badawi, A.; Jin, C.; Lin, J.; Mun, C.F.; Jie, S.J.; Tan, B.H.M.; Nan, X.; Aung, K.M.M.; Chandrasekhar, V.R. Towards the alexnet moment for homomorphic encryption: Hcnn, the first homomorphic cnn on encrypted data with gpus. *IEEE Trans. Emerg. Top. Comput.* **2020**, *9*, 1330–1343. [CrossRef]
29. Jiang, X.; Kim, M.; Lauter, K.; Song, Y. Secure outsourced matrix computation and application to neural networks. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October 2018; pp. 1209–1222.
30. Brutzkus, A.; Gilad-Bachrach, R.; Elisha, O. Low latency privacy preserving inference. In Proceedings of the International Conference on Machine Learning. PMLR, Long Beach, CA, USA, 9–15 June 2019; pp. 812–821.
31. Dathathri, R.; Saarikivi, O.; Chen, H.; Laine, K.; Lauter, K.; Maleki, S.; Musuvathi, M.; Mytkowicz, T. CHET: An optimizing compiler for fully-homomorphic neural-network inferencing. In Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, Phoenix, AZ, USA, 22 June 2019; pp. 142–156.
32. van Elsloo, T.; Patrini, G.; Ivey-Law, H. SEALion: A framework for neural network inference on encrypted data. *arXiv* **2019**, arXiv:1904.12840.
33. Boemer, F.; Lao, Y.; Cammarota, R.; Wierzynski, C. nGraph-HE: A graph compiler for deep learning on homomorphically encrypted data. In Proceedings of the 16th ACM International Conference on Computing Frontiers, Alghero, Italy, 30 April–2 May 2019; pp. 3–13.
34. Smart, N.P.; Vercauteren, F. Fully homomorphic SIMD operations. *Des. Codes Cryptogr.* **2014**, *71*, 57–81. [CrossRef]
35. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
36. Zagoruyko, S.; Komodakis, N. Wide residual networks. *arXiv* **2016**, arXiv:1605.07146.

37. Chillotti, I.; Gama, N.; Georgieva, M.; Izabachène, M. TFHE: Fast fully homomorphic encryption over the torus. *J. Cryptol.* **2020**, *33*, 34–91. [CrossRef]
38. Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; Bengio, Y. Binarized neural networks. In *Advances in Neural Information Processing Systems 29 (NIPS 2016)*; Curran Associates, Inc.: Red Hook, NY, USA, 2016; Volume 29.
39. Bourse, F.; Minelli, M.; Minihold, M.; Paillier, P. Fast homomorphic evaluation of deep discretized neural networks. In *Advances in Cryptology–CRYPTO 2018, Proceedings of the 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, 19–23 August 2018*; Proceedings, Part III 38; Springer: Cham, Switzerland, 2018; pp. 483–512.
40. Sanyal, A.; Kusner, M.; Gascon, A.; Kanade, V. TAPAS: Tricks to accelerate (encrypted) prediction as a service. In Proceedings of the International Conference on Machine Learning. PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 4490–4499.
41. Lou, Q.; Jiang, L. SHE: A fast and accurate deep neural network for encrypted data. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 10035–10043.
42. Clet, P.E.; Stan, O.; Zuber, M. BFV, CKKS, TFHE: Which One is the Best for a Secure Neural Network Evaluation in the Cloud? In *Applied Cryptography and Network Security Workshops, Proceedings of the ACNS 2021 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, and SiMLA, Kamakura, Japan, 21–24 June 2021*; Proceedings; Springer: Cham, Switzerland, 2021; pp. 279–300.
43. Rouhani, B.D.; Riazi, M.S.; Koushanfar, F. Deepsecure: Scalable provably-secure deep learning. In Proceedings of the 55th Annual Design Automation Conference, San Francisco, CA, USA, 24–29 June 2018; pp. 1–6.
44. Riazi, M.S.; Samragh, M.; Chen, H.; Laine, K.; Lauter, K.; Koushanfar, F. XONN: XNOR-based oblivious deep neural network inference. In Proceedings of the 28th USENIX Security Symposium (USENIX Security 19), Santa Clara, CA, USA, 14–16 August 2019; pp. 1501–1518.
45. Ball, M.; Carmer, B.; Malkin, T.; Rosulek, M.; Schimanski, N. Garbled Neural Networks Are Practical. *Cryptology ePrint Archive*. 2019. Available online: https://eprint.iacr.org/2019/338 (accessed on 13 November 2023).
46. Ng, L.K.; Chow, S.S. SoK: Cryptographic Neural-Network Computation. In Proceedings of the 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 22–24 May 2023; pp. 497–514.
47. Shamir, A. How to share a secret. *Commun. ACM* **1979**, *22*, 612–613. [CrossRef]
48. Micali, S.; Goldreich, O.; Wigderson, A. How to play any mental game. In Proceedings of the Nineteenth ACM Symposium on Theory of Computing (STOC), New York, NY, USA, 25–27 May 1987; pp. 218–229.
49. Mohassel, P.; Zhang, Y. SecureML: A system for scalable privacy-preserving machine learning. In Proceedings of the 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2017; pp. 19–38.
50. Patra, A.; Schneider, T.; Suresh, A.; Yalame, H. ABY2. 0: Improved Mixed-Protocol Secure Two-Party Computation. In Proceedings of the 30th USENIX Security Symposium (USENIX Security 21), Virtual Event, 11–13 August 2021; pp. 2165–2182.
51. Agrawal, N.; Shahin Shamsabadi, A.; Kusner, M.J.; Gascón, A. QUOTIENT: Two-party secure neural network training and prediction. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London, UK, 11–15 November 2019; pp. 1231–1247.
52. Riazi, M.S.; Weinert, C.; Tkachenko, O.; Songhori, E.M.; Schneider, T.; Koushanfar, F. Chameleon: A hybrid secure computation framework for machine learning applications. In Proceedings of the 2018 on Asia Conference on Computer and Communications Security, Incheon, Republic of Korea, 4–8 June 2018; pp. 707–721.
53. Wagh, S.; Gupta, D.; Chandran, N. SecureNN: 3-Party Secure Computation for Neural Network Training. *Proc. Priv. Enhancing Technol.* **2019**, *2019*, 26–49. [CrossRef]
54. Wagh, S.; Tople, S.; Benhamouda, F.; Kushilevitz, E.; Mittal, P.; Rabin, T. Falcon: Honest-majority maliciously secure framework for private deep learning. *arXiv* **2020**, arXiv:2004.02229. [CrossRef]
55. Kumar, N.; Rathee, M.; Chandran, N.; Gupta, D.; Rastogi, A.; Sharma, R. CrypTFlow: Secure tensorflow inference. In Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 18–21 May 2020; pp. 336–353.
56. Knott, B.; Venkataraman, S.; Hannun, A.; Sengupta, S.; Ibrahim, M.; van der Maaten, L. Crypten: Secure multi-party computation meets machine learning. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 4961–4973.
57. Demmler, D.; Schneider, T.; Zohner, M. ABY-A framework for efficient mixed-protocol secure two-party computation. In Proceedings of the NDSS, San Diego, CA, USA, 8–11 February 2015.
58. Halevi, S.; Shoup, V. Algorithms in helib. In *Advances in Cryptology–CRYPTO 2014, Proceedings of the 34th Annual Cryptology Conference, Santa Barbara, CA, USA, 17–21 August 2014*; Proceedings, Part I 34; Springer: Cham, Switzerland, 2014; pp. 554–571.
59. Lee, J.W.; Kang, H.; Lee, Y.; Choi, W.; Eom, J.; Deryabin, M.; Lee, E.; Lee, J.; Yoo, D.; Kim, Y.S.; et al. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access* **2022**, *10*, 30039–30054. [CrossRef]
60. Benaissa, A.; Retiat, B.; Cebere, B.; Belfedhal, A.E. TenSEAL: A library for encrypted tensor operations using homomorphic encryption. *arXiv* **2021**, arXiv:2104.03152.
61. Lou, Q.; Jiang, L. Hemet: A homomorphic-encryption-friendly privacy-preserving mobile neural network architecture. In Proceedings of the International Conference on Machine Learning, PMLR, Virtual, 18–24 July 2021; pp. 7102–7110.
62. Lyubashevsky, V.; Peikert, C.; Regev, O. On ideal lattices and learning with errors over rings. In *Advances in Cryptology–EUROCRYPT 2010, Proceedings of the 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, 30 May–3 June 2010*; Proceedings 29; Springer: Cham, Switzerland, 2010; pp. 1–23.

63. Cheon, J.H.; Kim, D.; Kim, D.; Lee, H.H.; Lee, K. Numerical method for comparison on homomorphically encrypted numbers. In Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, 8–12 December 2019; pp. 415–445.
64. Lee, E.; Lee, J.W.; Kim, Y.S.; No, J.S. Optimization of homomorphic comparison algorithm on rns-ckks scheme. *IEEE Access* **2022**, *10*, 26163–26176. [CrossRef]
65. Cortes, C.; Vapnik, V. Support-vector networks. *Mach. Learn.* **1995**, *20*, 273–297. [CrossRef]
66. Comaniciu, D.; Meer, P. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.* **2002**, *24*, 603–619. [CrossRef]
67. Friedman, J.H. Greedy function approximation: A gradient boosting machine. *Ann. Stat.* **2001**, *29*, 1189–1232. [CrossRef]
68. Brakerski, Z.; Gentry, C.; Vaikuntanathan, V. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory (TOCT)* **2014**, *6*, 1–36. [CrossRef]
69. Mishra, P.; Lehmkuhl, R.; Srinivasan, A.; Zheng, W.; Popa, R.A. Delphi: A cryptographic inference system for neural networks. In Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice, Virtual Event, 9–13 November 2020; pp. 27–30.
70. Al Badawi, A.; Polyakov, Y. Demystifying Bootstrapping in Fully Homomorphic Encryption. *Cryptology ePrint Archive*. 2023. Available online: https://eprint.iacr.org/2023/149 (accessed on 13 November 2023).
71. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. Pytorch: An imperative style, high-performance deep learning library. *Adv. Neural Inf. Process. Syst.* **2019**; 8024–8035.
72. Albrecht, M.; Chase, M.; Chen, H.; Ding, J.; Goldwasser, S.; Gorbunov, S.; Halevi, S.; Hoffstein, J.; Laine, K.; Lauter, K.; et al. Homomorphic encryption standard. In *Protecting Privacy through Homomorphic Encryption*; Springer: Cham, Switzerland, 2021; pp. 31–62.
73. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
74. Samaria, F.S.; Harter, A.C. Parameterisation of a stochastic model for human face identification. In Proceedings of 2nd IEEE Workshop on Applications of Computer Vision, Sarasota, FL, USA, 5–7 December 1994; pp. 138–142.
75. Tramèr, F.; Zhang, F.; Juels, A.; Reiter, M.K.; Ristenpart, T. Stealing machine learning models via prediction APIs. In Proceedings of the 25th USENIX security symposium (USENIX Security 16), Austin, TX, USA, 10–12 August 2016; pp. 601–618.
76. Carlini, N.; Jagielski, M.; Mironov, I. Cryptanalytic extraction of neural network models. In Proceedings of the Annual International Cryptology Conference, Barbara, CA, USA, 17–21 August 2020; pp. 189–218.
77. Aïvodji, U.; Gambs, S.; Ther, T. Gamin: An adversarial approach to black-box model inversion. *arXiv* **2019**, arXiv:1909.11835.
78. Bekman, T.; Abolfathi, M.; Jafarian, H.; Biswas, A.; Banaei-Kashani, F.; Das, K. Practical black box model inversion attacks against neural nets. In Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Bilbao, Spain, 13–17 September 2021; pp. 39–54.