


PrivGNN: High-Performance Secure Inference for Cryptographic Graph Neural Networks

Fuyi Wang^{1,2}, Zekai Chen³, Mingyuan Fan⁴, Jianying Zhou², Lei Pan¹, and Leo Yu Zhang⁵

¹ Deakin University, Geelong, Australia

² Singapore University of Technology and Design, Singapore

³ Fuzhou University, Fuzhou, China

⁴ East China Normal University, Shanghai, China

⁵ Griffith University, Gold Coast, Australia

leo.zhang@griffith.edu.au

Abstract. Graph neural networks (GNNs) are powerful tools for analyzing and learning from graph-structured (GS) data, facilitating a wide range of services. Deploying such services in privacy-critical cloud environments necessitates the development of secure inference (SI) protocols that safeguard sensitive GS data. However, existing SI solutions largely focus on convolutional models for image and text data, leaving the challenge of securing GNNs and GS data relatively underexplored. In this work, we design, implement, and evaluate PrivGNN, a lightweight cryptographic scheme for graph-centric inference in the cloud. By hybridizing additive and function secret sharings within secure two-party computation (2PC), PrivGNN is carefully designed based on a series of novel 2PC interactive protocols that achieve $1.5\times \sim 1.7\times$ speedups for linear layers and $2\times \sim 15\times$ for non-linear layers over state-of-the-art (SotA) solutions. A thorough theoretical analysis is provided to prove PrivGNN’s correctness, security, and lightweight nature. Extensive experiments across four datasets demonstrate PrivGNN’s superior efficiency with $1.3\times \sim 4.7\times$ faster secure predictions while maintaining accuracy comparable to plain-text graph property inference.

Keywords: Function secret sharing · Additive secret sharing · Secure inference · Graph neural networks.

1 Introduction

Recently, the rapid development of graph neural network (GNN) techniques has significantly impacted various domains such as drug discovery [18], social networks [33], and recommendation systems [17]. For example, pharmaceutical enterprises can train the specific GNNs [8] on known drug compounds to predict the binding potential of new molecules with target proteins, such as those related to cancer. These well-trained GNNs can then be offered as AI-driven services on a pay-as-you-go basis, allowing end-users to enhance drug discovery or analyze complex molecular data. However, graph-structured (GS) data often represents

sensitive and proprietary information, making end-users hesitant to outsource it to service providers due to privacy concerns. Also, highly specialized GNNs for graph-centric services are considered valuable intellectual property and should be protected against leakage while circumventing reverse engineering risks [34].

In response to these concerns, privacy-preserving deep learning (PPDL) schemes have been developed to enable secure inference (SI) using cryptographic techniques such as homomorphic encryption (HE) [25] and/or multi-party computation (MPC) [9]. However, HE-based schemes often suffer from heavy computational and communication overhead due to inefficient exponentiation and ciphertext expansion. As a result, recent efforts focus on leveraging MPC [9] for SI with secret-shared neural networks (NNs) and inputs, offering a more efficient alternative [22]. Advanced SI solutions [13,30,36], have successfully secured convolutional neural networks (CNNs) for unstructured data (e.g., images and text). However, SI of GNNs in the cloud for complex GS data, while explored in some initial studies [26,28,33,35], remains relatively underdeveloped.

Related works and challenges. Existing cryptographic GNN approaches primarily introduce HE [26,28] and MPC [33] to secure node features in graph data. However, they do not fully protect the relational structure of graphs, such as the maximum node degree [33] and the range of node degrees [28] (**Challenge ①**). For instance, while SecGNN [33] optimizes node representation by reducing the adjacency matrix size from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \cdot d_{max})$ —resulting in over 90% reduction for real-world datasets—it leaks the maximum degree d_{max} in the graph. Moreover, these approaches [26,28,33] often rely on cryptographic primitives that are computationally and communication-intensive, particularly for non-linear operations (**Challenge ②**). Non-linear operations, such as ReLU, implemented with primitives like garbled circuits in MPC or HE, are several orders of magnitude more expensive than linear layers in computation and communication [37]. While some optimizations have been proposed for non-linear operations [13,30,37], these approaches often impose heavy online computational burdens or require multiple communication rounds. Such constraints pose challenges to scalability and efficiency for real-world graph data containing millions or even billions of nodes and edges. Lastly, recent efforts like SecMPNN [18] and OblivGNN [35] optimize non-linear operations via lightweight secret sharing primitives in a fully outsourced setting, where a group of cloud servers jointly perform SI over encrypted/shared GNNs and inputs. However, this setup raises concerns about cloud-server collusion, relying on a strong non-collusion assumption for security models (**Challenge ③**). SecMPNN [18] also requires continuous third-party assistance during online phases, adding additional complexity.

To tackle the above challenges, we design, implement, and evaluate PrivGNN, a high-performance cryptographic scheme for secure GNN inference with a client-server setup. In this secure two-party computation (2PC) setup, PrivGNN distributes the SI workload between the client and the service provider, mitigating cloud-collusion risks and enhancing security. To improve efficiency, PrivGNN employs lightweight additive secret sharing (AddSS) and function secret sharing (FuncSS) in an offline-online paradigm, significantly reducing the computational

overhead during the online phase. By carefully designing the offline phase, the online phase is reduced to only one round of interaction with lightweight computation. Specifically, we develop and optimize a line of secure protocols for both linear and non-linear computations. These protocols integrate seamlessly into various layers across various graph-centric services, reducing the online SI latency. In summary, our contributions are threefold.

- We present PrivGNN, a fast SI scheme for graph-centric services via the delicate synergy of GNNs and cryptography. PrivGNN employs lightweight AddSS to securely protect the graph structure, node features, and well-trained GNNs, without relying on a strong non-collusion assumption in the 2PC setup.
- With AddSS and FuncSS, we propose a set of secure protocols for multiplication, comparison-based activations, and sigmoidal activation variants. Our designs shift the computational-intensive operations to the offline phase and streamline online communication with only one-round interaction.
- We formally prove the correctness, efficiency, and security of PrivGNN. We conduct extensive experiments showing that our protocols outperform SotA works. Results on various datasets highlight PrivGNN’s efficiency and scalability in applications like image classification and molecular property recognition.

2 Preliminaries

Additive secret sharing (AddSS) [6] divides a secret message over the plaintext space into multiple shares, each of which is distributed to different parties. AddSS’s key property is that the shares can be combined (added) together to reconstruct the original secret, while no individual shareholder possesses adequate information to determine the secret message independently. This paper adopts the 2-out-of-2 AddSS over the ring \mathbb{Z}_{2^l} and the definition is given below.

Definition 1. A 2-out-of-2 AddSS scheme over the ring \mathbb{Z}_{2^l} is a pair of probabilistic polynomial-time (PPT) algorithms $\{\text{Share}, \text{Reconstruct}\}$ where

- **Share:** On the secret message $x \in \mathbb{Z}_{2^l}$, Share outputs shares $\{\langle x \rangle_0, \langle x \rangle_1\} \in \mathbb{Z}_{2^l}$, s.t. $x = \langle x \rangle_0 + \langle x \rangle_1 \pmod{2^l}$.
- **Reconstruct:** With 2 shares $\{\langle x \rangle_0, \langle x \rangle_1\} \in \mathbb{Z}_{2^l}$, Reconstruct outputs x over the plaintext space \mathbb{Z} .

In the case of $l > 1$ (e.g., $l = 32$) which supports arithmetic operations (e.g., addition and multiplication), the arithmetic share pair is denoted by $\langle \cdot \rangle_\gamma$ ($\gamma \in \{0, 1\}$). In the case of $l = 1$ which supports Boolean operations XOR (\oplus), NOT (\neg) and AND (\otimes), the Boolean share pair is denoted by $[\cdot]_\gamma$. In the following, we assume all arithmetic operations to be performed in the ring \mathbb{Z}_{2^l} (i.e., all arithmetic operations are mod 2^l).

Function secret sharing (FuncSS) [3, 4] within the 2PC setup divides a function $f : \mathbb{G}_{\text{in}} \rightarrow \mathbb{G}_{\text{out}}$ into 2 shares $\{f_0, f_1\}$, where \mathbb{G}_{in} and \mathbb{G}_{out} are input and output groups. Each party receives one of the function shares, and for any input x , there exists $f_0(x) + f_1(x) = f(x)$. The definition of FuncSS is given below.

Definition 2. A 2PC FuncSS scheme over the ring \mathbb{Z}_{2^l} is a pair of algorithms $\{\text{Gen}, \text{Eval}\}$ where

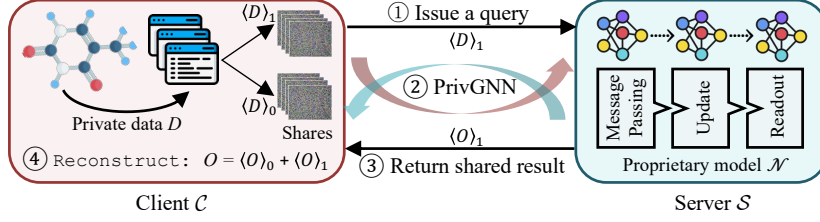


Fig. 1. A client-server secure inference scenario.

- **Gen:** With the security parameter κ and a function f , the PPT key generation algorithm $\text{Gen}(1^\kappa, f)$ outputs a pair of keys $\{k_0, k_1\}$, where each key implicitly represents $f_\gamma : \mathbb{G}_{\text{in}} \rightarrow \mathbb{G}_{\text{out}}$.
- **Eval:** With the party identifier $\gamma \in \{0, 1\}$, the key k_γ (key defining f_γ), and the public input $x \in \mathbb{Z}_{2^i}$, the PPT evaluation algorithm $\text{Eval}(\gamma, k_\gamma, x)$ outputs $y_\gamma \in \mathbb{Z}_{2^i}$, i.e., the value of $f_\gamma(x)$, where $f(x) = \sum_{\gamma=0}^1 y_\gamma$.

3 System Overview

System model. We consider the SI scenario with a client-server setup, where the client \mathcal{C} (e.g., a drug laboratory) holds private GS data D , and the server \mathcal{S} possesses a well-trained GNN model \mathcal{N} with private weights W , illustrated in Fig. 1. \mathcal{C} intends to utilize the model $\mathcal{N}(W, \cdot)$ to facilitate accurate results on its data D (i.e., $\mathcal{N}(W, D)$), while ensuring that \mathcal{C} 's data remains confidential. We regard $\gamma \in \{0, 1\}$ as the identifier of a party, $\gamma = 0$ represents \mathcal{C} and \mathcal{S} otherwise. Specifically, ① To ensure the privacy of raw graph D , \mathcal{C} uses AddSS to split D into two shares (i.e., $D = \langle D \rangle_0 + \langle D \rangle_1$), then sends the share $\langle D \rangle_1$ to \mathcal{S} to issue a SI query. ② \mathcal{C} and \mathcal{S} collaboratively take charge of SI tasks via utilizing a series of secure computation protocols in PrivGNN. ③ After execution of PrivGNN, \mathcal{S} returns the shared inference result $\langle O \rangle_1$ to \mathcal{C} . ④ Upon receiving $\langle O \rangle_1$, the client \mathcal{C} reconstructs to get the final plaintext result via $O = \langle O \rangle_0 + \langle O \rangle_1$ to complete the prediction and inference services.

Threat model. We assume that \mathcal{C} and \mathcal{S} in PrivGNN are semi-honest, i.e., they honestly obey the specification of the protocols, yet attempt to learn auxiliary information from intermediate shares. Non-collusion is unnecessary to assume, as collusion would imply a willingness to disclose their private data. The service provider (i.e., \mathcal{S}), operated by reputable vendors like Google, is incentivized to follow the protocols, as poor performance or violations of privacy regulations would have an irreversible negative impact on their credibility and profitability. Therefore, the semi-honest assumption is practical, as witnessed in existing secure outsourcing works [5, 18, 24, 30, 33]. We define security in the real/ideal world. In the real world, protocol Π is executed, yielding the output $\text{Real}_A^\Pi(1^\kappa)$, while in the ideal world, the ideal functionality \mathcal{F} is executed by a trusted third party, yielding the output $\text{Ideal}_{\text{Sim}}^\mathcal{F}(1^\kappa)$. Here, A is a stateful adversary, Sim is a stateful simulator, and $\kappa \in \mathbb{N}^+$ is the security parameter.

Definition 3. Let $\mathcal{F} = \{\mathcal{F}_0, \mathcal{F}_1\}$ be a ideal-world function and Π be a real-world protocol computing \mathcal{F} . For any PPT adversary \mathcal{A} in the real world, there exists a PPT simulator Sim in the ideal world, such that for corrupted party $\mathcal{P} \subset \{\mathcal{S}, \mathcal{C}\}$

$$\text{Ideal}_{\text{Sim}_{\mathcal{P}}}^{\mathcal{F}}(1^\kappa, D, W) \equiv_c \text{Real}_{\mathcal{A}_{\mathcal{P}}}^{\Pi}(1^\kappa, D, W). \quad (1)$$

where \equiv_c denotes computational indistinguishability against PPT adversaries except for a negligible advantage.

4 Our Approach

PrivGNN introduces the offline-online paradigm to minimize the online communication rounds and computational costs. This paradigm alleviates the computational burden on resource-constrained clients during the online phase, effectively enhancing the client experience. To achieve lightweight online performance, we use AddSS and FuncSS to design multiplication, quadratic polynomial, comparison protocols. In these protocols, the computational-intensive cryptographic operations, i.e., Gen^{BV} for generating Beaver multiplication triples [2] and Gen^{key} for generating FuncSS keys [4], are performed during the data-independent offline phase. These operations can be implemented using either a trusted third party [3, 11, 36] or specific 2PC protocols [5, 23]. This paper opts for the former one. All these protocols integrate seamlessly into NN layers.

4.1 Secure Matrix Multiplication Protocol

Given the secret-shared $\langle X \rangle$ and the server’s plaintext Y for secure matrix multiplication protocol SMatMul (i.e., $\langle Z \rangle = \langle X \times Y \rangle$), the key insight is that the output $\langle Z \rangle$ and client’s input $\langle X \rangle$ are secret-shared, while variable Y is the plaintext held by the server \mathcal{S} . We reformulate $X \times Y \rightarrow (X - A) \times Y + A \times Y$, where A is randomness held by the client \mathcal{C} . Thus, SMatMul ’s offline phase focuses on pre-computing the shares of $A \times Y$, and then the online phase evaluates $(X - A) \times Y$ in plaintext by \mathcal{S} . Compared with [11, 23], SMatMul reduces *one* unnecessary auxiliary randomness r , eliminating r -related offline computation and minimizing pseudorandom generator usage. Algorithm 1 presents the main steps and the details of SMatMul are described below.

- The offline phase aims to learn the shared $A \times Y$. Specifically, client and server sample A and B randomly, respectively, and then jointly invoke $\text{Gen}^{\text{BV}}(A, B)$ to generate matrix multiplication triples $\{A, \langle C \rangle_0\}$ for client and $\{B, \langle C \rangle_1\}$ for server, where $\langle C \rangle_0 + \langle C \rangle_1 = A \times B$. After that, the server sends $Y - B$ to the client. Next, the client locally compute $\langle A \times Y \rangle_0 = A \times (Y - B) + \langle C \rangle_0$ and the server sets $\langle A \times Y \rangle_1 = \langle C \rangle_1$.
- The online phase aims to learn the final shared $X \times Y$. The client computes $\langle X \rangle_0 - A$ and sends its result to the server. Finally, the client holds $\langle X \times Y \rangle_0 = \langle A \times Y \rangle_0$ directly and the server learns $\langle X \times Y \rangle_1 = (\langle X \rangle_0 + \langle X \rangle_1 - A) \times Y + \langle C \rangle_1$.

Secure element-wise multiplication protocol. Algorithm 1 is applicable for secure element-wise multiplication protocol SEleMul , with the modification of replacing matrix multiplication \times with element-wise multiplication

Algorithm 1 Secure Matrix Multiplication Protocol

Input: \mathcal{C} holds $\langle X \rangle_0 \in \mathbb{Z}_{2^i}^{m_1 \times m_2}$. \mathcal{S} holds $\langle X \rangle_1 \in \mathbb{Z}_{2^i}^{m_1 \times m_2}$, $Y \in \mathbb{Z}_{2^i}^{m_2 \times m_3}$.
Output: \mathcal{C} learns $\langle Z \rangle_0 \in \mathbb{Z}_{2^i}^{m_1 \times m_3}$. \mathcal{S} learns $\langle Z \rangle_1 \in \mathbb{Z}_{2^i}^{m_1 \times m_3}$, where $Z = X \times Y$.

Offline Phase: $\langle A \times Y \rangle$

- 1: \mathcal{C} and \mathcal{S} sample $A \xleftarrow{\$} \mathbb{Z}_{2^i}^{m_1 \times m_2}$, $B \xleftarrow{\$} \mathbb{Z}_{2^i}^{m_2 \times m_3}$, and then jointly invoke $\text{Gen}^{\text{BV}}(A, B) \rightarrow \{\langle C \rangle_0, \langle C \rangle_1\}$, where $C = A \times B$.
- 2: \mathcal{S} computes $Y - B$ and sends its result to \mathcal{C} and sets $\langle A \times Y \rangle_1 = \langle C \rangle_1$.
- 3: \mathcal{C} computes $\langle A \times Y \rangle_0 = A \times (Y - B) + \langle C \rangle_0$.

Online Phase: $\langle X \times Y \rangle$

- 4: \mathcal{C} computes $\langle X \rangle_0 - A$ and sends its result to \mathcal{S} . \mathcal{C} sets $\langle Z \rangle_0 = \langle A \times Y \rangle_0$.
- 5: \mathcal{S} computes $\langle Z \rangle_1 = (X - A) \times Y + \langle C \rangle_1 = (\langle X \rangle_0 + \langle X \rangle_1 - A) \times Y + \langle C \rangle_1$.

⊙, while ensuring that the sizes of X , Y , A , and B remain the same: $\langle Z \rangle = \text{SEleMul}(\langle X \rangle, Y)$, where $\langle z_{i,j} \rangle = \text{SMatMul}(\langle x_{i,j} \rangle, y_{i,j})$.

Secure fully-connected and convolutional layers. A fully-connected layer **SecFC** in NNs is essentially a matrix multiplication, thus **SecFC** is implemented directly using **SMatMul**: $\text{SecFC}(\langle D \rangle, W) = \text{SMatMul}(\langle D \rangle, W)$. A convolutional layer **SecCONV** can be expressed as a matrix multiplication with the help of reshape techniques [15] (**RshIn**, **RshFlt**, and **RshFlt** in Appendix A): $\text{SecCONV}(\langle D \rangle, W) = \text{RshOut}(\text{SMatMul}(\text{RshIn}(\langle D \rangle), \text{RshFlt}(W)))$.

4.2 Secure Quadratic Polynomial Protocol

We initially propose the secure quadratic polynomial protocol **SQuaPol**, which computes $z = p_2x^2 + p_1x + p_0$. In this protocol, both parties, \mathcal{C} and \mathcal{S} , hold secret-shared inputs $\langle x \rangle$ and outputs $\langle z \rangle$, while \mathcal{S} possesses the plaintext coefficients p_0, p_1, p_2 . We then extend **SQuaPol** to support polynomials of arbitrary degrees. Inspired by the customized **SMatMul** in Sec. 4.1, we have the insight of the following reformulation:

$$\begin{aligned} z &= p_2x^2 + p_1x + p_0 = p_2(x - a + a)^2 + p_1(x - a + a) + p_0 \quad \triangleright f \leftarrow x - a \\ &= p_2(f + a)^2 + p_1(f + a) + p_0 = p_2f^2 + \underline{p_2a^2} + \underline{2p_2af} + p_1f + \underline{p_1a} + p_0, \quad (2) \end{aligned}$$

where a is the randomness held by the client \mathcal{C} which is independent of the input x . Hence, **SQuaPol** can be efficiently divided into online and offline phases. The offline phase, which performs the computations labeled with underline in Eq. 2, is precomputed without knowing the input x , reducing the computational burden during the online phase. In the online phase, the client and server only need to compute the shared value f with minimal interaction, requiring just one round of communication and one message per party. This design significantly reduces both online communication overhead and latency, highlighting the efficiency and practicality of **PrivGNN**. Algorithm 2 presents the details of **SQuaPol** and the main steps are given below.

- The offline phase aims to learn the shared $\langle p_2a^2 \rangle_\gamma, \langle 2p_2a \rangle_\gamma, \langle p_1a \rangle_\gamma$ ($\gamma \in \{0, 1\}$) and precompute the parameter on the server side, which is used in the online

Algorithm 2 Secure Quadratic Polynomial Protocol

Input: \mathcal{C} holds $\langle x \rangle_0 \in \mathbb{Z}_{2^l}$. \mathcal{S} holds $\langle x \rangle_1 \in \mathbb{Z}_{2^l}$, $p_0, p_1, p_2 \in \mathbb{Z}_{2^l}$.

Output: \mathcal{C} learns $\langle z \rangle_0$. \mathcal{S} learns $\langle z \rangle_1$, where $z = p_2x^2 + p_1x + p_0$.

Offline Phase: $\langle p_2a^2 \rangle$, $\langle p_2a \rangle$, $\langle p_1a \rangle$,

1: \mathcal{C} and \mathcal{S} sample $\{a, a_1, a_2, a_3, a_4\} \xleftarrow{\$} \mathbb{Z}_{2^l}^5$, $\{b_1, b_2, b_3, b_4\} \xleftarrow{\$} \mathbb{Z}_{2^l}^4$, and then jointly invoke $\text{Gen}^{\text{BV}}(a_i, b_i)$ ($\forall i \in \{1, 2, 3, 4\}$) to generate triples $\{a_i, \langle c_i \rangle_0\}$ for \mathcal{C} and $\{b_i, \langle c_i \rangle_1\}$ for \mathcal{S} , where $c_i = a_i b_i$.

2: \mathcal{S} computes $e_1 \leftarrow p_1 - b_1$, $e_2 \leftarrow p_2 - b_2$, $e_3 \leftarrow p_2 - b_3$, and sends $e = \{e_1, e_2, e_3\}$ to \mathcal{C} .

3: \mathcal{C} computes $\langle p_1a \rangle_0 \leftarrow a_1 e_1 + \langle c_1 \rangle_0$, $\langle p_2a \rangle_0 \leftarrow a_2 e_2 + \langle c_2 \rangle_0$, $\langle p_2a^2 \rangle_0 \leftarrow a_3 e_3 + \langle c_3 \rangle_0$, $f_1 \leftarrow a - a_1$, $f_2 \leftarrow a - a_2$, $f_3 \leftarrow a^2 - a_3$, $f_4 = \langle p_2a \rangle_0 - a_4$, and sends $f = \{f_1, f_2, f_3, f_4\}$ to \mathcal{S} .

4: \mathcal{S} computes $\langle p_1a \rangle_1 \leftarrow p_1 f_1 + \langle c_1 \rangle_1$, $\langle p_2a \rangle_1 \leftarrow p_2 f_2 + \langle c_2 \rangle_1$, $\langle p_2a^2 \rangle_1 \leftarrow p_2 f_3 + \langle c_3 \rangle_1$.

Online Phase:

5: \mathcal{S} computes $e_4 \leftarrow \langle x \rangle_1 - b_4$ and sends e_4 to \mathcal{C} .

\mathcal{C} computes $f_5 \leftarrow \langle x \rangle_0 - a$ and sends f_5 to \mathcal{S} .

6: \mathcal{S} computes $\langle z \rangle_1 \leftarrow p_2(f_5 + \langle x \rangle_1)^2 + \langle p_2a^2 \rangle_1 + 2((f_5 + \langle x \rangle_1)(f_4 + \langle p_2a \rangle_1) + \langle c_4 \rangle_1) + p_1(f_5 + \langle x \rangle_1) + \langle p_1a \rangle_1 + p_0$.

\mathcal{C} computes $\langle z \rangle_0 \leftarrow \langle p_2a^2 \rangle_0 + 2(a_4(e_4 + f_5) + \langle c_4 \rangle_0) + \langle p_1a \rangle_0$.

phase of $\langle 2p_2af \rangle$ in Eq. 2. Therefore, four sets of multiplication triples need to be generated by invoking Gen^{BV} four times.

- During the online phase, after one communication round, the client computes $\langle z \rangle_0 \leftarrow \langle p_2a^2 \rangle_0 + 2(\underbrace{a_4(e_4 + f_5) + \langle c_4 \rangle_0}_{\langle p_2af \rangle_0}) + \langle p_1a \rangle_0$ and the server computes

$$\langle z \rangle_1 \leftarrow p_2 \underbrace{(f_5 + \langle x \rangle_1)^2}_f + \langle p_2a^2 \rangle_1 + 2(\underbrace{(f_5 + \langle x \rangle_1)(f_4 + \langle p_2a \rangle_1) + \langle c_4 \rangle_1}_{\langle p_2af \rangle_1}) + p_1 \underbrace{(f_5 + \langle x \rangle_1)}_f + \langle p_1a \rangle_1 + p_0.$$

Extension to higher-degree polynomials. Considering a d -degree polynomial $z = \sum_{i=0}^d p_i x^i \rightarrow \sum_{i=0}^d p_i (f + a)^i$, where $f = x - a$. We can use the binomial theorem [19] $((f + a)^i = \sum_{k=0}^i \binom{i}{k} a^k f^{i-k})$, then we obtain:

$$z = \sum_{i=0}^d p_i (f + a)^i = \sum_{i=0}^d \sum_{k=0}^i \binom{i}{k} p_i a^k f^{i-k}. \quad (3)$$

The secure computation rationale for Eq. 3 follows the same pattern as that of Eq. 2, with the protocol being divided into offline and online phases. Likewise, the underlined parts can be computed during the offline phase, where the client learns $\langle \binom{i}{k} p_i a^k \rangle_0$ and the server learns $\langle \binom{i}{k} p_i a^k \rangle_1$ ($\forall i \in \{1, 2, \dots, d\}, k \in \{1, 2, \dots, i\}$). In the online phase, the server reconstructs the plaintext f marked by a . At the end, only one-round online communication suffices to securely compute the arbitrary degree polynomial $\langle z \rangle = \sum_{i=0}^d \langle p_i x^i \rangle$.

4.3 Secure DReLU Protocol

Secure DReLU protocol **SDReLU** builds upon the FuncSS-based distributed comparison function $\mathcal{F}_{a,b}^{\text{cmp}}$ [3, 14], that is, $\mathcal{F}_{a,b}^{\text{cmp}}(x) = b$ if $x < a$; otherwise, $\mathcal{F}_{a,b}^{\text{cmp}}(x) = 0$. Recall from Sec. 2, the two-party $\mathcal{F}_{a,b}^{\text{cmp}}$ involves a pair of algorithms: $\text{Gen}^<(a, b)$ and $\text{Eval}^<(\gamma, k_\gamma, x)$. $\text{Gen}^<(a, b)$ creates two keys k_0, k_1 , then two parties learn the shared results $\mathcal{F}_\gamma^{\text{cmp}} \leftarrow \text{Eval}^<(\gamma, k_\gamma, x)$ of $\mathcal{F}_{a,b}^{\text{cmp}}(x)$. There are two challenges in employing $\mathcal{F}_{a,b}^{\text{cmp}}$ for **SDReLU**. *Challenge 1*: As x is in secret-shared form and needs to remain confidential, calling $\text{Eval}^>(\gamma, k_\gamma, x)$ by two parties poses a dilemma. *Challenge 2*: $\mathcal{F}_{a,b}^{\text{cmp}}$ is designed for less-than comparisons, whereas PrivGNN requires greater-than comparisons, as seen in layers like ReLU’s comparison of x against 0 or maxpool’s selection of the maximum value. Therefore, we made modifications to seamlessly integrate FuncSS-based $\mathcal{F}_{a,b}^{\text{cmp}}$ for efficient comparison of secret-shared data in PrivGNN. We present our approach below.

Addressing challenge 1. For a signed l -bit $x \in \mathbb{Z}_{2^l}$, $\text{DReLU}(x)$ is defined as:

$$\text{DReLU}(x) = 1 \{x < 2^{l-1}\} = 1 \oplus \text{MSB}(x). \quad (4)$$

We first define an offset function $f^a(x) = f(x - a)$ and establish the FuncSS-centric scheme accordingly, where $a \stackrel{\$}{\leftarrow} \mathbb{Z}_{2^l}$ is randomly generated by \mathcal{C} in \mathbb{Z}_{2^l} . Specifically, \mathcal{C} possesses the offset value of the input x and reveals the offset/-masked input $x + a \rightarrow \hat{x}$. The FuncSS keys are subsequently computed for $f^a(x + a)$, which is essentially equivalent to evaluating $f(x)$ on x . Accordingly, the offset function of Eq. 4 can be formulated as:

$$\text{DReLU}^{a,c}(\hat{x}) = \text{DReLU}(\hat{x} - a \pmod{2^l}) = \text{MSB}(\hat{x} - a \pmod{2^l}) \oplus 1 \oplus c.$$

where a and c are the input and output masks.

Addressing challenge 2. Inspired by CryptFlow2 [30], for shared $\langle x \rangle_0$ and $\langle x \rangle_1$ with $\langle y \rangle_0 = \langle x \rangle_0 \pmod{2^{l-1}}$ and $\langle y \rangle_1 = \langle x \rangle_1 \pmod{2^{l-1}}$, we reformulate $\text{MSB}(x)$ as: $\text{MSB}(x) = \text{MSB}(\langle x \rangle_0) \oplus \text{MSB}(\langle x \rangle_1) \oplus 1 \{2^{l-1} - \langle y \rangle_0 - 1 < \langle y \rangle_1\}$. Then, we set $\langle x \rangle_0 = \hat{x} = x + a$ and $\langle x \rangle_1 = 2^l - a$, we learn:

$$\text{DReLU}^{a,c}(\hat{x}) = \text{MSB}(\hat{x}) \oplus 1 \{2^{l-1} - \langle y \rangle_0 - 1 < \langle y \rangle_1\} \oplus \underline{\text{MSB}(2^l - a)} \oplus 1 \oplus c. \quad (5)$$

Hence, **SDReLU** is divided into online and offline phases, where the computations in the offline phase of Eq. 5 are labeled with underline as these are independent of the input x . Simultaneously, keys are prepared for online $\text{Eval}^<$ (i.e., $\mathcal{F}_{\langle y \rangle_1, 1}^{\text{cmp}}(2^{l-1} - \langle y \rangle_0 - 1)$) by invoking $\text{Gen}^<$. In the online phase, through only one communication round, \mathcal{C} and \mathcal{S} jointly recover the plaintext \hat{x} for \mathcal{S} learning $\text{MSB}(\hat{x})$. Meanwhile, with $\langle y \rangle_0 \leftarrow \hat{x} \pmod{2^{l-1}}$, \mathcal{C} and \mathcal{S} learn the Boolean-shared $1 \{2^{l-1} - \langle y \rangle_0 - 1 < \langle y \rangle_1\}$ locally based on the respective key. Algorithm 3 presents the **SDReLU** protocol’s main steps for executing $\text{DReLU}^{a,c}(\hat{x})$.

Secure ReLU layer. $\text{ReLU}(x)$ can be expressed as $\text{ReLU}(x) = x \cdot \text{DReLU}(x)$. To learn the result of secure ReLU layer **SecReLU** over the secret-shared $\langle x \rangle$, \mathcal{C} and \mathcal{S} learn $[z] \leftarrow \text{SDReLU}(\langle x \rangle)$ first and then multiply $[z]$ to $\langle x \rangle$: $\text{SecReLU}(\langle x \rangle) \leftarrow [z] \cdot \langle x \rangle$. However, direct multiplication of arithmetic shares $\langle x \rangle$ and Boolean

Algorithm 3 Secure DReLU Protocol**Input:** \mathcal{C} holds $\langle x \rangle_0 \in \mathbb{Z}_{2^l}$. \mathcal{S} holds $\langle x \rangle_1 \in \mathbb{Z}_{2^l}$.**Output:** \mathcal{C} learns $\langle z \rangle_0 \in \mathbb{Z}_2$. \mathcal{S} learns $\langle z \rangle_1 \in \mathbb{Z}_2$, where $z = \text{DReLU}(x)$.# **Offline Phase:** $\text{Gen}^{\text{DReLU}}$

- 1: Sample $\langle a \rangle_0, \langle a \rangle_1 \xleftarrow{\$} \mathbb{Z}_{2^l}$, $b \leftarrow -1$, and $a = \langle a \rangle_0 + \langle a \rangle_1 \pmod{2^l}$
- 2: Let $\langle x \rangle_1 = 2^l - a \in \mathbb{Z}_{2^l}$, $\langle y \rangle_1 = 2^l - a \pmod{2^{l-1}} \in \mathbb{Z}_{2^{l-1}}$.
- 3: $\{\tilde{k}_0, \tilde{k}_1\} \leftarrow \text{Gen}^<(1^\kappa, \langle y \rangle_1, b)$.

- 4: Sample a Boolean randomness $c \xleftarrow{\$} \mathbb{Z}_2$.

- 5: Let $r = c \oplus \left\lfloor \frac{\langle x \rangle_1}{2^{l-1}} \right\rfloor \oplus 1$.

- 6: Sample $[r]_0, [r]_1 \leftarrow \mathbb{Z}_2$, s.t., $[r]_0 \oplus [r]_1 = r$.

- 7: $\forall \gamma \in \{0, 1\}$, $k_\gamma = \tilde{k}_\gamma \parallel [r]_\gamma$.

- 8: **Return** $\{\langle a \rangle_\gamma, k_\gamma\}$

Online Phase: $\text{Eval}^{\text{DReLU}}$

- 9: $\forall \gamma \in \{0, 1\}$, parse $k_\gamma = \tilde{k}_\gamma \parallel [r]_\gamma$.

- 10: \mathcal{C} computes $\langle f \rangle_0 \leftarrow \langle x \rangle_0 + \langle a \rangle_0$ and sends $\langle f \rangle_0$ to \mathcal{S} .

- 11: \mathcal{S} computes $\langle f \rangle_1 \leftarrow \langle x \rangle_1 + \langle a \rangle_1$ and sends $\langle f \rangle_1$ to \mathcal{C} .

- 12: \mathcal{C} and \mathcal{S} recover $f \leftarrow \langle f \rangle_0 + \langle f \rangle_1 \pmod{2^{l-1}}$.

- 13: $\forall \gamma \in \{0, 1\}$, $[\tilde{z}]_\gamma \leftarrow \text{Eval}^<(\gamma, k_\gamma, 2^{l-1} - f - 1)$.

- 14: $\forall \gamma \in \{0, 1\}$, $[z]_\gamma \leftarrow \gamma \left\lfloor \frac{x+a}{2^{l-1}} \right\rfloor \oplus [r]_\gamma \oplus [\tilde{z}]_\gamma$.

shares $[z]$ is not feasible due to their calculation with different moduli. Various approaches tackle this issue by converting Boolean to arithmetic shares, typically requiring two rounds of online communication (i.e., one for conversion and another for multiplication). Upon investigation, **SBitXA** protocol (Algorithm 1 in **FssNN** [36]) reduces this to one round, with only $n + 1$ bits of online communication per party. In this work, we leverage **SBitXA** as a black box to learn $\text{SecReLU}(\langle x \rangle) \leftarrow \text{SBitXA}(\langle x \rangle, [z])$, with its functionality outlined in Appendix A.

Secure max pool layer. The rationale of secure max pool layer **SecMaxPool** in NNs is to select the maximum value from n^2 shared elements with an $(n \times n)$ -width pooling window. Given two elements $\langle x \rangle$ and $\langle y \rangle$, we reduce **SecMaxPool** to **SecReLU** according to below equation:

$$\begin{aligned} \max(\langle x \rangle, \langle y \rangle) &= x \cdot 1 \{ \langle x \rangle > \langle y \rangle \} + y \cdot (1 - 1 \{ \langle x \rangle > \langle y \rangle \}) \\ &= \underline{(\langle x \rangle - \langle y \rangle) \cdot 1 \{ \langle x \rangle - \langle y \rangle > 0 \}} + \langle y \rangle. \end{aligned} \quad (6)$$

The underlined part in Eq. 6 can be learned by invoking **SecReLU** protocol. Hence, **SecMaxPool** can be achieved by invoking **SecReLU** $(n^2 - 1)$ times for each $(n \times n)$ -width pooling window.

4.4 Secure Piecewise Polynomials Protocol

Sigmoidal activations in NNs pose challenges in 2PC due to the computational complexity of exponentiation and reciprocal, especially when working with blinded exponents. It is well-known that such activations can be approximated using piecewise continuous polynomials with negligible accuracy loss [20]. However, existing methods efficiently handle only low-degree polynomials, limiting

Algorithm 4 Secure Piecewise Polynomials Protocol**Input:** \mathcal{C} holds $\langle x \rangle_0 \in \mathbb{Z}_{2^l}$. \mathcal{S} holds $\langle x \rangle_1 \in \mathbb{Z}_{2^l}$.**Output:** \mathcal{C} learns $\langle z \rangle_0 \in \mathbb{Z}_2$. \mathcal{S} learns $\langle z \rangle_1 \in \mathbb{Z}_2$, where $z = P(x)$.

- 1: \mathcal{C} and \mathcal{S} execute **SDReLU** $k - 1$ times, resulting in $k - 1$ shared comparison bits: $\forall i \in \{1, 2, \dots, k - 1\}, [c_i] \leftarrow \text{SDReLU}(e_i - \langle x \rangle)$.
- 2: \mathcal{C} and \mathcal{S} execute **SQuaPol** k times, resulting in k shared polynomial results: $\forall i \in \{1, 2, \dots, k\}, \langle f_i \rangle \leftarrow \text{SQuaPol}(P_i(\langle x \rangle))$.
- 3: \mathcal{C} and \mathcal{S} compute the 1st-th polynomial via $\langle z_1 \rangle = \text{SBitXA}([c_1], \langle f_1 \rangle)$.
- 4: **for** $1 < i < k$ **do**
- 5: \mathcal{C} and \mathcal{S} compute the i -th polynomial via $\langle z_i \rangle = \text{SBitXA}([c_{i-1}] \oplus \gamma, \text{SBitXA}([c_i], \langle f_i \rangle))$.
- 6: **end for**
- 7: \mathcal{C} and \mathcal{S} compute the k -th polynomial via $\langle z_k \rangle = \text{SBitXA}([c_{k-1}] \oplus \gamma, \langle f_k \rangle)$.
- 8: \mathcal{C} and \mathcal{S} learn the AddSS-shared $P(\langle x \rangle)$: $\langle z \rangle \leftarrow \sum_{i=1}^k \langle z_i \rangle$.

approximation accuracy. To address this, we introduce a secure piecewise polynomial protocol **SPiePol**, that can be computed by our computational- and communication-saving designs. Given a piecewise polynomial function:

$$P(x) = P_1(x)\mathbf{1}_{x \in (-\infty, e_1)} + P_2(x)\mathbf{1}_{x \in [e_1, e_2)} + \dots + P_k(x)\mathbf{1}_{x \in [e_{k-1}, \infty)}, \quad (7)$$

where $P_i(x) = p_{i_0} + p_{i_1}x + \dots + p_{i_d}x^d$ ($\forall i \in \{1, \dots, k\}$) is a d -degree polynomial applied to different intervals of x . The indicator function $\mathbf{1}_{x \in [e_{i-1}, e_i)}$ ensures that each polynomial $P_i(x)$ is active only within its designated interval $[e_{i-1}, e_i)$. The k piecewise polynomials in $P(x)$ are transformed into a construction by **DReLU** (Eq. 8), where each polynomial $P_i(x)$ is activated by an indicator s_i . For example, $P_2(x)$ in Eq. 7 is activated when $e_1 \leq x < e_2$, i.e., $s_2 = 1$ and all other indicators are “0”. The indicator s_2 is derived from two **DReLU**s: $\neg c_1 \leftarrow \neg \text{DReLU}(e_1 - x) = 1$, indicating $e_1 \leq x$, and $c_2 \leftarrow \text{DReLU}(e_2 - x) = 1$, indicating $x < e_2$, with $s_2 = \neg c_1 \otimes c_2$, where \neg and \otimes represent logical NOT and AND, respectively. Thus, $P(x)$ ’s expression is given by:

$$P(x) = \text{DReLU}(e_1 - x) \cdot P_1(x) + \sum_{i=1}^{k-2} \neg \text{DReLU}(e_i - x) \cdot \text{DReLU}(e_{i+1} - x) \cdot P_{i+1}(x) + \neg \text{DReLU}(e_{k-1} - x) \cdot P_k(x). \quad (8)$$

To this end, we present the secure realization of **SPiePol** over the secret-sharing domain based on Eq. 8. Given the additive shares of each neuron output $\langle x \rangle$, and p_{ij} ($i \in [k], j \in [d]$) representing the plaintext weights held by the server \mathcal{S} , the client \mathcal{C} and server \mathcal{S} collaboratively compute $\langle z \rangle \leftarrow P(\langle x \rangle) = \text{SPiePol}(\langle x \rangle)$. The main steps are outlined in Algorithm 4.

Secure Sigmoid and Tanh layers. The server offers an efficient alternative to the expensive exponentiation and reciprocal operations inherent in standard sigmoidal activations, i.e., Sigmoid and Tanh. By approximating these activations with piecewise continuous polynomials (e.g., splines [7]), the server learns

the plaintext polynomial weights. As a result, secure Sigmoid and Tanh layers i.e., SecSig and SecTanh, can be implemented directly using SPiePol.

4.5 Putting Things Together

Having examined all essential supporting protocols (in Appendix C), we are ready to integrate these blocks into the scheme PrivGNN. PrivGNN considers the most widely used GNN model for graph classification tasks, namely message passing neural networks (MPNNs). The client \mathcal{C} first splits the graph-structured data $D = (G, E, H)$, where $G \in \mathbb{Z}_2^{n \times n}$ is the adjacency matrix, $E \in \mathbb{R}^{n \times n}$ represents the attributes of the edges, and $H \in \mathbb{R}^{n \times m}$ is the node feature matrix, with n as the number of nodes and m as the feature dimension. The data is split into random shared values $\langle D \rangle_0, \langle D \rangle_1$. At this point, the structure and data of the graph are fully protected. \mathcal{C} then uploads $\langle D \rangle_1 = (\langle G \rangle_1, \langle E \rangle_1, \langle H \rangle_1)$ to the server to access PrivGNN services. Due to space limitations, we outline the secure inference (SI) process in MPNN-centric PrivGNN. The SI process involves three functions: secure message passing function (PRIVMF), secure update function (PRIVUF), and secure readout function (PRIVRF) — which are securely executed as detailed below. Detailed information and the definition of MPNNs refer to Appendix B.

- PRIVMF consists of two types of NN layers: SecFC and SecReLU for message passing. With GS data $\langle D \rangle$ and server-hold plaintext parameter W , PRIVMF's workflow is $(\langle D \rangle, W) \rightarrow [\text{SecFC} \rightarrow \text{SecReLU}] \times \iota \rightarrow \text{SecFC} \rightarrow \text{AGGREGATE}(\cdot) \rightarrow \langle M \rangle = \{\langle m_v \rangle | v \in V\} = \{\langle \sum_{u \in N(v)} m_{u \rightarrow v} \rangle | v \in V\}$, where ι denotes the number of repetitions, AGGREGATE represents the aggregation function, V is the set of nodes in graph, and $N(v)$ is the set of neighboring nodes of node v .
- PRIVUF sets up secure update gates and secure reset gates, using SecSig, SecTanh, and SEleMul to compute the update states of the two gates. For each node $v \in V$ (i.e., $m_v \in M$ and $h_v \in H$), PRIVUF's workflow is $(\langle m_v \rangle, \langle h_v \rangle, W^0) \rightarrow [\text{SecFC} \rightarrow \text{SecReLU}] + [\text{SecFC} \rightarrow \text{SecReLU}] \rightarrow \text{SecSig} \rightarrow \langle \varphi_v \rangle, (\langle m_v \rangle, \langle h_v \rangle, W^1) \rightarrow [\text{SecFC} \rightarrow \text{SecReLU}] + [\text{SecFC} \rightarrow \text{SecReLU}] \rightarrow \text{SecSig} \rightarrow \langle \eta_v \rangle, (\langle m_v \rangle, \langle h_v \rangle, W^2) \rightarrow [\text{SecFC} \rightarrow \text{SecReLU}] \odot \langle \eta_v \rangle + [\text{SecFC} \rightarrow \text{SecReLU}] \rightarrow \text{SecTanh} \rightarrow \langle \vartheta_v \rangle$, and $(\langle \varphi_v \rangle, \langle \vartheta_v \rangle, \langle h_v \rangle) \rightarrow (1 - \langle \varphi_v \rangle) \odot \vartheta_v + \langle \varphi_v \rangle \odot \langle h_v \rangle \rightarrow \langle \hat{h}_v \rangle$. The overall updated node feature matrix is $\langle \hat{H} \rangle = \{\langle \hat{h}_v \rangle | v \in V\}$.
- With updated $\hat{H}^{(1)}$ and $\hat{H}^{(T)}$, where $\hat{H}^{(t)}$ denotes the outputs of t executions of PRIVMF $\xrightarrow{\text{PRIVUF}}$, PRIVRF executes two paths: $(\langle \hat{H}^{(1)} \rangle | \langle \hat{H}^{(T)} \rangle, W_R) \rightarrow [\text{SecFC} \rightarrow \text{SecReLU}] \times \iota \rightarrow \text{SecFC} \rightarrow \langle \tilde{\mathcal{H}}^{\iota+1} \rangle$ and $(\langle \hat{H}^{(T)} \rangle, W_Z) \rightarrow [\text{SecFC} \rightarrow \text{SecReLU}] \times \iota \rightarrow \text{SecFC} \rightarrow \langle \tilde{\mathcal{H}}^{\iota+1} \rangle$, where ι denotes the number of repetitions and $|$ denotes concatenation. PRIVRF then connect the results of the two paths: $\langle \mathcal{R} \rangle = \text{SMatMul}(\text{SEleMul}(\text{SecSig}(\langle \tilde{\mathcal{H}}^{\iota+1} \rangle), \langle \tilde{\mathcal{H}}^{\iota+1} \rangle), \text{SDReLU}(\langle \tilde{\mathcal{H}}^{\iota+1} \rangle))$.

5 Theoretical Analysis

Efficiency analysis. We analyze the computational and communication complexity of key protocols (SMatMul, SquaPol, SDRReLU, and SPiePol) and compare them with SotA schemes. We define SMul and SFunC as AddSS-based multiplication and FuncSS-based comparison operations. HEnc, HDec, HAdd, and

Table 1. Computation and communication costs of supporting protocols in PrivGNN, PAPI [5], and FastSecNet [11].

Protocols	Benchmarks	Computation	Communication
SMatMul	PAPI	$\mathcal{T}_{\text{SMul}}$	$2l$ (online: $2l$)
	PrivGNN	$\mathcal{T}_{\text{SMul}}$	$2l$ (online: l)
SQuaPol	PAPI	$7\mathcal{T}_{\text{HEnc}} + 3\mathcal{T}_{\text{HDec}} + 7\mathcal{T}_{\text{HAdd}} + 5\mathcal{T}_{\text{HMul}} + \mathcal{T}_{\text{SMul}}$	$5l_h + 2l$ (online: $2l$)
	PrivGNN	$\mathcal{T}_{\text{SMul}}$	$9l$ (online: $2l$)
SDReLU	FastSecNet	$2\mathcal{T}_{\text{SFunC}}$	$8(l+1)$ (online: $2l$)
	PrivGNN	$\mathcal{T}_{\text{SFunC}} + \mathcal{T}_{\text{SMul}}$	$2(2l+1)$ (online: $2l$)
SPiePol	FastSecNet	$2(k-1)\mathcal{T}_{\text{SFunC}} + \frac{d(d+1)k}{2}\mathcal{T}_{\text{SMul}}$	$8(k-1)(l+1) + d(d+1)kl$ (online: $2l(k-1) + \frac{d(d+1)kl}{2}$)
	PrivGNN	$(k-1)\mathcal{T}_{\text{SFunC}} + (2k-1)\mathcal{T}_{\text{SMul}}$	$6kl + 2k - 4l - 2$ (online: $4kl - 2l$)

Hmul represent the operations for encryption, decryption, addition, and multiplication with homomorphic ciphertexts. There are five parameters involved in this analysis: (1) the bit length l of an AddSS share, (2) the bit l_h length of the homomorphic ciphertext, (3) the pieces k of piecewise polynomials, (4) the degree d of each piece, (5) the time complexity \mathcal{T} of secure operations. Table 1 summarizes the analysis results. SQuaPol, SDReLU, and SPiePol show notable overall computational improvements, with most costs concentrated in the offline phase for generating cryptographic primitives like multiplication triples and keys. They also achieve lower overall communication costs compared to SotA.

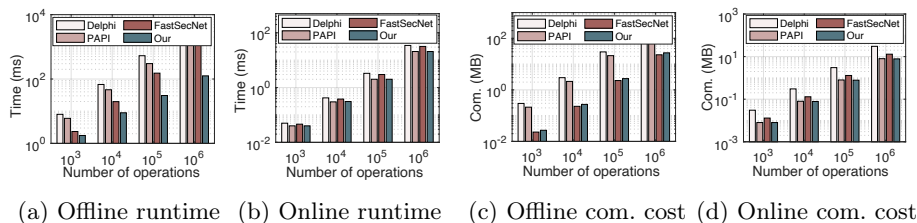
Security analysis. PrivGNN’s pipeline integrates a variety of cryptographic protocols for different layers, with each layer’s input and output in the additive secret-sharing domain. Using the sequential composition theorem [10], we deduce the overall security of PrivGNN’s inference as stated in Theorem 1.

Theorem 1. *PrivGNN’s secure inference scheme \prod^{PrivGNN} securely realizes the ideal functionality $\mathcal{F}^{\text{PrivGNN}}$ in the presence of one semi-honest adversary \mathcal{A} in the $(\prod_{\text{SecConv}}, \prod_{\text{SecReLU}}, \prod_{\text{SecMaxPool}}, \prod_{\text{SecSig}}, \prod_{\text{SecTanh}}, \prod_{\text{SecFC}}, \prod_{\text{SE1eMul}}$)-hybrid model.*

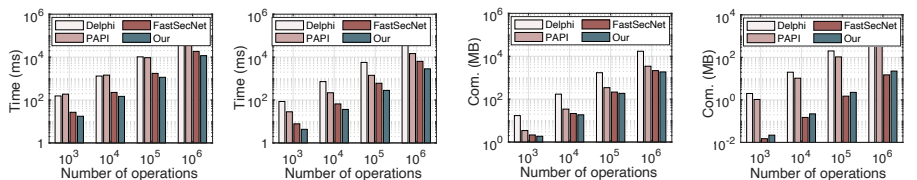
We analyze the security of PrivGNN against two types of semi-honest adversaries, as described in Definition 1. Specifically, we consider the following two cases based on the potential adversary: (1) a corrupted client ($\mathcal{P} \leftarrow \mathcal{C}$) and (2) a corrupted server ($\mathcal{P} \leftarrow \mathcal{S}$). The security of PrivGNN under these two semi-honest adversaries is proven according to the cryptographic standard outlined in Definition 1. Due to space constraints, we present a sketch of the security proof of the PrivGNN scheme in Appendix D.

6 Experimental Evaluation

Testbed. We implement a prototype of PrivGNN in Python 3.7 and PyTorch 1.9. Extensive experiments are conducted on two distinct servers with 64-core CPUs, 128GB RAM, and 2 NVIDIA GeForce RTX 2080Ti GPU. The environment simulates a local-area network with 1 Gbps bandwidth and 0.1 ms latency. Following prior work [11, 21, 31], we set the integer ring size of additive secret shares to $\mathbb{Z}_{2^{32}}$. Cleartext NN models are trained using PyTorch on an NVIDIA RTX 2080Ti GPU, using the standard SGD optimizer with a learning rate of 0.001, batch size of 128, momentum of 0.9, and weight decay of 1×10^{-6} .



(a) Offline runtime (b) Online runtime (c) Offline com. cost (d) Online com. cost

Fig. 2. Performance comparison of SQuaPo1 protocol.

(a) Offline runtime (b) Online runtime (c) Offline com. cost (d) Online com. cost

Fig. 3. Performance comparison of SecReLU protocol.

Datasets and models. We evaluate PrivGNN on MNIST, CIFAR-10, CIFAR-100, and QM9 [27] datasets. MNIST, CIFAR-10, and CIFAR-100, are utilized for image classification inference with CNNs. The QM9 dataset, comprising 134,000 stable organic molecules, is processed into molecular fingerprints using RDKit [1], and employed for molecular property inference with GNNs. The models include a 3-layer fully-connected network (FC-3) and a 4-layer CNN (CNN-4) [31] for MNIST, LeNet [16] and VGG-16 [32] for CIFAR10, ResNet-32 [12] and VGG-16 [32] for CIFAR-100, and the GNN in Sec. 4.5 for QM9.

6.1 Microbenchmarks

Secure SQuaPo1 protocol. Fig. 2 compares the performance of SQuaPo1 with Delphi [23], PAPI [5], and FastSecNet [11] in terms of runtime and communication (com.) cost across operation scales from 10^3 to 10^6 . SQuaPo1 outperforms in online computation, showing a $\sim 35\%$ reduction in time compared to Delphi and FastSecNet, benefiting from a single-round interaction. Online com. of SQuaPo1 also excels, with costs $4\times$ and $2\times$ lower than Delphi and FastSecNet. For offline computation, SQuaPo1 achieves improvements by one order of magnitude over other schemes. While its offline com. cost is $\sim 20\%$ higher than FastSecNet, SQuaPo1 prioritizes online efficiency, delivering strong overall performance.

Secure comparison-based activations. Fig. 3 reports the performance comparison of SecReLU. SecReLU achieves $2\times \sim 15\times$ speedup in online runtime compared to Delphi, PAPI, and FastSecNet, with higher online com. costs than FastSecNet. Specifically, SecReLU requires one FuncSS-based comparison and one multiplication for $\text{ReLU}(x) = \text{DReLU}(x) \cdot x$, while FastSecNet uses two comparisons, trading online time for lower com. In contrast, Delphi and PAPI’s use of garbled circuits and HE introduces higher overall costs. Further, SecReLU presents superior offline performance in both runtime and com. costs, stressing its overall efficiency.

Fig. 4. Performance comparison of SecSig/SecTanh protocols: SiRNN uses a 3-piece polynomial, MiniONN a 12-piece linear approximation, while our protocol employs a 12-piece quadratic approximation, offering better accuracy due to the higher polynomial degree.

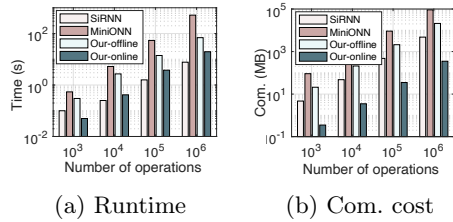


Table 2. Summary of inference accuracy (%)

	MNIST		CIFAR-10		CIFAR-100		QM9
	FC-3	CNN-4	LeNet-5	VGG-16	ResNet-32	VGG-16	GNN
Training Accuracy	96.96	99.16	96.81	88.03	74.90	72.22	98.17
Plain Inference	96.23	99.02	81.50	87.45	68.48	70.12	97.93
Secure Inference	96.12	99.00	80.57	87.40	68.16	70.12	96.42

Secure sigmoidal activations. Fig. 4 reports the performance of SecSig (equivalent to SecTanh and SPiePol) compared to SiRNN [29] and MiniONN [20]. SecSig adopts a 12-piece spline, sufficient for maintaining cross-entropy loss [20]. We compare its online portion to the total costs of the fully online baselines. The online running time and com. cost of SecSig are both less than those of SiRNN and MiniONN for small scales. For 10^3 operations, the online running time of SecSig is $2\times$ and $10.8\times$ faster than SiRNN and MiniONN. The 3-piece spline of SiRNN shows sub-linear growth in latency but offers limited approximation accuracy. Regarding online com. cost, SecSig is $13.6\times$ and $260\times$ lower than SiRNN and MiniONN across all operation scales.

6.2 Summary of Accuracy

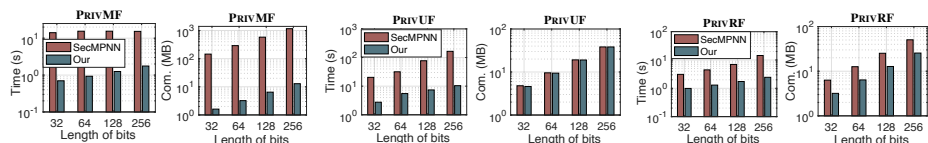
Table 2 compares the inference accuracy of PrivGNN with plaintext inference across various datasets and models, delivering its robust SI capabilities. Specifically, for MNIST, PrivGNN achieves an inference accuracies of 96.12% and 99.00% for the FC-3 and CNN-4 models, respectively, only $\sim 0.1\%$ lower than their plaintext. On CIFAR-10, PrivGNN records accuracies of 80.57% for LeNet-5 and 87.40% for VGG-16, slightly below the plaintext accuracies of 81.50% and 87.45%, respectively, but still reasonable. For CIFAR-100, ResNet-32 achieves a SI accuracy of 68.16%, closely trailing the plaintext accuracy of 68.48%. VGG-16 matches the plaintext accuracy of 70.12%. On QM9 with the GNN model, PrivGNN reports an accuracy of 96.42%, slightly below the plaintext accuracy of 97.93%. The possible minor accuracy drop is from truncation errors caused by the fixed-point arithmetic and the reformulated activations used in PrivGNN.

6.3 Performance Comparison with SotA Schemes

Secure convolutional inference. We evaluate PrivGNN on CNN-4 (MNIST), VGG-16 (CIFAR-10), and ResNet-32 (CIFAR-100), and compare it with SotA works. The reported results of SotA are sourced directly from the respective papers. Table 3 shows that PrivGNN improves CNN-4’s online runtime up to

Table 3. Performance comparison with SotA schemes (Scheme abbreviations: Medi = MediSC [21], Fast = FastSecNet [11], Cryp = CrypTFlow2 [30], Delp = Delphi [23], Chet = Cheetah [13], SecM = SecMPNN [18]).

Model	MNIST (CNN-4)				CIFAR-10 (VGG-16)				CIFAR-100 (ResNet-32)				QM9 (MPNN)		
	Scheme	XONN	Medi	Fast	Ours	Cryp	Delp	PAPI	Ours	Cryp	Chet	PAPI	Ours	SecM	Ours
Time (s)	Offline	-	1.21	0.25	0.22	42.50	88.10	47.80	23.15	62.50	-	65.70	38.36	-	5.45
	Online	0.30	4.42	0.07	0.06	4.20	6.30	1.30	1.53	6.40	15.95	4.50	3.35	102.57	2.48
	Total	0.30	5.63	0.32	0.28	46.70	94.40	49.10	24.68	68.90	15.95	70.20	41.71	102.57	7.93
Com. (MB)	Offline	-	2.54	40.19	22.59	30.72	51.20	40.96	217.74	122.88	-	143.36	468.26	-	87.04
	Online	62.77	2.62	0.65	1.28	686.08	40.96	30.72	16.55	583.68	112.64	122.88	24.38	156.13	9.10
	Total	62.77	5.16	40.84	23.87	716.8	92.16	71.68	234.29	706.56	112.64	266.24	492.64	156.13	96.14



(a) Runtime (b) Com. cost (c) Runtime (d) Com. cost (e) Runtime (f) Com. cost

Fig. 5. Performance comparison of different functions in GNNs.

$1.2\times \sim 73.6\times$. For VGG-16, PrivGNN exhibits $2.6\times$ and $4.0\times$ savings in online time when compared to the CrypTFlow2 [30] and Delphi [23], respectively. While PrivGNN’s online runtime is 21.5% higher than PAPI [5] due to PAPI’s optimization strategy of retaining a subset of ReLUs. For ResNet-32, PrivGNN outperforms prior works with online runtime reductions of $1.3\times \sim 4.7\times$, also enhancing online communication efficiency by $4.6\times \sim 23.9\times$.

Secure graph inference. The overall performance is shown in the last column of Table 3. We evaluate three functions across four secret-sharing ring sizes ($l = 32, 64, 128, 256$ bits) to assess resource usage. Fig. 5 shows that time and com. costs rise with larger secret-sharing ring sizes. As seen, the three functions PRIVMF, PRIVUF, and PRIVRF take 0.7, 2.7, and 1.0 seconds, respectively, to predict one molecule within the 32-bit ring. Overall, PrivGNN achieves an order of magnitude improvement over SecMPNN [18], attributed to the dual optimization of linear and non-linear layers. Notably, PRIVUF accounts for 50% \uparrow of the overall SI runtime due to the costly piecewise approximations of sigmoidal activations like SecSig and SecTanh.

7 Conclusion

We introduced PrivGNN, a high-performance cryptographic inference framework designed for privacy-preserving GNNs in the semi-honest client-server setup. By carefully designing a line of secure offline-online protocols for both linear and non-linear layers with lightweight AddSS and FuncSS, PrivGNN performs fast and low-interactive inference during the online phase. Extensive experimental results on benchmark and real-world datasets shed light on the scalability and efficiency of PrivGNN, outperforming prior PDDL and secure GNN schemes. Our approach paves the way for secure, scalable AI-driven services in various domains, including drug discovery and beyond.

References

1. RDKit: Open-source cheminformatics. <http://www.rdkit.org>, accessed 11 Apr 2024
2. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: *Advances in Cryptology—CRYPTO’91: Proceedings 11*. pp. 420–432. Springer (1992)
3. Boyle, E., Chandran, N., Gilboa, N., Gupta, D., Ishai, Y., Kumar, N., Rathee, M.: Function secret sharing for mixed-mode and fixed-point secure computation. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. pp. 871–900. Springer (2021)
4. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: *Annual international conference on the theory and applications of cryptographic techniques*. pp. 337–367. Springer (2015)
5. Cheng, K., Xi, N., Liu, X., Zhu, X., Gao, H., et al.: Private inference for deep neural networks: a secure, adaptive, and efficient realization. *IEEE Transactions on Computers* (2023)
6. Demmler, D., Schneider, T., Zohner, M.: ABY-a framework for efficient mixed-protocol secure two-party computation. In: *NDSS* (2015)
7. Dierckx, P.: *Curve and surface fitting with splines*. Oxford University Press (1995)
8. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. In: *International conference on machine learning*. pp. 1263–1272. PMLR (2017)
9. Goldreich, O.: *Foundations of cryptography: volume 2, Basic Applications*. Cambridge University Press (2009)
10. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game, or a completeness theorem for protocols with honest majority. In: *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pp. 218–229 (1987)
11. Hao, M., Li, H., Chen, H., Xing, P., Zhang, T.: FastSecNet: An efficient cryptographic framework for private neural network inference. *IEEE TIFS* **18**, 2569–2582 (2023)
12. 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*. pp. 770–778 (2016)
13. Huang, Z., Lu, W., Hong, C., Ding, J.: Cheetah: Lean and fast secure two-party deep neural network inference. In: *USENIX Security*. pp. 809–826 (2022)
14. Jawalkar, N., Gupta, K., Basu, A., Chandran, N., Gupta, D., Sharma, R.: Orca: Fss-based secure training and inference with GPUs. In: *S&P*. pp. 63–63. IEEE Computer Society (2023)
15. Kumar, N., Rathee, M., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: Cryptflow: Secure tensorflow inference. In: *S&P*. pp. 336–353. IEEE (2020)
16. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86**(11), 2278–2324 (1998)
17. Liao, X., Liu, W., Zheng, X., Yao, B., Chen, C.: PPGenCDR: A stable and robust framework for privacy-preserving cross-domain recommendation. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. vol. 37, pp. 4453–4461 (2023)
18. Liao, X., Xue, J., Yu, S., Liu, X., Shu, J.: SecMPNN: 3-party privacy-preserving molecular structure properties inference. In: *2022 IEEE International Conference on Acoustics, Speech and Signal Processing*. pp. 3004–3008. IEEE (2022)
19. Liu, C.s.: The essence of the generalized newton binomial theorem. *Communications in Nonlinear Science and Numerical Simulation* **15**(10), 2766–2768 (2010)

20. Liu, J., Juuti, M., Lu, Y., Asokan, N.: Oblivious neural network predictions via MiniONN transformations. In: CCS. pp. 619–631 (2017)
21. Liu, X., Zheng, Y., Yuan, X., Yi, X.: Towards secure and lightweight deep learning as a medical diagnostic service. In: 26th European Symposium on Research in Computer Security. pp. 519–541. Springer (2021)
22. Makri, E., Rotaru, D., Vercauteren, F., Wagh, S.: Rabbit: Efficient comparison for secure multi-party computation. In: International Conference on Financial Cryptography and Data Security. pp. 249–270. Springer (2021)
23. Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., Popa, R.A.: DELPHI: A cryptographic inference service for neural networks. In: USENIX Security. pp. 2505–2522 (2020)
24. Ohata, S., Nuida, K.: Communication-efficient (client-aided) secure two-party protocols and its application. In: International Conference on Financial Cryptography and Data Security. pp. 369–385. Springer (2020)
25. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International conference on the theory and applications of cryptographic techniques. pp. 223–238. Springer (1999)
26. Peng, H., Ran, R., Luo, Y., Zhao, J., Huang, S., et al.: LinGCN: Structural linearized graph convolutional network for homomorphically encrypted inference. *Advances in Neural Information Processing Systems* **36** (2024)
27. Ramakrishnan, R., Dral, P.O., Rupp, M., Von Lilienfeld, O.A.: Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data* **1**(1), 1–7 (2014)
28. Ran, R., Wang, W., Gang, Q., Yin, J., Xu, N., Wen, W.: CryptoGCN: Fast and scalable homomorphically encrypted graph convolutional network inference. *Advances in Neural information processing systems* **35**, 37676–37689 (2022)
29. Rathee, D., Rathee, M., Goli, R.K.K., Gupta, D., et al.: SiRnn: A math library for secure RNN inference. In: S&P. pp. 1003–1020. IEEE (2021)
30. Rathee, D., Rathee, M., Kumar, N., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: CrypTFlow2: Practical 2-party secure inference. In: CCS. pp. 325–342 (2020)
31. Riazi, M.S., Samragh, M., Chen, H., Laine, K., et al.: XONN: XNOR-based oblivious deep neural network inference. In: USENIX Security. pp. 1501–1518 (2019)
32. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014)
33. Wang, S., Zheng, Y., Jia, X.: Secgnn: Privacy-preserving graph neural network training and inference as a cloud service. *IEEE Transactions on Services Computing* **16**(4), 2923–2938 (2023)
34. Wu, F., Long, Y., Zhang, C., Li, B.: Linkteller: Recovering private edges from graph neural networks via influence analysis. In: S&P. pp. 2005–2024. IEEE (2022)
35. Xu, Z., Lai, S., Liu, X., Abuadba, A., Yuan, X., Yi, X.: OblivGNN: Oblivious inference on transductive and inductive graph neural network. In: USENIX Security. pp. 2209–2226 (2024)
36. Yang, P., Jiang, Z.L., Gao, S., Zhuang, J., Wang, H., Fang, J., Yiu, S., Wu, Y., Wang, X.: FssNN: communication-efficient secure neural network training via function secret sharing. *Cryptology ePrint Archive* (2023)
37. Zhou, L., Wang, Z., Cui, H., Song, Q., Yu, Y.: Bicaptor: Two-round secure three-party non-linear computation without preprocessing for privacy-preserving machine learning. In: S&P. pp. 534–551. IEEE (2023)

A Functionalities

We introduce the ideal functionalities \mathcal{F}_{ABB} of arithmetic black-box in Fig. 6 that are used in PrivGNN.

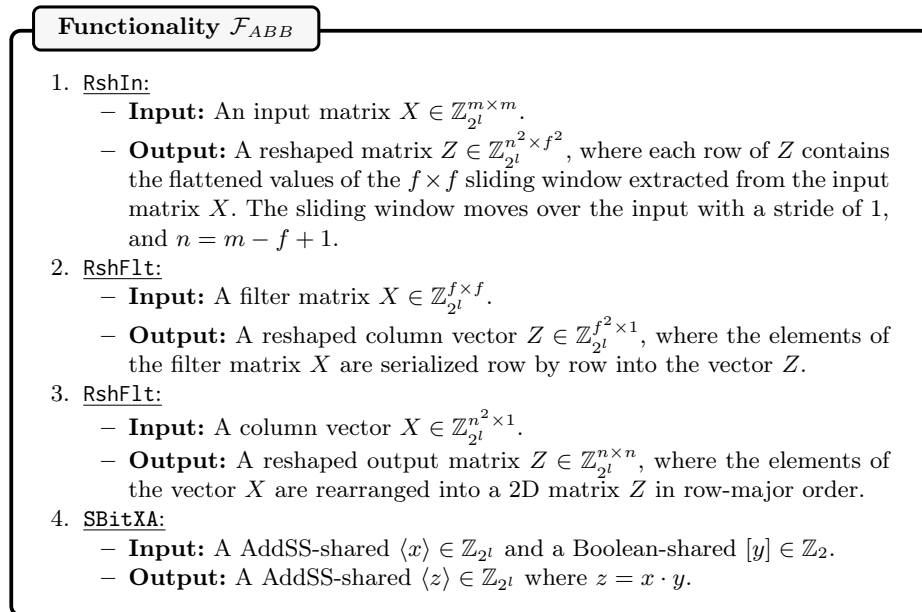


Fig. 6. The arithmetic black-box functionality.

B Detailed Functions in PrivGNN

We now outline our comprehensive scheme for secure GNN inference, PrivGNN, which is demonstrated using a typical GNN model: message passing neural networks (MPNNs), for molecular property inference in drug discovery. While this example focuses on MPNNs, PrivGNN provides a general solution for secure inference (SI) across various GNN-centric services. Specifically, MPNNs [8] used for learning GS data, aim to predict the overall properties of a graph by synthesizing node features. They update node features iteratively by aggregating information from neighboring nodes, simulating message passing between nodes. The detailed definition of MPNNs is provided in **Definition 4**.

Definition 4. Let $T \in \mathbb{N}$ denote the number of iterations and $G = (V, W)$ be a weighted graph with the node set V and the weight set W . For $t = \{1, 2, \dots, T\}$, let $\Phi^t : \mathbb{R}^{h^{(t-1)}} \rightarrow \mathbb{R}^{m^t}$, $\Psi^t : \mathbb{R}^{(h^{(t-1)}, m^t)} \rightarrow \mathbb{R}^{h^{(t)}}$, and $\Upsilon : \mathbb{R}^{h^{(T)}} \rightarrow \mathbb{R}^y$ be message passing, update, and readout functions, which are briefly introduced as follows:

- **Message Passing Function:** Initially, each node $v \in V$ is associated with a feature vector $h_v^{(0)}$, representing the initial state of the node. At each iteration t , a message passing function Φ^t is applied to each node and its

Algorithm 5 Secure Message Function (PRIVMF)

Input: \mathcal{C} holds $\langle G \rangle_0, \langle E \rangle_0, \langle H \rangle_0 \in \mathbb{Z}_{2^l}$. \mathcal{S} holds $\langle G \rangle_1, \langle E \rangle_1, \langle H \rangle_1 \in \mathbb{Z}_{2^l}$ and $W \in \mathbb{R}$.

Output: \mathcal{C} learns node embeddings $\langle M \rangle_0 \in \mathbb{Z}_{2^l}$. \mathcal{S} learns $\langle M \rangle_1 \in \mathbb{Z}_{2^l}$, where $M = \Phi(G, E, H)$.

\mathcal{C} and \mathcal{S} execute collaboratively:

- 1: $\langle \mathcal{E}^0 \rangle = \langle E \rangle$.
- 2: **for** $i \in \{0, \dots, \iota - 1\}$ **do**
- 3: $\langle \tilde{\mathcal{E}}^{i+1} \rangle = \text{SecFC}(\langle \mathcal{E}^i \rangle, W^i)$. \triangleright W^i denotes the learned weight matrix for the i -th SecFC.
- 4: $\langle \mathcal{E}^{i+1} \rangle = \text{SecReLU}(\langle \tilde{\mathcal{E}}^{i+1} \rangle)$.
- 5: **end for**
- 6: $\langle \tilde{\mathcal{E}}^\iota \rangle = \text{SecFC}(\langle \mathcal{E}^\iota \rangle, W^\iota)$.
- 7: $\langle \hat{\mathcal{E}}^\iota \rangle = \text{SMatMul}(\langle H \rangle, \langle \tilde{\mathcal{E}}^\iota \rangle)$.
- 8: $\langle M \rangle = \text{SMatMul}(\langle G \rangle, \langle \hat{\mathcal{E}}^\iota \rangle)$.
- 9: **return** $\langle M \rangle$

neighboring nodes to generate a message vector $m_{u \rightarrow v}^t$. $m_{u \rightarrow v}^t$ contains information from neighboring nodes u to central node v , expressed as $m_{u \rightarrow v}^t = \Phi^t(h_u^{(t-1)}, h_v^{(t-1)}, w_{u \rightarrow v})$, where $w_{u \rightarrow v} \in W$ denotes the edge features between nodes u and v . Next, the message vectors from neighboring nodes are aggregated to update the node v 's features, expressed as $m_v^t = \sum_{u \in N(v)} m_{u \rightarrow v}^t$, where $N(v)$ represents the set of neighboring nodes of node v .

- *Update Function:* An update function Ψ^t is applied to each node to update its feature $h_v^{(t-1)}$ from iteration $t-1$ to iteration t via $h_v^{(t)} = \Psi^t(h_v^{(t-1)}, m_v^t)$. The process is repeated for a fixed number of iterations or until convergence.
- *Readout Function:* After T iterations, the learnable readout function Υ maps the final state $h_v^{(T)}$ of the node v to the target y via $y = \sum_{v \in V} \Upsilon(h_v^{(T)})$.

Thus, PrivGNN consists of three stages: the secure message function (PRIVMF), the secure update function (PRIVUF), and the secure readout function (PRIVRF). Three secure functions are securely executed as detailed below.

Secure message passing function. PRIVMF extracts node features in GS data over the secret shared domain while ensuring no plaintext information leakage to the cloud server. The input to the PRIVMF function is GS data $\langle D \rangle = (\langle G \rangle, \langle E \rangle, \langle H \rangle)$ and server-hold plaintext GNN model parameter W . Various inputs $\langle D \rangle$ to be inferred have a relatively fixed data size. If there are size differences, padding with zeros is applied to ensure compatibility with the GNN model. The secure message function consists of two types of NN layers: SecFC and SecReLU. In detail, the process involves $(\langle D \rangle, W) \rightarrow [\text{SecFC} \rightarrow \text{SecReLU}] \times \iota \rightarrow \text{SecFC} \rightarrow \text{AGGREGATE}(\cdot) \rightarrow \langle M \rangle = \{\langle m_v \rangle | v \in V\}$, where ι denotes the number of repetitions and AGGREGATE represents the aggregation function. After PRIVMF, the client and server output the shared value $\{\langle m_v \rangle | v \in V\} = \{\langle \sum_{u \in N(v)} m_{u \rightarrow v} \rangle | v \in V\}$, where V is the set of nodes in the

Algorithm 6 Secure Update Function (PRIVUF)**Input:** \mathcal{C} holds $\langle M \rangle_0, \langle H \rangle_0 \in \mathbb{Z}_{2^l}$. \mathcal{S} holds $\langle M \rangle_1, \langle H \rangle_1 \in \mathbb{Z}_{2^l}$ and $W \in \mathbb{R}$.**Output:** \mathcal{C} learns updated feature $\langle \hat{H} \rangle_0 \in \mathbb{Z}_{2^l}$. \mathcal{S} learns $\langle \hat{H} \rangle_1 \in \mathbb{Z}_{2^l}$, where $\hat{H} = \Psi(H, M)$. **\mathcal{C} and \mathcal{S} execute collaboratively:**

```

1: for  $v \in \{0, \dots, |V| - 1\}$  do
2:    $\langle R_v^1 \rangle = \text{SecFC}(\langle m_v \rangle, W_M^0), \langle R_v^2 \rangle = \text{SecFC}(\langle h_v \rangle, W_H^0)$ .
3:   for  $j \in \{0, \dots, |h_v|\}$  do
4:      $\langle \varphi_{v,j} \rangle = \text{SecSig}(\langle R_{v,j}^1 \rangle + \langle R_{v,j}^2 \rangle)$ .
5:   end for
6:    $\langle Z_v^1 \rangle = \text{SecFC}(\langle m_v \rangle, W_M^1), \langle Z_v^2 \rangle = \text{SecFC}(\langle h_v \rangle, W_H^1)$ .
7:   for  $j \in \{0, \dots, |h_v|\}$  do
8:      $\langle \eta_{v,j} \rangle = \text{SecSig}(\langle Z_{v,j}^1 \rangle + \langle Z_{v,j}^2 \rangle)$ .
9:   end for
10:   $\langle N_v^1 \rangle = \text{SecFC}(\langle m_v \rangle, W_H^2), \langle N_v^2 \rangle = \text{SecFC}(\langle h_v \rangle, W_H^2)$ .
11:   $\langle N_v^3 \rangle = \text{SEleMul}(\langle \eta_v \rangle, \langle N_v^2 \rangle)$ 
12:  for  $j \in \{0, \dots, |h_v|\}$  do
13:     $\langle \vartheta_{v,j} \rangle = \text{SecTanh}(\langle N_{v,j}^1 \rangle + \langle N_{v,j}^3 \rangle)$ .
14:  end for
15:   $\langle U_v^1 \rangle = \text{SEleMul}(\langle \varphi_v \rangle, \langle h_v \rangle)$ .
16:   $\langle U_v^2 \rangle = \langle 1 \rangle - \langle \varphi_v \rangle$ .
17:   $\langle U_v^3 \rangle = \text{SEleMul}(\langle \vartheta_v \rangle, \langle U_v^2 \rangle)$ .
18:   $\langle U_v \rangle = \langle U_v^1 \rangle + \langle U_v^3 \rangle$ .
19:   $\langle \hat{h}_v \rangle = \text{SMatMul}(\langle U_v \rangle, \langle h_v \rangle)$ .
20: end for
21: return  $\langle \hat{H} \rangle = \{\langle \hat{h}_v \rangle\}_{v=0}^{|V|-1}$ 

```

graph, and $N(v)$ represents the set of neighboring nodes of node v . The shared $\langle M \rangle$ is then sent to PRIVUF. The main steps are outlined in Algorithm 5.

Secure update function. PRIVUF aims to update each node's features in the graph without leaking any information about node features. The secure gated recurrent unit serves as its core. Specifically, with the node features output from PRIVMF denoted as $\langle M \rangle$ as input, PRIVUF sets up secure update gates and secure reset gates using secure Sigmoid SecSig. It also utilizes secure SecTanh and secure element-wise multiplication SEleMul to compute the update states of the two gates. The detailed process of PRIVUF for each node $v \in V$ (i.e., $m_v \in M$ and $h_v \in H$) is formulated as below:

$$\begin{aligned}
\langle \varphi_v \rangle &= \text{SecSig}(\text{SecFC}(\langle m_v \rangle, W_M^0) + \text{SecFC}(\langle h_v \rangle, W_H^0)) \\
\langle \eta_v \rangle &= \text{SecSig}(\text{SecFC}(\langle m_v \rangle, W_M^1) + \text{SecFC}(\langle h_v \rangle, W_H^1)) \\
\langle \vartheta_v \rangle &= \text{SecTanh}(\text{SecFC}(\langle m_v \rangle, W_M^2) + \text{SEleMul}(\langle \eta_v \rangle, \text{SecFC}(\langle h_v \rangle, W_H^2))) \\
\langle \hat{h}_v \rangle &= \text{SEleMul}(1 - \langle \varphi_v \rangle, \langle \vartheta_v \rangle) + \text{SEleMul}(\langle \varphi_v \rangle, \langle h_v \rangle).
\end{aligned}$$

Algorithm 7 Secure Readout Function (PRIVRF)

Input: \mathcal{C} holds $\langle \hat{H}^{(1)} \rangle_0, \langle \hat{H}^{(T)} \rangle_0 \in \mathbb{Z}_{2^l}$. \mathcal{S} holds $\langle \hat{H}^{(1)} \rangle_1, \langle \hat{H}^{(T)} \rangle_1 \in \mathbb{Z}_{2^l}$ and $W \in \mathbb{R}$.

Output: \mathcal{C} learns updated feature $\langle R \rangle_0 \in \mathbb{Z}_{2^l}$. \mathcal{S} learns $\langle R \rangle_1 \in \mathbb{Z}_{2^l}$, where $R = \Upsilon(\hat{H}^{(1)}, \hat{H}^{(T)})$.

\mathcal{C} and \mathcal{S} execute collaboratively:

- 1: $\langle \mathcal{H}^0 \rangle = \langle \hat{H}^{(T)} \rangle_1$.
- 2: $\langle \mathcal{H}\mathcal{H}^0 \rangle = \langle \hat{H}^{(1)} \rangle | \langle \hat{H}^{(T)} \rangle$. \triangleright | denotes concatenation.
- 3: **for** $v \in \{0, \dots, |V|\}$ **do**
- 4: **for** $j \in \{0, \dots, \iota - 1\}$ **do**
- 5: $\langle \tilde{\mathcal{H}}_v^j \rangle = \text{SecFC}(\langle \mathcal{H}_v^j \rangle, \langle W_Z^j \rangle)$, $\langle \mathcal{H}_v^{j+1} \rangle = \text{SecReLU}(\langle \tilde{\mathcal{H}}_v^j \rangle)$.
- 6: $\langle \tilde{\mathcal{H}}\tilde{\mathcal{H}}_v^j \rangle = \text{SecFC}(\langle \mathcal{H}\mathcal{H}_v^j \rangle, \langle W_R^j \rangle)$, $\langle \mathcal{H}\mathcal{H}_v^{j+1} \rangle = \text{SecReLU}(\langle \tilde{\mathcal{H}}\tilde{\mathcal{H}}_v^j \rangle)$.
- 7: **end for**
- 8: $\langle \tilde{\mathcal{H}}_v^{\iota+1} \rangle = \text{SecFC}(\langle \mathcal{H}_v^\iota \rangle, \langle W_Z^\iota \rangle)$, $\langle \mathcal{H}\tilde{\mathcal{H}}_v^{\iota+1} \rangle = \text{SecFC}(\langle \mathcal{H}\mathcal{H}_v^\iota \rangle, \langle W_R^\iota \rangle)$.
- 9: **for** $j \in \{0, \dots, |\langle \mathcal{H}\tilde{\mathcal{H}}_v^{\iota+1} \rangle|\}$ **do**
- 10: $\langle S_{v,j}^{\iota+1} \rangle = \text{SecSig}(\langle \mathcal{H}\tilde{\mathcal{H}}_v^{\iota+1} \rangle)$.
- 11: **end for**
- 12: **end for**
- 13: $\langle K \rangle = \text{SEleMul}(\langle S^{\iota+1} \rangle, \langle \tilde{\mathcal{H}}^{\iota+1} \rangle)$.
- 14: $\langle \varphi \rangle = \text{SDReLU}(\langle \tilde{\mathcal{H}}^{\iota+1} \rangle)$.
- 15: $\langle \mathcal{R} \rangle = \text{SMatMul}(\langle \varphi \rangle, \langle K \rangle)$.
- 16: **return** $\langle \mathcal{R} \rangle$

The overall output $\langle \hat{H} \rangle = \{\langle \hat{h}_v \rangle\}_{v=0}^{|V|-1}$ from PRIVUF serves as the input information for subsequent PRIVRF. The main steps are outlined in Algorithm 6.

Secure readout function. After multiple executions (e.g., T times) of PRIVMF and PRIVUF to learn the final feature $\hat{h}_v^{(T)}$ for each node, PRIVRF can employ these $\hat{h}_v^{(T)}$ directly for classification, prediction, or other GNN-based inference tasks. Similar to PRIVMF, PRIVRF first comprises two types of NN layers: SecFC and SecReLU. In detail, the process involves $(\langle \hat{H}^{(1)} \rangle | \langle \hat{H}^{(T)} \rangle, W_R) \rightarrow [\text{SecFC} \rightarrow \text{SecReLU}] \times \iota \rightarrow \text{SecFC} \rightarrow \langle \tilde{\mathcal{H}}\tilde{\mathcal{H}}^{\iota+1} \rangle = \{\langle \tilde{\mathcal{H}}\tilde{\mathcal{H}}_v^{\iota+1} \rangle | v \in V\}$ and $(\langle \hat{H}^{(T)} \rangle, W_Z) \rightarrow [\text{SecFC} \rightarrow \text{SecReLU}] \times \iota \rightarrow \text{SecFC} \rightarrow \langle \tilde{\mathcal{H}}^{\iota+1} \rangle = \{\langle \tilde{\mathcal{H}}_v^{\iota+1} \rangle | v \in V\}$, where ι denotes the number of repetitions. Subsequently, PRIVRF utilizes SecSig and SEleMul to connect the results of the two network paths:

$$\langle \mathcal{R} \rangle = \text{SMatMul}(\text{SEleMul}(\text{SecSig}(\langle \tilde{\mathcal{H}}\tilde{\mathcal{H}}^{\iota+1} \rangle), \langle \tilde{\mathcal{H}}^{\iota+1} \rangle), \text{SDReLU}(\langle \tilde{\mathcal{H}}^{\iota+1} \rangle)).$$

The main steps are outlined in Algorithm 7.

After the execution of PRIVRF, the shared results are securely sent to the client, such as molecule laboratory. The laboratory then reconstructs the result to obtain the predicted properties of the new molecule.

C Correctness Analysis

Theorem 2 (The correctness of SMatMul). *For any input matrices $\langle X \rangle \in \mathbb{Z}_{2^l}^{m_1 \times m_2}$ and $Y \in \mathbb{Z}_{2^l}^{m_2 \times m_3}$, the SMatMul protocol yields the correct result $\langle Z \rangle \in \mathbb{Z}_{2^l}^{m_1 \times m_3}$ for the matrix multiplication $\langle Z \rangle_0 + \langle Z \rangle_1 = X \times Y$.*

Proof. Considering SMatMul consists of offline and online phases described in Algorithm 1, we first analyze the correctness of the offline phase of SMatMul protocol for generating shared $\langle A \times Y \rangle$, as outlined below:

$$\begin{aligned} \langle A \times Y \rangle_0 + \langle A \times Y \rangle_1 &= A \times (Y - B) + \langle C \rangle_0 + \langle C \rangle_1 \\ &= A \times Y - A \times B + C = A \times Y. \end{aligned}$$

Next, we prove the correctness of the online phase of the SMatMul protocol in producing shared $\langle Z \rangle = \langle X \times Y \rangle$, as outlined below:

$$\begin{aligned} \langle Z \rangle_0 + \langle Z \rangle_1 &= \langle A \times Y \rangle_0 + \langle A \times Y \rangle_1 + (X - A) \times Y \\ &= A \times Y + X \times Y - A \times Y = X \times Y. \square \end{aligned}$$

Theorem 3 (Correctness of SQuaPol). *Given any shared $\langle x \rangle \in \mathbb{Z}_{2^l}$ and plain the plaintext coefficients p_0, p_1, p_2 held by the one party, the SQuaPol protocol yields the correct result that satisfies $\langle z \rangle_0 + \langle z \rangle_1 = p_2x^2 + p_1x + p_0$.*

Proof. Considering SQuaPol consists of offline and online phases described in Algorithm 2, we first prove the correctness of the offline phase of SQuaPol protocol for generating shared $\langle p_2a^2 \rangle$, $2\langle p_2a \rangle$, and $\langle p_1a \rangle$, as outlined below:

- $\langle p_2a^2 \rangle_0 + \langle p_2a^2 \rangle_1 \rightarrow a_3e_3 + \langle c_3 \rangle_0 + p_2f_3 + \langle c_3 \rangle_1 \rightarrow a_3(p_2 - b_3) + p_2(a^2 - a_3) + c_3 \rightarrow a_3p_2 - a_3b_3 + p_2a^2 - a_3p_2 + c_3 \rightarrow p_2a^2$, where $e_3 \leftarrow p_2 - b_3, f_3 \leftarrow a^2 - a_3$.
- $\langle p_2a \rangle_0 + \langle p_2a \rangle_1 \rightarrow a_2e_2 + \langle c_2 \rangle_0 + p_2f_2 + \langle c_2 \rangle_1 \rightarrow a_2(p_2 - b_2) + p_2(a - a_2) + c_2 \rightarrow a_2p_2 - a_2b_2 + p_2a - a_2p_2 + c_2 \rightarrow p_2a$, where $e_2 \leftarrow p_2 - b_2, f_2 \leftarrow a - a_2$.
- $\langle p_1a \rangle_0 + \langle p_1a \rangle_1 \rightarrow a_1e_1 + \langle c_1 \rangle_0 + p_1f_1 + \langle c_1 \rangle_1 \rightarrow a_1(p_1 - b_1) + p_1(a - a_1) + c_1 \rightarrow a_1p_1 - a_1b_1 + p_1a - a_1p_1 + c_1 \rightarrow p_1a$, where $e_1 \leftarrow p_1 - b_1, f_1 \leftarrow a - a_1$.

Next, we prove the correctness of the online phase of the SQuaPol protocol in producing shared $\langle p_2af \rangle$ and $\langle z \rangle = \langle p_2x^2 + p_1x + p_0 \rangle$, as outlined below:

$$\begin{aligned} &\langle p_2af \rangle_0 + \langle p_2af \rangle_1 \quad \triangleright e_4 \leftarrow \langle x \rangle_1 - b_4, f_4 \leftarrow \langle p_2a \rangle_0 - a_4, f_5 \leftarrow \langle x \rangle_0 - a \\ &= a_4(e_4 + f_5) + \langle c_4 \rangle_0 + (f_5 + \langle x \rangle_1)(f_4 + \langle p_2a \rangle_1) + \langle c_4 \rangle_1 \\ &= a_4(\langle x \rangle_1 - b_4 + \langle x \rangle_0 - a) + \langle c_4 \rangle_0 + (\langle x \rangle_0 - a + \langle x \rangle_1)(\langle p_2a \rangle_0 - a_4 + \langle p_2a \rangle_1) + \langle c_4 \rangle_1 \\ &= a_4(x - a - b_4) + (x - a)(p_2a - a_4) + c_4 \quad \triangleright f \leftarrow x - a \\ &= a_4f - c_4 + p_2af - a_4f + c_4 = p_2af, \end{aligned}$$

$$\begin{aligned} &\langle z \rangle_0 + \langle z \rangle_1 \quad \triangleright f \leftarrow f_5 + \langle x \rangle_1 \\ &= \langle p_2a^2 \rangle_0 + 2(a_4(e_4 + f_5) + \langle c_4 \rangle_0) + \langle p_1a \rangle_0 + p_2f^2 + \\ &\quad \langle p_2a^2 \rangle_1 + 2(f(f_4 + \langle p_2a \rangle_1) + \langle c_4 \rangle_1) + p_1f + \langle p_1a \rangle_1 + p_0 \\ &= \langle p_2a^2 \rangle_0 + 2\langle p_2af \rangle_0 + \langle p_1a \rangle_0 + p_2f^2 + \langle p_2a^2 \rangle_1 + 2\langle p_2af \rangle_1 + p_1f + \langle p_1a \rangle_1 + p_0 \\ &= p_2f^2 + p_2a^2 + 2p_2af + p_1f + p_1a + p_0 = p_2x^2 + p_1x + p_0. \quad \square \end{aligned}$$

Theorem 4 (Correctness of SDRReLU). *Given AddSS-shared input $\langle x \rangle \in \mathbb{Z}_{2^l}$, $[z] \in \mathbb{Z}_2 = \text{SDReLU}(\langle x \rangle)$ is correctly computed and securely shared between the two parties if the final Boolean-shared output $[z]$ satisfy $[z]_0 \oplus [z]_1 = \text{DReLU}(x) = 1\{x < 2^{l-1}\}$.*

Proof. Likewise, SDRReLU consists of offline and online phases described in Algorithm 3. The correctness of the offline phase involves two aspects: first, invoking the key generation algorithm $\text{Gen}^<$ to produce a pair of keys $\{\tilde{k}_0, \tilde{k}_1\}$ and preparing $\text{Eval}^<$ for online evaluation; second, generating the boolean-shared value $[r] = [\text{MSB}(2^l - a) \oplus 1 \oplus c]$, where a and c are random numbers independent of x . Regarding the first aspect, the correctness of $\text{Gen}^<$ and $\text{Eval}^<$ has been directly proven in [36]. Now, let's demonstrate the correctness of the second aspect. By definition, $[r]_0 \oplus [r]_1 = \text{MSB}(2^l - a) \oplus 1 \oplus c$. Since $\text{MSB}(2^l - a)$ corresponds to the most significant bit of $\langle x \rangle_1 = 2^l - a$, we can express it as $\left\lfloor \frac{\langle x \rangle_1}{2^{l-1}} \right\rfloor$. Thus, we have:

$$[r]_0 \oplus [r]_1 = \left\lfloor \frac{\langle x \rangle_1}{2^{l-1}} \right\rfloor \oplus 1 \oplus c.$$

We then prove the correctness of the online phase of SDRReLU in producing Boolean-shared $[z]$, as outlined below:

$$\begin{aligned} [z]_0 \oplus [z]_1 &= [r]_0 \oplus \text{Eval}^<(0, \tilde{k}_0, e)_0 \oplus \left\lfloor \frac{x+a}{2^{l-1}} \right\rfloor \oplus [r]_1 \oplus \text{Eval}^<(1, \tilde{k}_1, e) \\ &= \left\lfloor \frac{x+a}{2^{l-1}} \right\rfloor \oplus [r]_0 \oplus [r]_1 \oplus \text{Eval}^<(0, \tilde{k}_0, e)_0 \oplus \text{Eval}^<(1, \tilde{k}_1, e) \\ &= \text{MSB}(\hat{x}) \oplus \text{MSB}(2^l - a) \oplus 1 \oplus c \oplus 1\{e < 2^l - a \pmod{2^{l-1}}\} \\ &= \text{DReLU}^{a,c}(x+a) = \text{DReLU}(x). \end{aligned}$$

where $e \leftarrow 2^{l-1} - (x+a \pmod{2^{l-1}}) - 1$. □

D Security Analysis

We describe PrivGNN using the hybrid model [10] in Theorem 1. In the hybrid model, interactions are identical to real interactions, except that sub-protocol executions are substituted with calls to corresponding trusted functionalities. A protocol invoking functionality \mathcal{F} is said to be in the \mathcal{F} -hybrid model. We briefly describe the two cases, then make use of the simulation-based technique to prove Theorem 1. The two cases are described below.

- **Corrupted client** ($\mathcal{P} \leftarrow \mathcal{C}$). A corrupted semi-honest client \mathcal{C} should learn no information about the well-trained model weights except the hyper-parameters of the model architecture. Formally, there should exist a PPT simulator $\text{Sim}_{\mathcal{C}}$ that can simulate the view $\text{View}_{\mathcal{P}}^{\Pi}$ of the corrupted client in real-world protocol execution: $\text{Sim}_{\mathcal{C}} \equiv_c \text{View}_{\mathcal{C}}^{\Pi}$.

- **Corrupted server** ($\mathcal{P} \leftarrow \mathcal{S}$). A corrupted and semi-honest server should learn nothing about the graph data D inputted by \mathcal{C} . Formally, there should exist a PPT simulator $\text{Sim}_{\mathcal{S}}$ that can simulate the view $\text{View}_{\mathcal{S}}^{\Pi}$ of the corrupted server in real-world protocol execution: $\text{Sim}_{\mathcal{S}} \equiv_c \text{View}_{\mathcal{S}}^{\Pi}$.

Proof. All proposed cryptographic protocols for different layers are stitched together to execute PrivGNN’s SI services and the input and output of each layer are in the AddSS domain. Assuming that adversary $\mathcal{A}_{\mathcal{C}}$ (or $\mathcal{A}_{\mathcal{S}}$) for the corrupted client (or server) attempts to obtain information of intermediate results, NN models’ W (or client’s data D). In the ideal world, we construct the simulators $\text{Sim}_{\mathcal{C}}$ and $\text{Sim}_{\mathcal{S}}$ to simulate the PrivGNN’s execution. The view from the simulator $\text{Sim}_{\mathcal{C}}$ (or $\text{Sim}_{\mathcal{S}}$) is indistinguishable from the real-world view of the corrupted client (or server), satisfying Definition 1.

- **Simulator for the corrupted client:**

Hybrid 0 In the offline phase of Π^{PrivGNN} , the simulator $\text{Sim}_{\mathcal{C}}$ generates $R_1, R_2, R_3, R_4 \xleftarrow{\$} \mathbb{Z}_{2^l}$ to emulate $W - B$, e , $\langle a \rangle_0$ and the FuncSS-based key k_0 in the real-world protocol execution. Since ideal functions reveal no information, the $\mathcal{A}_{\mathcal{C}}$ ’s view in this Hybrid 0 protocol is simulatable by $\text{Sim}_{\mathcal{C}}$, and thus, the $\mathcal{A}_{\mathcal{C}}$ gains no additional knowledge from Hybrid 0.

Hybrid 1 Further to Hybrid 0, $\mathcal{A}_{\mathcal{C}}$ executes the secure linear layers but asks the trusted third party to implement the ideal version of the non-linear protocols. The difference in the view $\text{View}_{\mathcal{A}_{\mathcal{C}}}^{\Pi}$ in Hybrid 1 and Hybrid 0 is only due to replacing the ideal function of secure linear layers with its actual execution. As analyzed above, the $\mathcal{A}_{\mathcal{C}}$ ’s view in linear layers is simulatable, i.e., $\text{Sim}_{\mathcal{C}} \equiv_c \text{View}_{\mathcal{A}_{\mathcal{C}}}^{\Pi}$. So, it is concluded that no PPT adversary can distinguish the execution of Hybrid 1 and Hybrid 0 from its view.

Hybrid 2 On the basis of Hybrid 1, besides the real execution of both offline phase and secure online linear protocols, $\mathcal{A}_{\mathcal{C}}$ also executes secure non-linear activation layers. Similarly, the difference in the view $\text{View}_{\mathcal{A}_{\mathcal{C}}}^{\Pi}$ in Hybrid 2 and Hybrid 1 is only due to replacing the ideal function of secure non-linear layers with its real execution. As we proved above, the $\mathcal{A}_{\mathcal{C}}$ ’s view in secure non-linear protocols is simulatable by $\text{Sim}_{\mathcal{C}}$ as the view of the building block FuncSS-based $\text{Eval}^>$ is simulatable. It is concluded that no PPT adversary can distinguish the execution of Hybrid 2 and Hybrid 1 from its view.

$\text{Sim}_{\mathcal{C}}$ conducts the above computations for every layer. Finally, $\text{Sim}_{\mathcal{C}}$ outputs the simulated shares $\langle O \rangle_0$ and $\langle O \rangle_1$ of the last activation layer. The reconstructed value of these two shares is uniformly distributed in ring \mathbb{Z}_{2^l} , the same as the real-world result execution. Therefore, the output $O = \mathcal{N}(W, D)$ from $\text{Sim}_{\mathcal{C}}$ is computationally indistinguishable to the corrupted client’s view $\text{View}_{\mathcal{A}_{\mathcal{C}}}^{\Pi}$.

- **Simulator for the corrupted server:**

Hybrid 0 In the offline phase of Π^{PrivGNN} , the simulator $\text{Sim}_{\mathcal{C}}$ generates $R_1, R_2, R_3, \xleftarrow{\$} \mathbb{Z}_{2^l}$ to emulate f , $\langle a \rangle_1$ and the FuncSS-based key k_1 in the

real-world protocol execution. Since ideal functions reveal no information, the \mathcal{A}_S 's view in this Hybrid 0 protocol is simulatable by Sim_S , and thus, the \mathcal{A}_S gains no additional knowledge from Hybrid 0.

Hybrid 1 Further to Hybrid 0, \mathcal{A}_S executes the secure linear layers but asks the trusted third party to implement the ideal version of the non-linear protocols. The difference in the view $\text{View}_{\mathcal{A}_S}^\Pi$ in Hybrid 1 and Hybrid 0 is only to replace the ideal function of secure linear layers with its actual execution. As analyzed above, the \mathcal{A}_S 's view in linear layers is simulatable, i.e., $\text{Sim}_S \equiv_c \text{View}_{\mathcal{A}_S}^\Pi$. So, it is concluded that no PPT adversary can distinguish the execution of Hybrid 1 and Hybrid 0 from its view.

Hybrid 2 based on Hybrid 1, besides the real execution of both offline phase and secure online linear protocols, \mathcal{A}_S also executes secure non-linear activation layers. Similarly, the difference in the view $\text{View}_{\mathcal{A}_S}^\Pi$ in Hybrid 2 and Hybrid 1 is only to replace the ideal function of secure non-linear layers with its real execution. As we proved above, the \mathcal{A}_S 's view in secure non-linear protocols is simulatable by Sim_S as the view of the building block FuncSS-based $\text{Eval}^\triangleright$ is simulatable. It is concluded that no PPT adversary can distinguish the execution of Hybrid 2 and Hybrid 1 from its view.

Sim_S conducts the above computations for every layer. Finally, Sim_S generates the simulated share $\langle O \rangle_1 \in \mathbb{Z}_{2^t}$ of the last activation layer, same as the result of real-world execution. Therefore, the output $\langle O \rangle_1$ from Sim_S is computationally indistinguishable to the corrupted server's view $\text{View}_{\mathcal{A}_S}^\Pi$.

Applying the sequential composition theory [9] to the above hybrid security analysis, we claim that PrivGNN is secure under the semi-honest adversaries model. Hence, the proof is finished. \square