# 2024 Master Thesis

# Investigation of CKKS-Induced Error on Accuracy Changes of GraphSAGE Inference with Fully Homomorphic Encryption

Submission Date: July 22nd, 2024
Advisor: Prof. Hayato Yamana
Research guidance: Research on Parallel and Distributed Architecture

A Thesis Submitted to the Department of Computer Science and Communications Engineering, the Graduate School of Fundamental Science and Engineering of Waseda University in Partial Fulfillment of the Requirements for the Degree of Master of Engineering

Student ID: 5122FG24-2

**Houtao Huang**

# Abstract

The integration of Graph Neural Network (GNN) into recommendation systems has demonstrated remarkable performance, but using personal data to make recommendations raises significant privacy concerns. This thesis explores the application of fully homomorphic encryption (FHE) to GraphSAGE, a variant of GNN, to enable privacy-preserving recommendations. Although the Cheon-Kim-Kim-Song (CKKS) scheme is widely used for privacy-preserving machine learning, the CKKS scheme introduces errors such as polynomial approximation errors, encoding errors, and rescaling errors. This research aims to investigate the impact of these CKKS-induced errors on the accuracy of FHE-encrypted GraphSAGE inference. Our results show that a higher degree of polynomial approximation increases the latency most because of the increased frequency of homomorphic multiplication. Using a higher scale setting results in smaller encoding errors and increases accuracy by 1.15% with 0.19 hours latency increasing. Fewer rescaling operations introduce smaller rescaling errors. For threshold rescaling, the accuracy decreases by 0.48%, but latency decreases by 1.23 hours. To increase the accuracy of the inference, we can consider increasing the scale setting first or use a more precise polynomial approximation technique. To reduce the latency of the inference, lightweight polynomial approximation should be used first followed by a small number of rescaling.

# Contents

# Chapter 1

# Introduction

Nowadays, employing machine learning (ML) for recommendation systems has received rising attraction because of its ability to simulate consumption from customers. By employing ML for recommendation systems, companies can rapidly provide recommendations that are specific to each client so that revenue increases potentially. For recommendation systems, graph neural network (GNN) has emerged as a useful tool because of its advantage of not only considering individual data points but also considering the relationships between the data points in the dataset [1].

However, to provide recommendations, plenty of private information of customers has to be used in training and employing ML models, which also raises concerns about leaking the privacy of customers to recommendation systems providers. Therefore, the potential to enable privacy-preserving computation has made the combination of fully homomorphic encryption (FHE) and machine learning (ML) an attractive tool to achieve privacy-preserving machine learning (PPML). By employing FHE on ML models, the ML models are able to compute without decryption so that privacy can be protected.

Although the research on the integration of FHE and ML has become a popular topic recently and there is already much-existing research such as FHE version of ResNet-20 [2], FHE version of convolutional neural network (CNN) [3] and FHE version of Support Vector Machine (SVM) [4]. Among different FHE schemes, the Cheon-Kim-Kim-Song (CKKS) scheme [5] has been most widely used for problems of integration of FHE and machine learning [6] because of the CKKS scheme's advantage of approximate computations on real numbers [7]. The research about combining FHE and GNN is still lacking because of the difficulty of considering the relationships among instances while the relationships are encrypted for privacy preservation.

The CKKS scheme introduces errors such as polynomial approximation error of ReLU, encoding errors [8], and rescaling errors [8] during approximate arithmetic. Besides, because the CKKS scheme is constrained in its ability to directly implement conditional branching, the non-linear activation functions in the neural network cannot be implemented. Thus, the non-linear activation functions should be approximated by polynomials for homomorphic operations [9], which also introduces errors. Some existing papers researched the effect of CKKS-induced errors in other neural networks[10, 9, 11, 12]. However, the CKKS-induced errors may have a larger effect on GNN since it not only considers the individual data but also the relationships among data, which requires more homomorphic computations. Therefore, understanding the effect of CKKS scheme errors on the accuracy of GNN inference is crucial to ensure the reliability of privacy-preserving GNN.

This thesis focuses on investigating the effect of errors from the CKKS scheme on the accuracy of GraphSAGE [13] inference with FHE. GraphSAGE [13] is one of the GNN models proposed by Hamilton et al. in 2017. To achieve our goal, this thesis implemented an FHE-encrypted GraphSAGE inference over the CKKS scheme by Microsoft SEAL [14]. The main contributions of this thesis are as follows:

1. First propose a privacy-preserving GraphSAGE model where the privacy of the inputs is encrypted by FHE.

2. Investigating the effect of polynomial approximation error of ReLU on the accuracy of FHE-encrypted GraphSAGE inference.

3. Investigating the effect of encoding error from CKKS scheme on the accuracy of FHE-encrypted GraphSAGE inference.

4. Investigating the effect of rescaling error from CKKS scheme on the accuracy of FHE-encrypted GraphSAGE inference.

The privacy-preserving recommendation system of the scenario of this thesis is shown in Figure 1.1. There are three parties in the privacy-preserving recommendation system of this thesis. They are customers who send queries, a graph owner who owns the graph and the pre-trained GraphSAGE model, and a computation resource to make inferences. The graph owner utilizes the historical data of customers to generate the graph and train the GraphSAGE model. The privacy of the customers can be prevented from being

leaked from the graph owner and computation resource while making inferences through the privacy-preserving recommendation system.

The threat model in this system is that the graph owner and computation resources are both semi-honest, which means that they follow the protocol of the privacy-preserving recommendation system but intend to reveal the privacy of customers. The workflow of this privacy-preserving recommendation system is as follows:

1. A Customer sends a request to the graph owner to ask for a recommendation. After receiving the request, the graph owner sends a query questionnaire to ask the customer to fill in. The questionnaire contains two parts: features of themselves (features are the own information of each individual customer), the relationship between them, and something such as favorite restaurants or favourite institutions depending on the scenario.

2. The customer will fill out this questionnaire and encrypt it with the public key as the encrypted query.

3. The graph owner encrypts the graph with the same public key and sends the pre-trained model, encrypted graph, and encrypted query to computation resources to make an inference.

4. Finally, the computation resource returns the inference result which is encrypted. Then, only the customer who has the secret key can decrypt it and check the result.

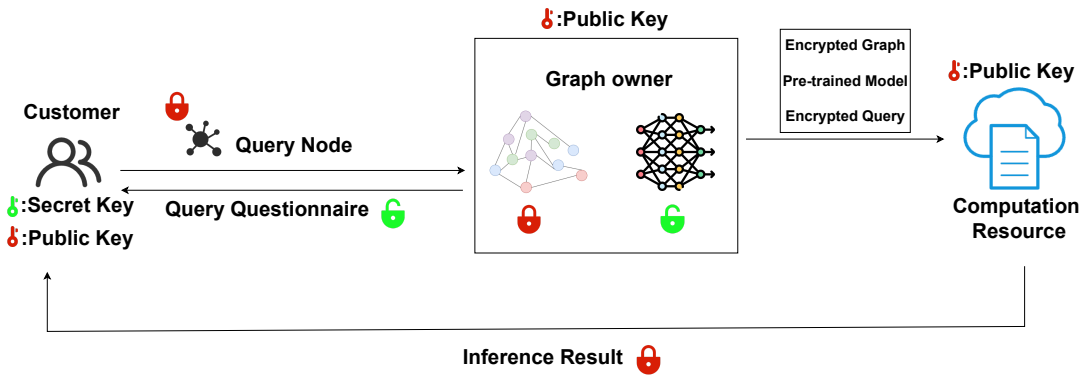



Figure 1.1: Overview of proposed privacy-preserving recommendation system

There are 5 chapters remaining for this thesis. Chapter 2 describes the background knowledge which is relevant to this thesis. Chapter 3 explains related work to this thesis. Chapter 4 presents the methods we used and experiments we conducted. Chapter 5 shows the results and discussion of the results of our experiments. Lastly, Chapter 6 is the conclusion of this thesis.

# Chapter 2

# Preliminary

In this chapter, we explain the background knowledge for this thesis.

Section 2.1 introduces the definition of homomorphic encryption (HE). HE is an encryption technique that enables computation (addition or/and multiplication) of ciphertexts without decryption. In this thesis, we use fully homomorphic encryption (FHE) which is one type of HE.

Section 2.2 introduces the Cheon-Kim-Kim-Song (CKKS) scheme [5], one of the FHE schemes. CKKS scheme[5] is most used in machine learning problems [6]. In this thesis, we utilize CKKS scheme.

Section 2.3 explains the errors in the CKKS scheme[5]. Although the CKKS scheme is widely used in secure machine learning problems, the CKKS scheme introduces some errors. Three types of errors are included in Section 2.3: Section 2.3.1 introduces the polynomial approximation errors while approximating the ReLU function; Section 2.3.2 introduces the encoding error while encoding the message into plaintext in CKKS scheme; Section 2.3.3 introduces the rescaling error while doing rescaling on the ciphertext in CKKS scheme.

Section 2.4 explains the GraphSage[13] model. The GraphSage model was proposed by William L. et al. in 2017, which is one of the most famous graph neural network (GNN) models.

## 2.1 Homomorphic Encryption (HE)

HE is an encryption technique that enables computation (addition and/or multiplication) with ciphertext without decryption. Equations 2.1 and 2.2 show the processing of homomorphic addition and homomorphic multiplication respectively, where $P_t x$ and $P_t y$ are two plaintexts of x and y respectively. The $Enc$ and $Dec$ represent encryption and decryption, respectively.

$$Dec(Enc(P_t x) + Enc(P_t y)) = x + y \tag{2.1}$$

$$Dec(Enc(P_t x) \times Enc(P_t y)) = x \times y \tag{2.2}$$

There are different types of homomorphic encryption (HE). For instance, partially homomorphic encryption, which supports either addition (additive Homomorphic Encryption) [15] or multiplication (multiplicative homomorphic encryption) [16] and fully homomorphic encryption (FHE) [17] which can support an arbitrary number of both multiplication and addition over ciphertexts by applying a technique called bootstrapping. Besides, one type of HE called leveled homomorphic encryption (LHE) [18] can also support both homomorphic addition and homomorphic multiplication but in pre-defined times.

## 2.2 Cheon-Kim-Kim-Song (CKKS) Scheme

There are many different HE schemes, for instance, Paillier scheme [15], Boneh-Goh-Nissim (BGN) scheme [19], Brakerski-Fan-Vercauteren (BFV) scheme [20], and Brakerski-Gentry-Vaikuntanathan (BGV) scheme [21]. Among various HE schemes, for machine learning problems, the Cheon-Kim-Kim-Song (CKKS) scheme [5] is the most used [6] because it is able to do approximate computation on real numbers [7].

There are several terminologies in the CKKS scheme, the Figure 2.1 shows the processing of a message encrypted in the CKKS scheme and performing homomorphic computations and Rescaling/Relinearization for FHE (bootstrapping and rotation are omitted).

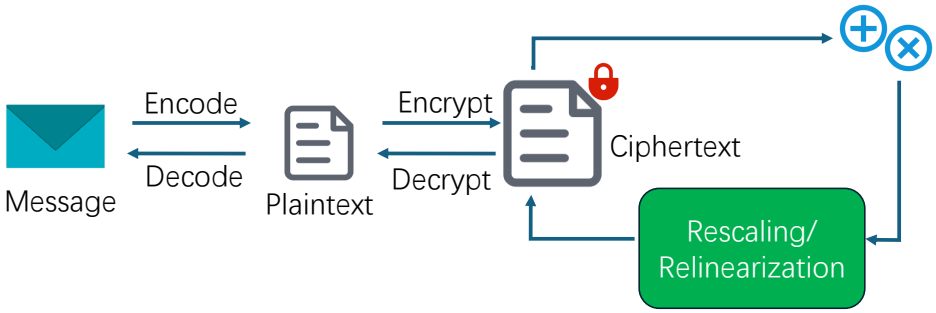



Figure 2.1: Processing in CKKS scheme

**Message**: A message is an original input which is encrypted in the CKKS scheme.

**Plaintext**: Plaintext is the polynomial format data as a polynomial within a ring structure converted from messages. The polynomial rings is $\mathbb{Z}[X]/(X^n + 1)$. The $\mathbb{Z}$ means the set of integers, $X$ means the variable in the polynomial and $n$ is the degree of the polynomial rings. After encoding the message into plaintext, the HE can encrypt and do further computations.

**Encoding/Decoding**: Encoding is the process of transforming the message into plaintext which can be encrypted later in HE. Decoding is the opposite operation that converts the plaintext into the message. In the CKKS scheme, a set of messages can be packed together and then encoded and encrypted [5]. For instance, A message contains three decimal numbers, [2.12, 3.19, 6.12]. All three elements in the message can be packed into one plaintext but in different slots. The number of slots is half of the polynomial modulus degree of the CKKS scheme [5].

**Rotation**: A technique to switch the element in the slots in ciphertext/plaintext clockwise or counterclockwise.

**Level**: The number of remaining modular reductions (rescaling operations) that a ciphertext can do before the ciphertext cannot be decrypted

correctly.

**Key Generation**: For CKKS scheme, five types of keys are required. The secret key ($sk$) is for decrypting the ciphertext into plaintext. The $sk$ is a polynomial with coefficients. The public key ($pk$) is for encrypting the plaintext into ciphertext. The $pk$ consists of two polynomials $(pk_0, pk_1)$. The $pk_0 = -a \times s + e \mod q$ and $pk_1 = a \mod q$. The $a$ is a random polynomial sampled in uniform distribution and $e$ is sampled in Gaussian distribution. The $\times$ operation in the equation is convolution for each pair of polynomial rings. The evaluation key ($evk$) is applied for homomorphic computations. The rotation key ($rk$) is for rotating the slots in the ciphertext. The relinearization key ($relink$) is used for the relinearization operation, which reduces the components of the ciphertext from 3 to 2 after homomorphic multiplication to manage the size of the ciphertext.

**Encrypt/Decrypt**: Encryption is the process of encrypting the plaintext to ciphertext. The algorithm of encryption is shown as equation 2.3, where the $v$is a random polynomial from a uniform distribution, $C_t$ is the ciphertext, $P_t$ is the plaintext and the $e$ is the random noise from a Gaussian distribution. The $\times$ operation in the equation is convolution for each pair of polynomial rings.

$$C_t = v \times pk + P_t + e \tag{2.3}$$

The decryption is the verse, decrypting the ciphertext back to plaintext. The algorithm of decryption is shown as Equation 2.4, where the $q$ is the ciphertext modulus. The $\times$ operation in the equation is convolution for each pair of polynomial rings.

$$P_t = (C_t \times sk) \mod q \tag{2.4}$$

**Ciphertext**: Ciphertext is the ciphertext encrypted from plaintext.

**Homomorphic Addition/Homomorphic Multiplication**: Homomorphic addition and holomorphic multiplication are the addition and multiplication over ciphertext as we mentioned in section 2.1. Besides, the CKKS scheme supports the homomorphic addition and homomorphic multiplication between ciphertext and plaintext as we have shown in Equation 2.5 and 2.6.

$$Dec(Enc(P_t x) + Encode(y)) = x + y \tag{2.5}$$

$$Dec(Enc(P_t x) \times Encode(y)) = x * y \tag{2.6}$$

**Scale**: In the CKKS scheme, there is a factor being introduced to scale the incoming message during encoding. The factor is called the scale. This scale is useful for maintaining the precision of the message during several operations in the CKKS scheme. For example, the message=1.35 with scale $10^2$ will be encoded into around 135 as a plaintext (it is not exactly 135 because of the errors from Fast Fourier Transform (FFT)).

**Rescaling**: The scale of the result ciphertext will increase after performing homomorphic multiplication. In this situation, rescaling is performed for scale management since the ciphertext cannot be decoded correctly if the scale exceeds a threshold. For instance, assume there are two ciphertexts whose scale is $\Delta$ respectively. After one time of homomorphic multiplication, the scale of the result is $\Delta^2$. By employing rescaling, we aim to reduce the scale of the result and manage noise growth. When applying rescaling, the polynomial modulus decreases one element and the level decreases one.

**Relinearization**: Assume that the $C_t^1$ and $C_t^2$ are two ciphertexts. The ciphertext can be recognized as two components. For instance, $C_t^1$ consist of $C_{0,1}$ and $C_{1,1}$. After performing homomorphic multiplication of two ciphertexts, the number of components in the result (ciphertext) increases as shown in Equation 2.7. Relinearization can be used to reduce the number of components in the ciphertext [22].

$$\begin{aligned} C_t^1 \times C_t^2 &= (C_{0,1}, C_{1,1}) \times (C_{0,2}, C_{1,2}) \\ &= (C_{0,1} \times C_{0,2}, C_{0,1} \times C_{1,2} + C_{1,1} \times C_{0,2}, C_{1,1} \times C_{1,2}) \end{aligned} \tag{2.7}$$

## 2.3 Errors in CKKS Scheme

### 2.3.1 Approximation error of activation function

Non-linear activation functions, such as ReLU, cannot be directly implemented in FHE privacy-preserving machine learning models due to the CKKS scheme's limitations in supporting conditional branching. To address this, polynomial functions are used to approximate non-linear activation functions in FHE privacy-preserving machine learning models. However, these polynomial approximations introduce errors since they do not exactly replicate the behaviour of the original non-linear functions. This discrepancy between the polynomial approximation and the actual activation function can affect the

overall accuracy and performance of the privacy-preserving machine learning models.

### 2.3.2 Encoding Error

As we introduced in section 2.2, the encoding converts the message into plaintext which can be operated in the CKKS scheme. Furthermore, during encoding, an initial scale is employed to save the precision of the message. Thus, regarding different initial scale settings, a different amount of rescaling error is introduced and affects the precision of the message. Figure 2.2 shows an example of precision loss from the message due to the use of improper initial scaling for encoding for real numbers. In Figure 2.2, the information we want to encrypt in the message 1.2345678 is encoded with the initial scale of $10^7$. However, because the Fast Fourier Transform (FFT) is an approximation method and it introduces errors because of finite precision arithmetic, even if the message is integer, errors are introduced from encoding. After decoding with the same scale ($10^7$), the decoded result is around 1.2345678 but not exactly 1.2345678.

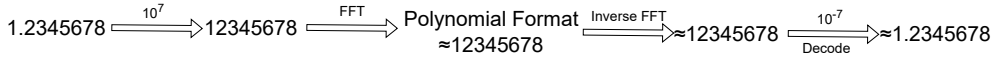



Figure 2.2: An example of encoding error

### 2.3.3 Rescaling Error

As we introduced about rescaling in section 2.2, the rescaling is important since the scale of ciphertext will increase after homomorphic multiplication. The ciphertext cannot be decrypted correctly if the scale is over the ciphertext modulus $q$. However, rescaling also introduces some errors. Figure 2.3 shows an example of an occurrence of rescaling error. Assume that there is a ciphertext 1.23456 and a ciphertext 2.34567 with initial scale $\Delta$ is $10^5$. After multiplication, the scale of the result will become $10^{10}$. Thus, we can apply rescaling to reduce the scale of the result ciphertext. The rescaling makes some errors since the CKKS scheme employs FFT to round the result to round $289,587$. As a result, a rescaling error of about $3.55 \times 10^{-7}$ is introduced.

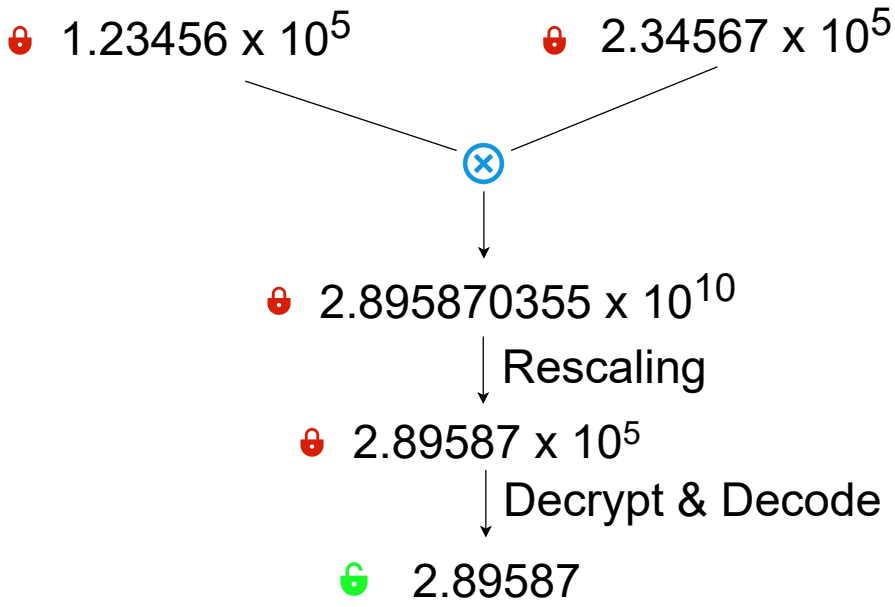



Figure 2.3: An example of rescaling error

## 2.4 GraphSAGE

A data structure called *graph* that requires researchers to consider not only individual instances but also the relationships between instances in the dataset. In this scenario, graph neural network (GNN) has the advantage compared with other traditional ML models such as convolutional neural network (CNN) since it considers the relationships [1].

There are several famous GNN modes. The graph convolutional network (GCN) [19]. The GCN aggregates features from neighbors of a node to do inference. In 2017, graph attention network (GAT) [23] was proposed, which aggregates features with attention coefficient. GraphSAGE [13] was proposed by Hamilton et al. in 2017. Compared with GCN and GAT, there are some advantages of GraghSAGE as follows:

1. **Inductive Learning**: GraphSAGE is an inductive learning model; which means that GraphSAGE is able to infer for the nodes which are never shown during the training.

2. **Neighbors Sampling**: For GCN and GAT, all nodes in the graph need to be considered for inference. However, GraphSAGE only consid-

ers the specific number of neighbors nodes, which reduces computation complexity and saves resources.

Algorithm 1 shows the algorithm of the node update in GraphSAGE [13]. The $L$ means the levels of the neighbors being considered in GraphSAGE. Each node contains $|V|$ features. For updating the feature values of each node, we need to aggregate the node's neighbors relevant feature values and we get the aggregation result as $\mathbf{h}^l_{\mathcal{N}(v)}$ at Line 3 in Algorithm ∼1. There are many aggregation methods such as mean aggregation and max aggregation. After getting the aggregation result, do a *CONCAT* operation for the feature of the node itself and aggregation result. Multiply with the concat result with $\mathbf{W}^l$ to get the update feature of the input node. Finally, let the updated feature of the node go through a non-linear activation function to get $\mathbf{h}^l_v$.

**Algorithm 1** Features updation for nodes in GraphSAGE [13]

**Input:** Level $L$; weight matrices $\mathbf{W}^l, \forall l \in \{1, \ldots, L\}$; input features at level $l$ $\{\mathbf{h}^l_v, \forall v \in \mathcal{V}\}$; non-linearity $\sigma$; aggregation function $\text{AGGREGATE}_l, \forall l \in \{1, \ldots, L\}$; neighbors finding function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$
**Output:** Vector representations of updated notes at level $l$ $\mathbf{h}^l_v$ for all $v \in \mathcal{V}$

1: **for** $l = 1 \ldots L$ **do**
2:  **for** $v \in \mathcal{V}$ **do**
3:  $\mathbf{h}^l_{\mathcal{N}(v)} \leftarrow \text{AGGREGATE}_l\left(\{\mathbf{h}^{l-1}_v, \forall v \in \mathcal{N}(v)\}\right)$
4:  $\mathbf{h}^l_v \leftarrow \sigma\left(\mathbf{W}^l \cdot \text{CONCAT}(\mathbf{h}^{l-1}_v, \mathbf{h}^l_{\mathcal{N}(v)})\right)$
5:  **end for**
6: **end for**

Figure 2.4 shows an example of a target node A to get updated by the inference of the GraphSAGE model. In Figure 2.4 example, the feature of the target node is updated by two sageLayers. Each sageLayer considers two neighbors. The activation function used in this example is $y = x$ and the weights for both sageLayers are $[1, 1]$.

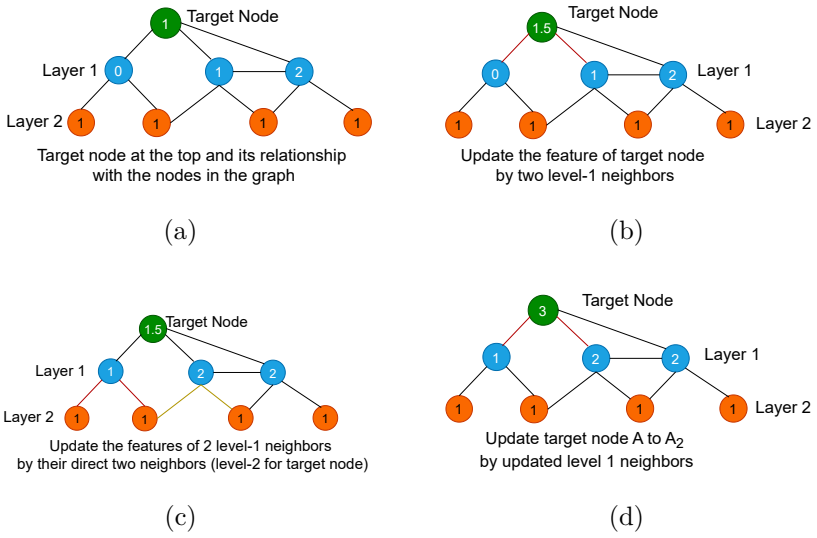



Figure 2.4: An example of inference of GraphSAGE with two layers neighbors

# Chapter 3

# Related Work

In this chapter, we present some existing works for this thesis. Section 3.1 explains existing research about privacy-preserving GNN models. Section 3.2 explains the works investigating the effect of CKKS scheme errors on the accuracy of ML models with FHE.

## 3.1 Privacy-preserving GNN

The existing research on privacy-preserving GNN mainly utilizes federated learning techniques. In 2021, Yang Pei et al. [24] proposed a decentralized federated graph neural network (D-FedGNN). D-FedGNN allows multiple participants to train a GNN without a centralized model. Yang Pei et al. have conducted their D-FedGNN and resulted that their D-FedGNN has a similar performance to the state-of-art centralized federated GNN model proposed by Chaoyang He et al. [25] but faster training time on datasets in MoleculeNet benchmark [26].

In 2022, Chaochao Chen et al. [27] proposed a vertical federated graph neural network (VFGNN) for privacy-preserving GNN. For the method proposed by Yang Pei et al. [24], the multiple participants have different parts of the nodes in the graph. However, in the method proposed by Chaochao Chen et al. [27], to protect the privacy of the graph, all clients have the same nodes but different features and partial relationships for those nodes. Chaochao Chen et al. employed a semi-honest server to infer the inference. This method requires communication among different parties.

In 2023, Songlei Wang et al. [28] proposed a privacy-preserving GNN model called SecGNN. In their scenario, the SecGNN can employ the out-

sourced cloud system by utilizing additive secret sharing to protect privacy. SecGNN uses a multi-server, decentralized trust setup where the power of the cloud is divided among three cloud servers housed by different cloud service providers in order to be compatible with the additive secret-sharing operating paradigm.

All of the methodologies proposed by previous papers require high bandwidth for communication because they all require multi-party computation protocols. By employing FHE, we can implement the computation of inference while protecting the privacy of a single party, which can obviously overcome the disadvantage of high communication consumption requirements.

Besides, In 2023, Sajadmanesh Sina proposed a method to utilize different privacy (DP) to implement privacy-preserving GNN [29]. The method in [29] introduces two types of DP, local DP and global DP. The local DP adds noise to features of nodes before training and the global DP adds noise to the aggregation step of neighbors in GNN. Compared with DP, FHE provides the ability to perform computations on ciphertext without decryption. However, the DP requires a trade-off between data utility and privacy.

The summary of previous papers is shown in Table 3.1.

Table 3.1: Summary of papers related to privacy-preserving GNN

| Paper | Published Year | Method | Advantages | Disadvantages |
|---|---|---|---|---|
| Yang Pei et al. [24] | 2021 | Federated Learning | Using key exchange to allow multiple participants to train a GNN | Require high bandwidth for communication among parties |
| Chaochao Chen et al. [27] | 2022 | Federated Learning | Parties have the same nodes but different features and partial relationships for those nodes in the graph | Require high bandwidth for communication among parties |
| Songlei Wang et al. [28] | 2023 | Federated Learning | Utilizing additive secret sharing to protect privacy | Require high bandwidth for communication among parties |
| Sajadmanesh Sina [29] | 2023 | Differential Privacy | Adding different types of noise to protect privacy, fast and reducing communication costs | DP requires a trade-off between data utility and privacy |
| This thesis | 2024 | Fully Homomorphic Encryption | Performing computations on ciphertext without decryption; reducing communication costs | High Latency |

## 3.2 CKKS-induced errors in PPML

### 3.2.1 Polynomial approximation error of ReLU

Because of the limitations in supporting conditional branching, FHE does not support the implementation of the non-linear activation functions, such as ReLU. Some of the existing research is about approximating the ReLU by the polynomial.

In 2016, Nathan Dowlin et al. [10] proposed the first privacy-preserving neural network with HE called CryptoNets. In CryptoNets, Nathan Dowlin et al. used a square function to replace the ReLU function. The approximation of ReLU function is utilized twice. Although the square function works well in CryptoNets as getting the accuracy with 99% in the experiments with the MNIST dataset. However, for a larger range of inputs, the square function does not perform well in approximating ReLU.

Therefore, papers [30, 31, 32] tried to use Taylor approximate polynomials to approximate ReLU. Although the Taylor approximation performs well in the small range, when the range gets wider, especially for negative numbers, the Taylor approximation cannot approximate the ReLU well. Besides, papers [33, 34, 35, 36] another approximation approach is least-square. The least squares approximation method finds the polynomial which has the smallest square sum error with real ReLU with a given degree. The least squares approximation method has an accurate approximation on the whole range [37].

In 2023, Junghyun Lee et al. [9] proposed a method to approximate ReLU and max-pooling functions. The method proposed in this paper is based on the composition of minimax approximate polynomials. According to the results shown in this paper, a precision parameter $\alpha$ is required for deciding the approximation precision level. The polynomial of degree must be bigger than $2^{1.0013\alpha-2.8483}$ for the proposed method in [9]. When the $\alpha$ is over 12, the polynomial approximation of ReLU has only around 1% difference against the original ReLU in the range of [-50,50]. However, for achieving high accurate approximations of ReLU, the degree of the polynomial approximation generated by the method proposed in [9] is high. According to the experiments in this paper, when the $\alpha$ is 12, the highest degree of the polynomial approximation is 15. Therefore, the approximation method proposed in [9] is not feasible for privacy-preserving GraphSAGE because of the requirement of a large number of homomorphic multiplications.

Although the existing research investigates the polynomial approximation methods on different neural network models. Because privacy-preserving GNN not only considers the node itself individually but also the relationships among the nodes, more homomorphic multiplication is required to extract the relationships among the nodes in the graph. The errors introduced in different polynomial approximations of ReLU may extend and affect the accuracy of privacy-preserving GNN with FHE inference. Therefore, investigating the effect of low-degree polynomial approximation of activation functions on the accuracy of FHE-encrypted GNN is essential for implementing privacy-preserving recommendation systems.

### 3.2.2 Encoding error and rescaling error

In 2023, Devharsh Trivedi et al. [11] implemented FHE-encrypted Logistic Regression (LR) and Support Vector Machine (SVM) for website log anomaly detection. In their experiments, they discussed the effect of different scale settings on the accuracy of LR and SVM. According to the experiment results, when the scale settings are different, the accuracy of models is affected. Especially, when the scale is $2^{40}$, the accuracy of LR with polynomial approximation activation function $A_{50}^3 = 0.49714848 + 0.026882438x - (7.728304e-6)x^3$ increased to 93.50%. However, when the scale setting is $2^{30}$, the accuracy is 91.20%.

Also in 2023, Sogo Pierra Sanon et al. [12] discussed the impact of errors in the CKKS scheme on traffic prediction in 5G wireless networks. In their experiments, Sogo Pierra Sanon et al. implemented privacy-preserving recurrent neural network inference under homomorphic parameter settings at different scales using federated learning and the CKKS scheme. According to their results, the average mean square error of the FHE encrypted recurrent neural network performs as well as the unencrypted model when the scale exceeds $2^{25}$ for the number of federated parties required for federated learning ranging from 2 to 8.

Although both papers [11, 12] discuss the effect of the error from the CKKS scheme on the accuracy of PPML, there are still some limitations that make our thesis meaningful. For example, although [11] and [12] employed different scale settings of the homomorphic parameters to investigate the error from the CKKS scheme on different PPML models, the considerations are incomplete. According to Section 2.3, both encoding and rescaling operations in the CKKS scheme will introduce errors. Thus, it is crucial

for us to separate the experiments to discuss the effect of encoding error and rescaling error individually to investigate their effect on the accuracy of FHE-encrypted ML respectively. By analyzing these errors separately, we can identify the primary reason for accuracy degradation in these two types of error. This separation allows for a more precise understanding of the error mechanisms so that we can enhance the accuracy of the FHE-encrypted ML models.

Besides, the investigation of the effect on the accuracy of integration of FHE with GNN remains underexplored. In FHE-encrypted GNN, it is vital to encrypt the graph to protect privacy (both node features and relationships), which is large and complicated. Trying to extract the encrypted information in the graph to do the inference for FHE-encrypted GNN requires more computations and operations. Therefore, it is important for us to investigate the effect of the CKKS scheme on the accuracy of FHE-encrypted GNN models. By investigating the effect of the CKKS scheme, we can identify the specific reasons for the accuracy degradation of FHE-encrypted GNN models, which is crucial for us to implement high-accuracy FHE-encrypted GNN models in the future.

The summary of previous papers is shown in Table 3.2.

Table 3.2: Summary of papers related to CKKS-induced errors in PPML

| Paper | Published Year | PPML model | Encoding Error Investigation | Rescaling Error Investigation |
|---|---|---|---|---|
| Devharsh Trivedi et al. [11] | 2023 | Linear Regression & Support Vector Machine | ✓ | |
| Sogo Pierra Sanon et al. [12] | 2023 | Recurrent Neural Network | ✓ | |
| This thesis | 2024 | GraphSAGE | ✓ | ✓ |

# Chapter 4

# Experiment Designs

In this chapter, we explain the experiments to investigate how errors introduced by the CKKS scheme[5] affect the accuracy of the FHE-encrypted GraphSAGE inference.

Section 4.1 provides a detailed description of the dataset we used in our experiments. Besides, the generation of the graph is introduced in this section.

Section 4.2 describes the pre-trained plaintext GraphSAGE model[13]. In specific, the number of SageLayer, the number of neighbors each SageLayer considers, and the activation functions employed are introduced in this section.

Section 4.3 introduces the graph and queries. How graph and queries are encrypted in our experiments by FHE, and the details of encrypted nodes and queries are explained in this section. Besides, how we extract the neighbors and aggregate them to make inferences in FHE-encrypted GraphSAGE is also included in this section.

Section 4.4 analyzes the complexity of doing mean aggregation for different levels of neighbors.

Section 4.5 details the experimental environment for doing our experiments. The devices, the version of the C++ and other external packages employed are introduced in this section.

Section 4.6 discusses the experiments conducted to investigate the impact of three types of errors introduced by the CKKS scheme [5] on the inference of FHE-encrypted GraphSAGE. This section is divided into three subsec-

tions, addressing polynomial approximation error of ReLU, encoding error and rescaling error.

Section 4.7 provides the details of the evaluation metric we used to evaluate the results of our experiments.

## 4.1 Dataset

In our experiments, we use the Cora Dataset [38]. There are 2,708 scientific publications in the Cora dataset. According to the contents of the publications, these 2,708 scientific publications can be categorized into 7 classes. Additionally, there are 5,429 connections among these publications, forming a graph. Each connection represents a citation from one publication to another, indicating a directional relationship.

For each scientific publication, there are 1433 features to explain whether the scientific publication contains a specific unique word or not. For each word, if the word is contained in a publication, the value of the feature is 1. Otherwise, the value is 0.

Table 4.1: Cora Dataset

| Dataset | Node | Edge | Features | Classes |
|---|---|---|---|---|
| Cora | 2,708 | 5,409 | 1,433 | 7 |

## 4.2 Pre-trained GraphSAGE model

The pre-trained model in our experiments uses the architecture proposed by William L. Hamilton et al. [13] called GraphSAGE. The GraphSAGE model contains two sageLayer, three activation layers, and one linear layer for a classification problem with the Cora dataset involving 7 classes.

The only difference between the GraphSAGE model and the GraphSAG model in William L. Hamilton experiments is that each sageLayer considers only 5 neighbors. The reason we chose to consider only 5 neighbors per sageLayer is that when considering only 5 neighbors, the GraphSAGE can

already have the accuracy as 86.58%. However, after increasing the number of considered neighbors for each sageLayer to 10, the accuracy only has a small improvement as 87.98%. In addition, sageLayer that considers too many neighbors will cause the size of the encrypted graph in our experiments to increase, resulting in a huge amount of memory consumption.

Figure 4.1 provides a detailed illustration of the plaintext GraphSAGE architecture for the Cora dataset. The dimensions change in the inputs and output are shown in Figure 4.1. The "a x b" means the dimension of the inputs or outputs. For instance, before going through the SageLayer1, the dimension of input is 1 x 1433, which means a query node with 1,433 features in the Cora dataset. Table 4.2 presents the configurations of the GraphSAGE model in our experiments. The workflow of the GraphSAGE making an inference by a query is as follows:

1. The sageLayer 1 samples 5 direct neighbors (Level-1 neighbors) of the input and aggregates the features of the 5 level-1 neighbors by mean. After getting the mean aggregation of level-1 neighbors, *CONCAT* operation is done with features of input and multiply with a *weight1* whose dimension is $2466 \times 128$ as shown in Algorithm 1. Besides, the sageLayer 1 does the same operations to the 5 direct neighbors (level-2 neighbors of input) of level-1 neighbors to update the features for level-1 neighbors, which is required for sageLayer 2.

2. The activation activates the inputs by ReLU function.

3. The sageLayer 2 does the *CONCAT* operations for updated input and its updated 5 level-1 neighbors. Multiplying the *CONCAT* result with the *weight2*.

4. The activation activates the result by ReLU function for the result from sageLayer2

5. The liner layer classifies the input into 7 classes. The class with the highest possibility is the predicted result.

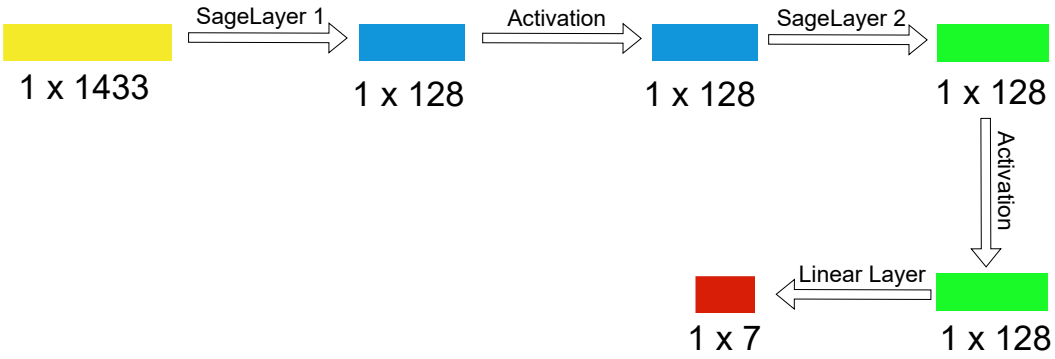



Figure 4.1: GraphSAGE architecture for Cora Dataset

Table 4.2: Pre-trained Plaintext GraphSAGE model

| Layer type | Description | Considered Neighbors | Input Dimensions | Output Dimensions |
|---|---|---|---|---|
| SageLayer 1 | Aggregating the neighbors by mean and update the node embedding values | 5 | 1,433 | 128 |
| Activation | Applying ReLU function | N/A | 128 | 128 |
| SageLayer 2 | Aggregating the neighbors by mean and update the node embedding values | 5 | 128 | 128 |
| Linear Layer | Contains 7 weights and bias to classify the node among 7 classes | N/A | 128 | 7 |

As we mentioned in our Figure 1.1, the parameters of the GraphSAGE model are not encrypted. Therefore, the parameters of SageLayer1, Sage-

Layer2 and Linear Layer remain in plaintext. Due to the limitations of FHE, which does not support conditional branches, we use a polynomial approximation to approximate the ReLU activation function. In our experiments, we used the least-square method to approximate the ReLU function [33] in the range from -3 to 3. Similar to Takumi Ishiyama et al. [33], we also used the 'curve_fit' function in 'SciPy' library [39] to approximate the ReLU by quadratic polynomial and quartic polynomial. The two polynomial approximations of ReLU are shown in Table 4.3. The *rg* means the approximation range and the *deg* means the degree of the approximation. Figure 4.2 shows the difference between ReLU-rg3-deg2, ReLU-rg3-deg4 and ReLU in the range -3 to 3.

Table 4.3: Polynomial approximation of ReLU

| Name | Degree | Range | Formula |
|---|---|---|---|
| ReLU-rg3-deg2 | 2 | [-3,3] | $0.1459x^2 + 0.4804x + 0.3033$ |
| ReLU-rg3-deg4 | 4 | [-3,3] | $-0.0107x^4 + 0.2514x^2 + 0.4859x + 0.1906$ |

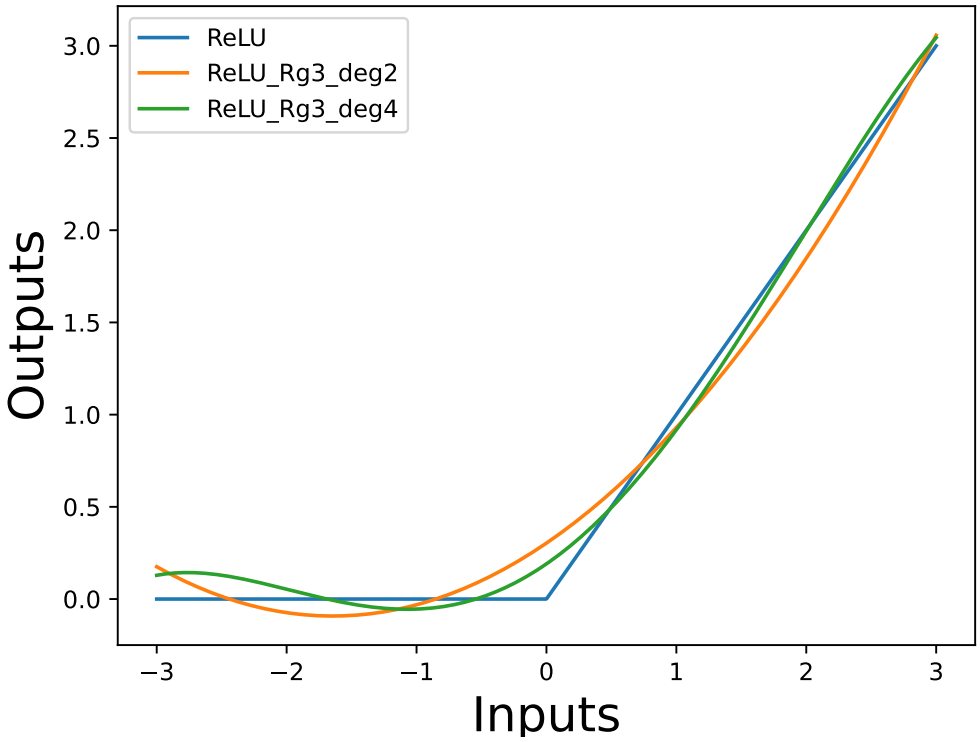



Figure 4.2: Approximation of ReLU in range -3 to 3

## 4.3 Graph and Queries

To make inferences of FHE-encrypted GraphSAGE, the graph and queries are needed as inputs. The graph is the graph-structured data that contains the numerous nodes and each node will have features and relationships with other nodes in the graph. A query consists of two parts: an adjacency matrix and node features. All information in the graph and queries are encrypted by FHE.

### 4.3.1 Graph

The graph is recognized as two parts: the features of the node itself and the features of the neighbor of the node. As an example shown in Figure 4.3, there are five nodes in the graph and each node has two features.

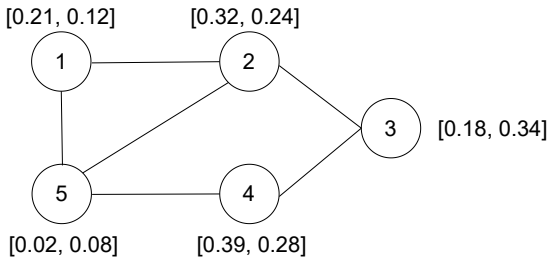



Figure 4.3: An example of a graph

For convenience in computation, we convert the graph into a graph chart according to the graph and the pre-trained GraphSAGE model. The graph chart is a 2-d vector. For example, the graph in Figure 4.3 can be converted to a graph chart as shown in Figure 4.4, where the pre-trained GraphSAGE model considers two neighbors for level-2 neighbors.

In the graph chart as shown in Figure 4.4, each row represents a node in the graph and its neighbors (2 neighbors in this example). For instance, the first row represents the features of node 1 and the features of its neighbours, node 2 and node 5 respectively. Vertically, each column in the graph chart represents one of the feature values. In the example of the graph shown in Figure 4.3, each node has two features. Therefore, in its converted graph chart as shown in Figure 4.4, the first and second columns represent the feature value of each node itself. The third and fourth columns; and the fifth and sixth columns represent the features of the first and the second picked neighbors of level 1 nodes, respectively.

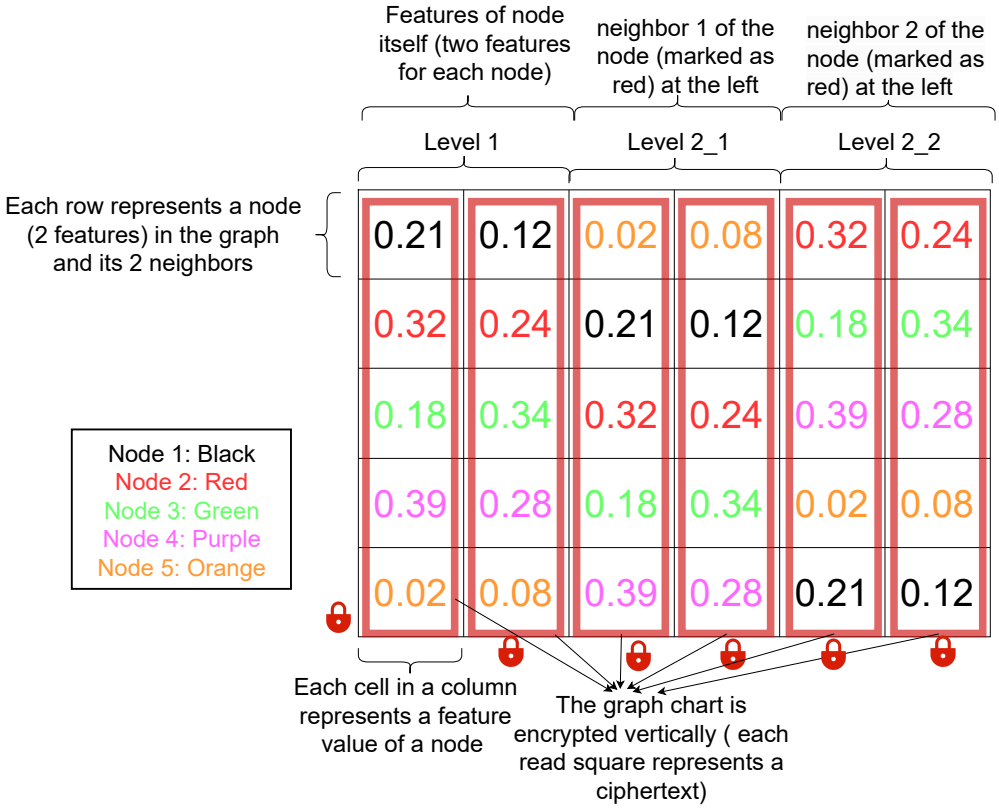



Figure 4.4: An example of a graph chart with 2 levels neighbors being considered

In the example as shown in Figure 4.4, the pre-trained GraphSAGE model considers 2 neighbors for level 1 nodes. In reality, nodes in the graph may have less or more than 2 neighbors. If the number of neighbors of a node is over 2 in the graph, we sample 2 neighbors in all of its neighbors. If the number of neighbors of a node is less than 2, we oversample the neighbor by picking the same neighbor twice [13]. The neighbors are randomly sampled in no particular order.

The equation to calculate the number of rows in the graph chart is shown in Equation 4.1. The $N_r$ represents the number of the rows in the graph chart and $N_n$ represents the number of nodes in the graph.

$$N_r = N_n \tag{4.1}$$

Besides, the equation to calculate the number of columns in the graph chart is shown in 4.2, the $N_c$ means the number of the columns in the graph

chart, the $N_f$ represents the number of the feature of each node and the $N_2$ represents the number of level-2 neighbors required by GraphSAGE model.

$$N_c = N_f * (1 + N_2) \tag{4.2}$$

Accordinig to the graph that is shown in Figure 4.3, the graph contains 5 nodes and each node has 2 features. Therefore, as shown in Figure 4.4, the graph chart has six columns in total. Because we encrypted the graph chart vertically, six ciphertexts are created. Therefore, the number of ciphertexts for the graph chart $N_{ct}^g$ can be calculated as equation 4.3.

$$N_{ct}^g = N_c = N_f * (1 + N_2) \tag{4.3}$$

According to section 4.1, the Cora dataset includes 2,708 nodes, and each node has 1,433 features. In our GraphSAGE model, we consider five level-2 neighbors for each node. Consequently, our graph chart has 8,598 columns in total. In our experiments, we encrypted the graph chart vertically, resulting in 8,598 ciphertexts to create the encrypted graph chart.

### 4.3.2 Queries

The query consists of two parts: an adjacency matrix and node features. The adjacency matrix is used to extract the relationships (whether the query node is connected with one of the nodes in the graph or not) between the query node and the nodes in the graph. As an example shown in Figure 4.5, the left one is the adjacent matrix and the right one is the features of the query node. The example of Figure 4.5 is made by a scenario in which each node has two features and the GraphSAGE model considers 2 level-1 neighbors. The adjacent matrix will be encrypted into one ciphertext and each feature of the node will be encrypted into one ciphertext for convenience. Therefore, the number of ciphertexts for a query can be calculated as equation 4.4. The $N_{ct}^q$ is the number of ciphertext for a query and the $N_f$ is the number of features.

$$N_{ct}^q = 1 + N_f \tag{4.4}$$

For the Cora dataset and the plaintext GraphSAGE model, each query contains 1,433 ciphertexts to store its features. In other words, each feature will be a ciphertext. Additionally, there is another ciphertext whose 2,708 slots have been used for storing the adjacency matrix of the query node. Since our Plaintext GraphSAGE model considers only five level-1 neighbors, only five slots in the adjacency matrix ciphertext have a value of 1, which

indicates a connection between the query node and specific nodes in the graph, while the other slots have a value of 0, which indicates no connection.

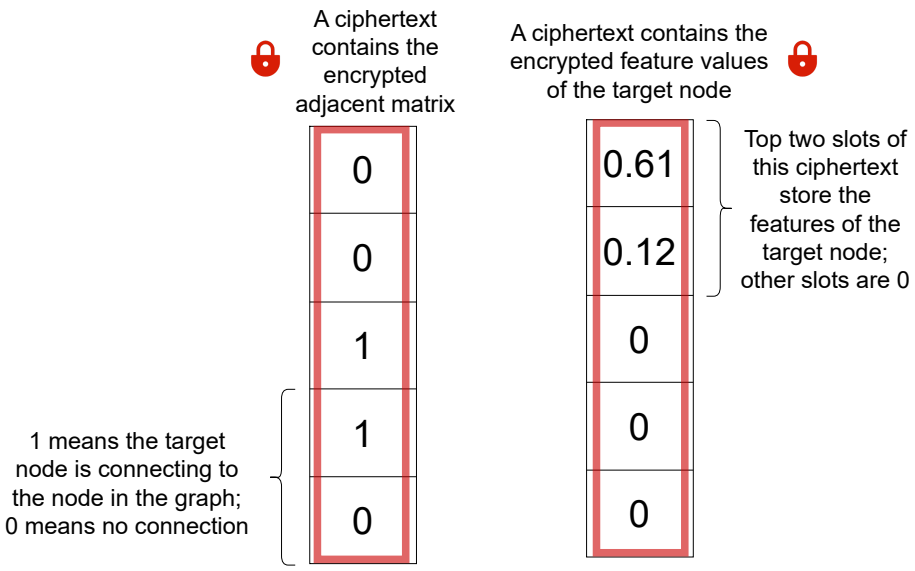



Figure 4.5: An example of a query

### 4.3.3 Neighbors Extraction and Aggregation

To extract the neighbors, we need to do homomorphic multiplication between the graph chart and the adjacent matrix of the query node. As we mentioned above, the graph chart is encrypted vertically. Thus, after a homomorphic multiplication, only the neighbors picked have values but others become 0 as shown in Figure 4.6.

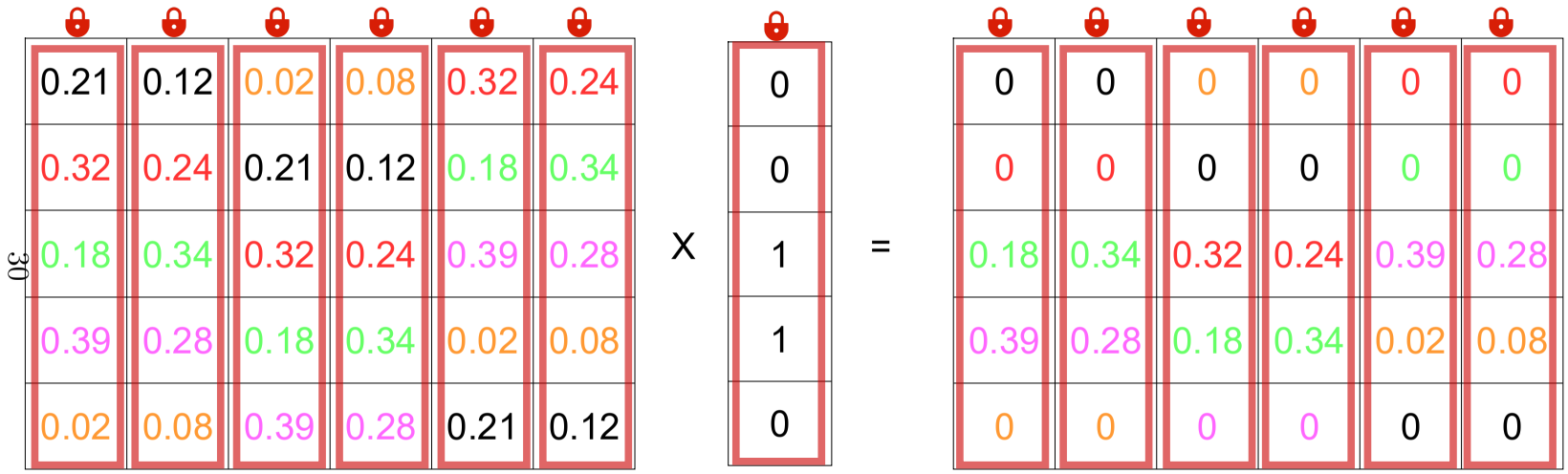



Figure 4.6: An example of extracting neighbors

After this homomorphic multiplication, we can get the level-1 neighbours' features from the first and second columns. Then we can use the rotation technique mentioned in Section 2.2 to get the sum of the neighbors and multiply the sum by homomorphic multiplication with a plaintext ($1/N_1$, $N_1$ is the number of the level-1 neighbors) to get the result of mean aggregation. The computation is shown as in Figure 4.7

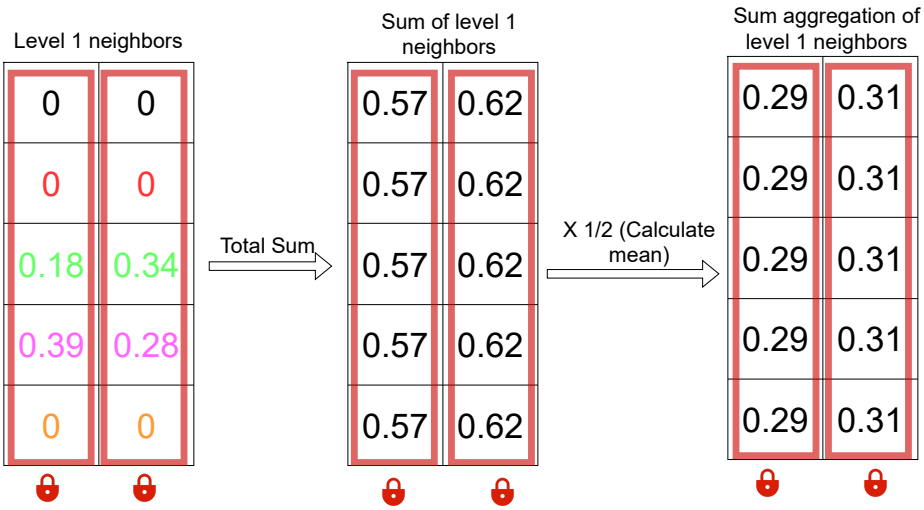
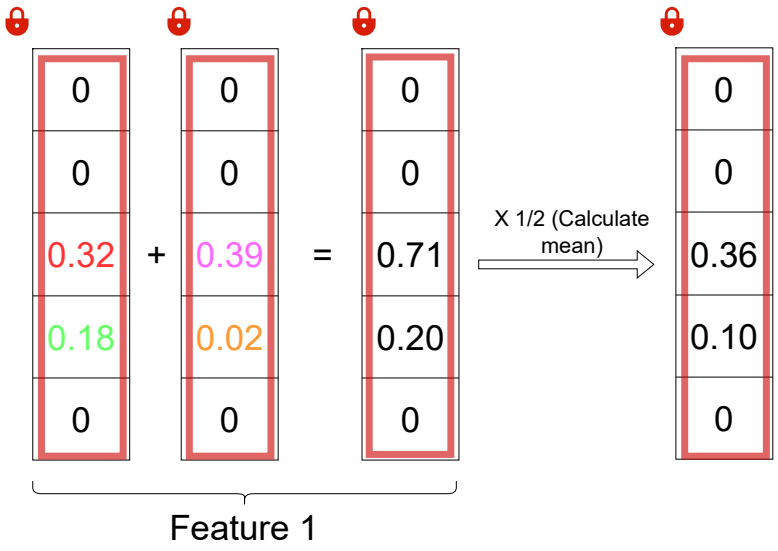
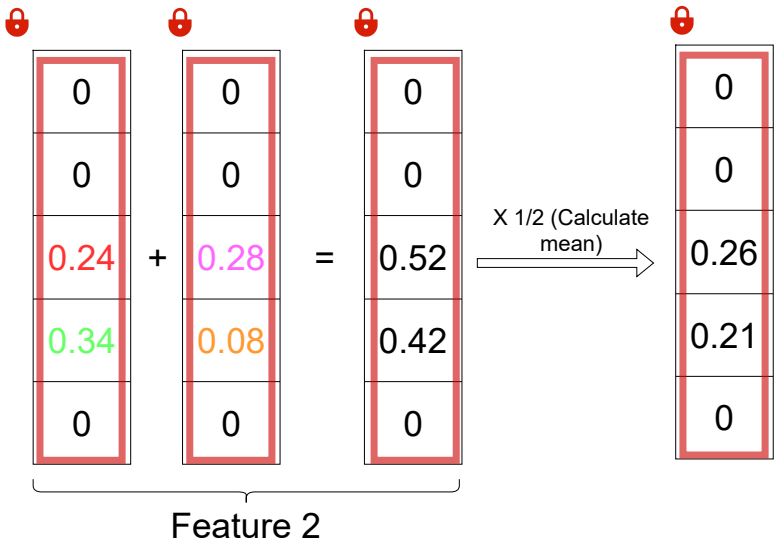



Figure 4.7: Mean aggregation of level 1 neighbors

Similar to getting the mean aggregation of level-1 neighbors. We can get the sum of the level-2 neighbors by adding the ciphertexts in the graph chart together regarding their feature index. The example is shown in Figure 4.8. After getting the summation, we can multiply the ciphertext which represents the summation of each feature with a plaintext ($1/N_2$, $N_2$ is the number of the level-2 neighbors) to get the result of mean aggregation.




(a)




(b)

Figure 4.8: Mean aggregation of level 2 neighbors

## 4.4 Complexity Analysis of Mean Aggregation

For the mean aggregation of level 1 neighbors, we need to calculate the sum of the numbers in different slots of a ciphertext. Therefore, we need to employ rotation and homomorphic addition. Specifically, times of rotation and homomorphic addition are $log_2(N_{slot}) - 1$. The $N_{slot}$ means the number of the slots in a ciphertext. After getting the sum of the level 1 neighbors, we need to do $N_f$ times homomorphic multiplication, where the $N_f$ means the number of features of a node. The computation complexity of getting mean aggregation of level 1 neighbors is $log_2(N_{slot}) + N_f$.

For the mean aggregation of level 2 neighbors, we need to do $N_f$ times homomorphic addition to get the sum of the neighbors. After getting the summation, do $N_f$ times homomorphic multiplication to get the mean aggregation of level 2 neighbors. The computation complexity of getting mean aggregation of level 2 neighbors is $N_f$.

The summary of the complexity analysis is shown in Table 4.4.

Table 4.4: Summary of complexity analysis

| Phase | Operations | Times of operations |
|---|---|---|
| Mean aggregation of level 1 neighbors | Rotations | $log_2(N_{slot}) - 1$ |
| | Multiplications over ciphertext | $log_2(N_{slot}) - 1$ |
| | Additions over ciphertext | $N_f$ |
| Mean aggregation of level 2 neighbors | Rotations | 0 |
| | Multiplications over ciphertext | $N_f$ |
| | Additions over ciphertext | $N_f$ |

## 4.5 Experiment Environment

In our experiments, we used SEAL 4.1 [40] and C++ to generate a FHE-encrypted GraphSAGE. To reduce the latency, we implement parallelization in the inference process by OpenMP. Table 4.5 lists the specifications of the machine used in the experiment.

Table 4.5: Machine Specification

| | | |
|---|---|---|
| CPU | Model Number | Intel Xeon E7-8880 v3 |
| | Frequency (base) | 2.30 GHz |
| | Frequency(turbo) | 3.10 GHz |
| | #cores/CPU | 18 |
| | Hyper-threading | Disabled |
| | L1i cache capacity/core | 32 KiB |
| | L1d cache capacity/core | 32 KiB |
| | L2 cache capacity/core | 256 KiB |
| | L3 cache capacity/CPU | 45 MiB |
| | #CPUs | 4 |
| Memory | Capacity(total) | 3TB |
| OS | | CentOS 7.9.2009 |
| Library | Microsoft SEAL | 4.1.1 |
| | Numpy | 1.16.2 |
| | Pillo | 8.3.2 |
| | Photon | 0.3.51 |
| | Pyparsing | 2.3.1 |
| | Scikit-learn | 0.20.3 |
| | Scipy | 1.2.1 |
| | Six | 1.12.0 |
| | Sklearn | 0.0 |
| | Torch | 1.0.1.post2 |
| | Torchvision | 0.2.0.post3 |
| Compiler & API | g++ | 11.2.1 |
| | OpenMP | 4.5 |

## 4.6 Experiments of investigating errors from CKKS scheme

### 4.6.1 Experiments of polynomial approximation error of ReLU

To investigate the effects of different polynomial approximation methods of ReLU on the accuracy of inference of privacy-preserving GraphSAGE, we design the controlled experiments as utilizing the two different polynomial approximations of ReLU in the implementation of privacy-preserving GraphSAGE. In one group, we use the quadratic polynomial to approximate ReLU

as $0.1459x^2 + 0.4804x + 0.3033$ as a name called ReLU-rg3-deg2 mentioned in Table 4.3. In another group, we use the quartic approximation (ReLU-rg3-deg4) of ReLU as $-0.0107x^4 + 0.2514x^2 + 0.4859x + 0.1906$. The FHE parameters in the experiments will be the same except for quartic approximation requires one more level to do one more homomorphic multiplication. The Table 4.6 shows the experiment settings in detail.

Table 4.6: Experiments settings of polynomial approximation error of ReLU

| Level | Degree of polynomial rings N | Polynomial approximation method | Scale | Bits of coefficient modulus |
|---|---|---|---|---|
| 11 | 32,768 | ReLU-rg3-deg2 | $2^{30}$ | $(60, 30 \dots 30, 30)$ |
| 12 | 32,768 | ReLU-rg3-deg4 | $2^{30}$ | $(60, 30 \dots 30, 30)$ |

For experiments in Section 4.6.2 and Section 4.6.3, we utilize the ReLU-rg3-deg2 as the activation function.

### 4.6.2 Experiments of encoding error

Here we assess the effect of encoding error on the accuracy of inference of the FHE-encrypted GraphSAGE. We run the inference of the FHE-encrypted GraphSAGE with the test data mentioned in Section 4.1.

To implement the controlled experiments for investigating the effect of encoding error on inference, we use different FHE parameters. In detail, our strategy involves changing the initial scale for encoding inputs into plaintext in Microsoft SEAL[14] so that it changes the accuracy when encrypting and decrypting. The setting we chose for our experiments is shown in Table 4.7.

Table 4.7: Experiments settings of encoding error

| Level | Degree of polynomial rings N | Slot counts | Initial scale | Bits of coefficient modulus |
|---|---|---|---|---|
| 11 | 32,768 | 16,384 | $2^{30}$ | $(60, 30 \dots 30, 30)$ |
| 11 | 32,768 | 16,384 | $2^{40}$ | $(60, 40 \dots 40, 40)$ |
| 11 | 32,768 | 16,384 | $2^{50}$ | $(60, 50 \dots 50, 50)$ |

### 4.6.3 Experiments of rescaling error

To investigate the effects of rescaling error on the accuracy of inference of FHE-encrypted GraphSAGE, we modify the rescaling times after the mul-

tiplication involving ciphertext-ciphertext or ciphertext-plaintext. Specifically, we created three experiment groups in our experiments. We set the initial scale as $2^{30}$ for three groups. We were trying to change the bits of coefficient modulus so that we could do different ways of rescaling in our experiments. For Naive Rescaling group, we set the bits of coefficient modulus as $(60, 60 \ldots 60, 60)$ and never do any rescaling in nodes inference of FHE-encrypted GraphSAGE. For No Rescaling group, we set the bits of coefficient modulus as $(60, 30 \ldots 30, 30)$ and we rescale the scale after every time of homomorphic multiplication. Lastly, for the Threshold Rescaling group, we set the coefficient modulus as $(60, 60 \ldots 60, 60)$ and we do rescaling only when the scale is over 90 bits. Table 4.8 presents the settings we use in our experiments.

Table 4.8: Experiments settings of rescaling error

| Group Name | Degree of polynomial rings N | Rescaling Strategy | Scale | Bits of coefficient modulus |
|---|---|---|---|---|
| Naive Rescaling | 32,768 | rescale every time | $2^{30}$ | $(60, 30 \ldots 30, 30)$ |
| No Rescaling | 32,768 | without rescaling | $2^{30}$ | $(60, 60 \ldots 60, 60)$ |
| Threshold Rescaling | 32,768 | rescale when scale bits over 90 | $2^{30}$ | $(60, 60 \ldots 60, 60)$ |

## 4.7 Evaluation Metrics

In our experiments, we employ accuracy as our evaluation metric. As we mentioned in section 4.6, each type of error is discussed with different settings. The difference in the inference accuracy with different groups will be analyzed and discussed in the results of our experiments.

$$Accuracy = \frac{TP + TN}{N} \tag{4.5}$$

Equation 4.5 shows the definition of accuracy, where $'TP"$ is the number of correctly predicted positive instances, $'$ $TN'$ is the number of correctly predicted negative instances, and *'N'* is the total number of predicted instances.

Besides, the time used to make each time inference is also recorded in experiments in section 4.6.2 and section 4.6.3 to investigate the relationship

between different scale settings or rescaling times in the CKKS scheme and latency of making an inference.

# Chapter 5

# Evaluation Result and Discussion

## 5.1 Pre-trained GraphSAGE

For our experiments, we used 5-fold cross-validation to train and test the pre-trained GraphSAGE model. This method involves devising the whole CORA dataset into five subsets. One of the five subsets was used as the test set and the remaining four subsets were used as training set. There are 10 epochs for each time of training. Because we only focus on the inference phase, choosing one model configuration that has the best accuracy is enough for our privacy-preserving GraphSAGE experiments. After determining the model configuration that has the best accuracy 84.58% during the cross-validation phases, we used the test set of this round of cross-validation as the test set for FHE-encrypted GraphSAGE inference in our further experiments.

## 5.2 Experiments results of polynomial approximation error of ReLU

The experimental results of investigating the polynomial approximation error of ReLU on the accuracy of the inference of the privacy-preserving GraphSAGE are shown in Table 5.1. According to the results shown in Table 5.1, when employing ReLU-rg3-deg2, the privacy-preserving GraphSAGE has the lower accuracy as 75.40% compared with privacy-preserving GraphSAGE employing ReLU-rg3-deg4 whose accuracy is 76.12%. This result is expected since ReLU-rg3-deg4 is more accurate to approximate ReLU in the range -3 to 3 as we have shown in Figure 4.2. However, the latency of privacy-preserving

GraphSAGE employing RuLU-rg3-deg4 is higher as 6.89 hours for one inference. This result is also expected since employing ReLU-rg3-deg4 requires one more homomorphic multiplication.

Table 5.1: pre-trained GraphSAGE and experimental results of using polynomial approximation of ReLU on privacy-preserving GraphSAGE

| Bits of coefficient modulus | Model and activation function | Scale | Rescaling Strategy | Accuracy | Latency for one inference |
|---|---|---|---|---|---|
| N/A | pre-trained GraphSAGE (ReLU) | N/A | N/A | 84.58% | 0.6s |
| 60,30...30,30 | privacy-preserving GraphSAGE (ReLU-rg3-deg2) | $2^{30}$ | rescale every time | 75.40% | 5.85h |
| 60,30...30,30 | privacy-preserving GraphSAGE (ReLU-rg3-deg4) | $2^{30}$ | rescale every time | 76.12% | 6.89h |

## 5.3 Experimental results of encoding error

The experimental results of investigating the encoding error on the accuracy of the inference of the FHE-encrypted GraphSAGE are shown in Table 5.2. According to the results in the table, while the scaling is increasing, the accuracy is also getting higher. Especially, when the scale is set as $2^{50}$, our FHE-encrypted GraphSAGE model can have an accuracy as 76.55%. This is in line with our expectations.

When the scale setting is higher, the precision of ciphertexts will increase, which will improve the accuracy performance of the FHE-encrypted GraphSAGE. High scale setting is beneficial for making inferences for cases where the difference between class possibilities is on the order of decimals. The FHE-encrypted GraphSAGE calculates the possibility of each class (7 classes

in Cora dataset) as input and outputs the class with the highest possibility as the prediction result. The higher the scale setting, the more digits can be accurately calculated for each class of possibility, thus classify the input more accurately. However, since higher precision of ciphertext is considered, the computation consumption also increases. Therefore, the latency of the experiment with scale setting as $2^{50}$ also has the biggest latency, 11.4 minutes more compared with the scale setting as $2^{30}$ for each inference.

Table 5.2: Summary of experimental results of encoding error

| Bits of coefficient modulus | Rescaling Strategy | Activation function | Scale | Accuracy | Latency for one inference |
|---|---|---|---|---|---|
| 60,30...30,30 | rescale every time | ReLU-rg3-deg2 | $2^{30}$ | 75.40% | 5.85h |
| 60,40...40,40 | rescale every time | ReLU-rg3-deg2 | $2^{40}$ | 75.81% | 5.91h |
| 60,50...50,50 | rescale every time | ReLU-rg3-deg2 | $2^{50}$ | 76.55% | 6.04h |

## 5.4 Experimental results of rescaling error

The experimental results of investigating the rescaling error on the accuracy of the inference of the FHE-encrypted GraphSAGE are shown in Table 5.3. For each experiment, each ciphertext conducts nine times homomorphic multiplication. As the result shows in Table 5.3, with the number of rescaling operations employed, the latency increases. In specific, for no rescaling, the latency decreases to 3.15 hours but also with the lowest accuracy at 72.21%. On the opposite, the naive rescaling group does rescaling operations after every homomorphic multiplication, which has the highest latency of 5.85 hours but also the highest accuracy of 75.40%. The threshold rescaling group does the rescaling operation only when the scale is over 90, The threshold rescaling group has the middle latency among the three groups in our experiments at 4.62 hours and also the middle level of the accuracy as 73.92%.

Table 5.3: Summary of experimental results of rescaling error

| Group Name | Bits of coefficient modulus | Rescaling Strategy | Scale | Accuracy | Latency for one inference |
|---|---|---|---|---|---|
| Naive Rescaling | $(60, 30 \ldots 30, 30)$ | rescale every time | $2^{30}$ | 75.40% | 5.85h |
| No Rescaling | $(60, 60 \ldots 60, 60)$ | without rescaling | $2^{30}$ | 74.21% | 3.15h |
| Threshold Rescaling | $(60, 60 \ldots 60, 60)$ | rescale when scale bits over 90 | $2^{30}$ | 74.92% | 4.62h |

## 5.5 Summary of Experimental Results

According to the experimental results we have, the CKKS-induced error affects the accuracy of the privacy-preserving machine learning GraphSAGE with FHE. For the errors introduced by the polynomial approximation of ReLU, the higher degree of polynomial approximation results in higher accuracy but also increases latency because more times of homomorphic multiplication are needed. In our experiments, the latency increases by 0.93 hours with 0.72% accuracy increasing. For rapid response applications, latency is very important for customers, the ReLU-rg3-deg2 which has shorter latency is more suitable.

The encoding error introduced by the CKKS scheme also has effects on the accuracy of the privacy-preserving machine learning GraphSAGE with FHE. Higher scale setting in the homomorphic parameters results in higher accuracy but also increases latency. According to our experimental results, increasing the scale setting leads to a significant accuracy improvement, while the increase in latency is not very large compared with a higher degree of polynomial approximation of ReLU. The highest scale setting $2^{50}$ results in a 1.15% improvement in accuracy but only increases the latency by 0.19 hours compared with the scale setting as $2^{30}$.

The different rescaling strategy has a huge effect on the latency of in-

ference but a small effect on the accuracy of inference. If we do threshold rescaling, the latency decreases by 1.23 hours but the accuracy only decreases 0.48%. Therefore, to decrease the latency, employing fewer rescaling operations is a good option since it has a small effect on the accuracy of inference for privacy-preserving GraphSAGE.

# Chapter 6

# Conclusion

GraphSAGE, one of the most famous Graph neural network (GNN) model proposed by William L. et al. [13], performs well in recommendation systems because it considers not only individual instances but also the relationships between the instances [1]. However, GraphSAGE requires customers' private information to make recommendations. In this scenario, fully homomorphic encryption (FHE) can be a useful tool since it supports privacy-preserving computation. Although the CKKS scheme [5] in FHE has been widely used in privacy-preserving machine learning (PPML) because of its advantage of approximate arithmetic [6], it is vital to investigate how the errors affect the accuracy of FHE-encrypted GraphSAGE inference.

In this thesis, we first proposed the FHE-encrypted GraphSAGE model for inference. In our scenario, the nodes graph is encrypted, which means all the features of each node and the relationship among nodes are hidden by FHE. Besides, the information of the query such as features and relationship with the nodes in the nodes graph are also encrypted by FHE. This scenario can ensure that privacy can be protected during recommendation creation by FHE-encrypted GraphSAGE. Furthermore, we do experiments to investigate the effect of encoding and rescaling techniques on the accuracy of the proposed FHE-encrypted GraphSAGE, respectively.

The results show that a higher degree of polynomial approximation results in higher accuracy but increases the latency most because of more times of homomorphic multiplications. Although the encoding error has effects on the accuracy of the inference, the increased latency by using a higher scale setting is less than using a higher degree of polynomial approximation but also the accuracy increases more. Using fewer rescaling operations has the most effect on decreasing latency. Therefore, to increase the accuracy of the

inference, we can consider increasing the scale setting first. To reduce the latency of the inference, employing fewer rescaling operations is a good method.

In future, we want to explore our research on comparing the impact of CKKS-induced error on the accuracy of inference of various privacy-preserving machine learning models with FHE such as the models mentioned in Section 3.2 with our proposed privacy-preserving GraphSAGE model. By reproducing the models mentioned in Section 3.2 to compare with the privacy-preserving GraphSAGE model, we hope to obtain clearer insights into whether CKKS-induced errors have a more significant effect on the privacy-preserving GraphSAGE model. Besides, the encryption error in the CKKS scheme is also a potential area for us to discuss in our future research because encryption error is also another important type of error that is introduced by the CKKS scheme during encryption.

# Acknowledgement

First of all, I would like to express my deepest gratitude to my advisor, Prof. Yamana. Thank you for giving me the opportunity to study in Japan. I have always considered myself an unintelligent student, but your patience and kindness in offering valuable advice have been immensely appreciated. The professional knowledge and life lessons you have shared with me will benefit me throughout my life.

I want to say thank you to my lab members. Thank you all for being with me through these fulfilling and wonderful two years. I am especially grateful to Suzuki-san for your support of my research. I deeply admire your passion for research and your dedication to guiding the juniors. Thanks to the SC team members for our countless discussions over the past two years. You are not only co-workers but also comrades who have overcome challenges with me. I wish you all the best in the future. I also appreciate our Chinese members; whenever I missed my home in China, you brought the warmth of home to me and eased my homesickness. Thank you to every member of 51-06. Despite our different backgrounds and sometimes there are language barriers because of my limited English or Japanese, you have treated me as a true friend. I am sincerely grateful to both the seniors who have graduated from the lab and my juniors for welcoming me. You have made my life over the past two years rich and colorful.

I want to say thank you to all of my friends. If there is one thing I feel truly blessed by, it is having so many friends who love and support me. Thank you for always encouraging and comforting me through difficulties and letting me know you are always there for me. I want to express my heartfelt love for you. Although I have not seen some of you for years due to distance, I promise I will visit you this year, no matter where you are. Especially to my childhood friends, those I have known since kindergarten or elementary school, thank you for being with me throughout my school years. I value our friendship above all else and am grateful that you chose me being

your childhood friend. I also want to thank those friends with whom I used to be very close but now don't stay in touch as frequently. If I have ever done anything that made you sad, please forgive me. I apologize on behalf of my younger, more immature self. Thank you for enriching my experiences and helping me become who I am today.

I want to say thank you to my family. Thank you for always supporting my decisions, whether through emotional or financial support. You have always allowed me to choose my own path and pursue what I want to do. I feel fortunate to be your family in this lifetime, and if there is a next life, I would like to meet you again without any hesitation. As I see myself growing older, I also see you aging and my younger brother growing up. I hope you can always be happy and healthy. It is now my turn to protect you. Thank you to my grandmother; your selfless love showed me what true love is for the first time. I hope you stay healthy forever. I will be with you to enjoy each day, and I will always love you.

Lastly, I want to say thank you to myself. After over 20 years as a student, through both highs and lows, thank you for persisting. Thank you for choosing to be a kind person and a good student. As you approach 25 and prepare to leave student life behind, I hope you continue to maintain the kindness and purity you have today. Treat everyone around you with sincerity and kindness, and cherish every moment. While wealth and fame have never been your goals, I hope you live a joyful and regret-free life. Thank you for shaping who I am today.

Thank you so much to everyone who encourages and supports me. I am incredibly satisfied with my student life and can't wait to start the new phase of my life!

# References


[1] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[2] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim, and J.-S. No, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *IEEE Access*, vol. 10, pp. 30 039–30 054, 2022.

[3] T. Ishiyama, T. Suzuki, and H. Yamana, "Latency-aware inference on convolutional neural network over homomorphic encryption," in *Proceedings of Information Integration and Web Intelligence: 24th International Conference, IiWAS 2022, Virtual Event, November 28-30, 2022*. Berlin, Heidelberg: Springer-Verlag, 2022, p. 324-337. [Online]. Available: https://doi.org/10.1007/978-3-031-21047-1_27

[4] S. Park, J. Byun, and J. Lee, "Privacy-preserving fair learning of support vector machine with homomorphic encryption," in *Proceedings of the ACM Web Conference 2022*, ser. WWW '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 3572-3583. [Online]. Available: https://doi.org/10.1145/3485447.3512252

[5] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology–ASIACRYPT 2017: Proceedings of 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Part I 23*. Springer, 2017, pp. 409–437.

[6] K. Li and R. Huang, "A ckks-based privacy preserving extreme learning machine," *International Journal of Information Security*, vol. 24, no. 1, pp. 166–175, 2022.

[7] S. Panda, "Principal component analysis using CKKS homomorphic encryption scheme," Cryptology ePrint Archive, Paper 2021/914, 2021, https://eprint.iacr.org/2021/914. [Online]. Available: https://eprint.iacr.org/2021/914

[8] A. Kim, A. Papadimitriou, and Y. Polyakov, "Approximate homomorphic encryption with reduced approximation error," in *Cryptographers' Track at the RSA Conference*, vol. 13161. Springer, 2022, pp. 120–144.

[9] J. Lee, E. Lee, J.-W. Lee, Y. Kim, Y.-S. Kim, and J.-S. No, "Precise approximation of convolutional neural networks for homomorphically encrypted data," *IEEE Access*, vol. 11, pp. 62 062–62 076, 2023.

[10] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, p. 201-210.

[11] D. Trivedi, A. Boudguiga, N. Kaaniche, and N. Triandopoulos, "Sigml++: Supervised log anomaly with probabilistic polynomial approximation," *Cryptography*, vol. 7, no. 4, p. 372-388, 2023.

[12] S. P. Sanon, C. Lipps, and H. D. Schotten, "Fully homomorphic encryption: Precision loss in wireless mobile communication," in *Proceedings of 2023 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, 2023, pp. 466–471.

[13] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 1025-1035.

[14] "Microsoft SEAL (release 4.0)," https://github.com/Microsoft/SEAL, Mar. 2022, microsoft Research, Redmond, WA.

[15] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, ser. EUROCRYPT'99. Berlin, Heidelberg: Springer-Verlag, 1999, p. 223-238.

[16] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469–472, 1985.

[17] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, STANFORD UNIVERSITY, Stanford, CA, USA, 2009, aAI3382729.

[18] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Trans. Comput. Theory*, vol. 6, no. 3, jul 2014. [Online]. Available: https://doi.org/10.1145/2633600

[19] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *CoRR*, vol. abs/1609.02907, 2016. [Online]. Available: http://arxiv.org/abs/1609.02907

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

[21] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ser. ITCS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 309-325. [Online]. Available: https://doi.org/10.1145/2090236.2090262

[22] S. Halevi and V. Shoup, "Algorithms in HElib," Cryptology ePrint Archive, Paper 2014/106, 2014, https://eprint.iacr.org/2014/106. [Online]. Available: https://eprint.iacr.org/2014/106

[23] P. Velikovi, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018. [Online]. Available: https://arxiv.org/abs/1710.10903

[24] Y. Pei, R. Mao, Y. Liu, C. Chen, S. Xu, F. Qiang, and B. E. Tech, "Decentralized federated graph neural networks," in *Proceedings of International workshop on federated and transfer learning for data sparsity and confidentiality in conjunction with IJCAI*, 2021, pp. 1–7.

[25] C. He, K. Balasubramanian, E. Ceyani, C. Yang, H. Xie, L. Sun, L. He, L. Yang, P. S. Yu, Y. Rong *et al.*, "Fedgraphnn: A federated learning system and benchmark for graph neural networks," *arXiv preprint arXiv:2104.07145*, 2021.

[26] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande, "Moleculenet: a benchmark for molecular machine learning," *Chemical science*, vol. 9, no. 2, pp. 513–530, 2018.

[27] C. Chen, J. Zhou, L. Zheng, H. Wu, L. Lyu, J. Wu, B. Wu, Z. Liu, L. Wang, and X. Zheng, "Vertically federated graph neural network for privacy-preserving node classification," *arXiv preprint arXiv:2005.11903*, 2020.

[28] S. Wang, Y. Zheng, and X. Jia, "Secgnn: Privacy-preserving graph neural network training and inference as a cloud service," *IEEE Transactions on Services Computing*, vol. 16, no. 04, pp. 2923–2938, jul 2023.

[29] S. Sajadmanesh, "Privacy-preserving machine learning on graphs," EPFL, Tech. Rep., 2023.

[30] W. Liu, F. Pan, X. A. Wang, Y. Cao, and D. Tang, "Privacy-preserving all convolutional net based on homomorphic encryption," in *Advances in Network-Based Information Systems: The 21st International Conference on Network-Based Information Systems (NBiS-2018)*. Springer, 2019, pp. 752–762.

[31] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff, "Privacy-preserving classification on deep neural network," *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 35, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:42011253

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

[33] T. Ishiyama, T. Suzuki, and H. Yamana, "Highly accurate cnn inference using approximate activation functions over homomorphic encryption," in *Proceedings of 2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 3989–3995.

[34] A. Kim, Y. Song, M. Kim, K. Lee, and J. H. Cheon, "Logistic regression model training based on the approximate homomorphic encryption," *BMC medical genomics*, vol. 11, pp. 23–31, 2018.

[35] K. Han, S. Hong, J. H. Cheon, and D. Park, "Logistic regression on homomorphic encrypted data at scale," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 9466–9471.

[36] J. Chiang, "Privacy-preserving logistic regression training with a faster gradient variant," *arXiv preprint arXiv:2201.10838*, 2022.

[37] J. Chiang, "A simple solution for homomorphic evaluation on large intervals," 2024. [Online]. Available: https://arxiv.org/abs/2405.15201

[38] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Automating the construction of internet portals with machine learning," *Information Retrieval*, vol. 3, pp. 127–163, 2000.

[39] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[40] "Microsoft SEAL (release 4.1)," https://github.com/Microsoft/SEAL, Jan. 2023, microsoft Research, Redmond, WA.