GuardHFL: Privacy Guardian for Heterogeneous Federated Learning

Hanxiao Chen 1 2  Meng Hao 1 Hongwei Li 1 Kangjie Chen 3 Guowen Xu 3 Tianwei Zhang 3 Xilin Zhang 1

Abstract

Heterogeneous federated learning (HFL) enables clients with different computation and communication capabilities to collaboratively train their own customized models via a query-response paradigm on auxiliary datasets. However, such a paradigm raises serious privacy concerns due to the leakage of highly sensitive query samples and response predictions. We put forth GuardHFL, the first-of-its-kind efficient and privacy-preserving HFL framework. GuardHFL is equipped with a novel HFL-friendly secure querying scheme built on lightweight secret sharing and symmetric-key techniques. The core of GuardHFL is two customized multiplication and comparison protocols, which substantially boost the execution efficiency. Extensive evaluations demonstrate that GuardHFL significantly outperforms the alternative instantiations based on existing state-of-the-art techniques in both runtime and communication cost.

1. Introduction

As a promising variant of federated learning (FL), heterogeneous federated learning (HFL) (Li & Wang, 2019) enables clients equipped with different computation and communication capabilities to collaboratively train their own customized models via a query-response paradigm on auxiliary datasets. However, such a paradigm raises serious privacy concerns due to the leakage of highly sensitive query samples and response predictions. We put forth GuardHFL, the first-of-its-kind efficient and privacy-preserving HFL framework. GuardHFL is equipped with a novel HFL-friendly secure querying scheme built on lightweight secret sharing and symmetric-key techniques. The core of GuardHFL is two customized multiplication and comparison protocols, which substantially boost the execution efficiency. Extensive evaluations demonstrate that GuardHFL significantly outperforms the alternative instantiations based on existing state-of-the-art techniques in both runtime and communication cost.

1. Introduction

As a promising variant of federated learning (FL), heterogeneous federated learning (HFL) (Li & Wang, 2019) enables clients equipped with different computation and communication capabilities to collaboratively train their own customized models via a query-response paradigm on auxiliary datasets. However, such a paradigm raises serious privacy concerns due to the leakage of highly sensitive query samples and response predictions. We put forth GuardHFL, the first-of-its-kind efficient and privacy-preserving HFL framework. GuardHFL is equipped with a novel HFL-friendly secure querying scheme built on lightweight secret sharing and symmetric-key techniques. The core of GuardHFL is two customized multiplication and comparison protocols, which substantially boost the execution efficiency. Extensive evaluations demonstrate that GuardHFL significantly outperforms the alternative instantiations based on existing state-of-the-art techniques in both runtime and communication cost.

1. Introduction

As a promising variant of federated learning (FL), heterogeneous federated learning (HFL) (Li & Wang, 2019) enables clients equipped with different computation and communication capabilities to collaboratively train their own customized models via a query-response paradigm on auxiliary datasets. However, such a paradigm raises serious privacy concerns due to the leakage of highly sensitive query samples and response predictions. We put forth GuardHFL, the first-of-its-kind efficient and privacy-preserving HFL framework. GuardHFL is equipped with a novel HFL-friendly secure querying scheme built on lightweight secret sharing and symmetric-key techniques. The core of GuardHFL is two customized multiplication and comparison protocols, which substantially boost the execution efficiency. Extensive evaluations demonstrate that GuardHFL significantly outperforms the alternative instantiations based on existing state-of-the-art techniques in both runtime and communication cost.

1. Introduction

As a promising variant of federated learning (FL), heterogeneous federated learning (HFL) (Li & Wang, 2019) enables clients equipped with different computation and communication capabilities to collaboratively train their own customized models via a query-response paradigm on auxiliary datasets. However, such a paradigm raises serious privacy concerns due to the leakage of highly sensitive query samples and response predictions. We put forth GuardHFL, the first-of-its-kind efficient and privacy-preserving HFL framework. GuardHFL is equipped with a novel HFL-friendly secure querying scheme built on lightweight secret sharing and symmetric-key techniques. The core of GuardHFL is two customized multiplication and comparison protocols, which substantially boost the execution efficiency. Extensive evaluations demonstrate that GuardHFL significantly outperforms the alternative instantiations based on existing state-of-the-art techniques in both runtime and communication cost.

1. Introduction

As a promising variant of federated learning (FL), heterogeneous federated learning (HFL) (Li & Wang, 2019) enables clients equipped with different computation and communication capabilities to collaboratively train their own customized models via a query-response paradigm on auxiliary datasets. However, such a paradigm raises serious privacy concerns due to the leakage of highly sensitive query samples and response predictions. We put forth GuardHFL, the first-of-its-kind efficient and privacy-preserving HFL framework. GuardHFL is equipped with a novel HFL-friendly secure querying scheme built on lightweight secret sharing and symmetric-key techniques. The core of GuardHFL is two customized multiplication and comparison protocols, which substantially boost the execution efficiency. Extensive evaluations demonstrate that GuardHFL significantly outperforms the alternative instantiations based on existing state-of-the-art techniques in both runtime and communication cost.

1. Introduction

As a promising variant of federated learning (FL), heterogeneous federated learning (HFL) (Li & Wang, 2019) enables clients equipped with different computation and communication capabilities to collaboratively train their own customized models via a query-response paradigm on auxiliary datasets. However, such a paradigm raises serious privacy concerns due to the leakage of highly sensitive query samples and response predictions. We put forth GuardHFL, the first-of-its-kind efficient and privacy-preserving HFL framework. GuardHFL is equipped with a novel HFL-friendly secure querying scheme built on lightweight secret sharing and symmetric-key techniques. The core of GuardHFL is two customized multiplication and comparison protocols, which substantially boost the execution efficiency. Extensive evaluations demonstrate that GuardHFL significantly outperforms the alternative instantiations based on existing state-of-the-art techniques in both runtime and communication cost.

1. Introduction

As a promising variant of federated learning (FL), heterogeneous federated learning (HFL) (Li & Wang, 2019) enables clients equipped with different computation and communication capabilities to collaboratively train their own customized models via a query-response paradigm on auxiliary datasets. However, such a paradigm raises serious privacy concerns due to the leakage of highly sensitive query samples and response predictions. We put forth GuardHFL, the first-of-its-kind efficient and privacy-preserving HFL framework. GuardHFL is equipped with a novel HFL-friendly secure querying scheme built on lightweight secret sharing and symmetric-key techniques. The core of GuardHFL is two customized multiplication and comparison protocols, which substantially boost the execution efficiency. Extensive evaluations demonstrate that GuardHFL significantly outperforms the alternative instantiations based on existing state-of-the-art techniques in both runtime and communication cost.

1. Introduction

As a promising variant of federated learning (FL), heterogeneous federated learning (HFL) (Li & Wang, 2019) enables clients equipped with different computation and communication capabilities to collaboratively train their own customized models via a query-response paradigm on auxiliary datasets. However, such a paradigm raises serious privacy concerns due to the leakage of highly sensitive query samples and response predictions. We put forth GuardHFL, the first-of-its-kind efficient and privacy-preserving HFL framework. GuardHFL is equipped with a novel HFL-friendly secure querying scheme built on lightweight secret sharing and symmetric-key techniques. The core of GuardHFL is two customized multiplication and comparison protocols, which substantially boost the execution efficiency. Extensive evaluations demonstrate that GuardHFL significantly outperforms the alternative instantiations based on existing state-of-the-art techniques in both runtime and communication cost.
We introduce GuardHFL, the first-of-its-kind efficient and privacy-preserving HFL framework to address the above challenges. GuardHFL is built upon the standard HFL training paradigm (Li & Wang, 2019), which contains three stages: local training, querying, and local re-training. To provide privacy guarantees for HFL, GuardHFL proposes an HFL-friendly secure querying scheme based on lightweight secret sharing and symmetric-key primitives. The core of this scheme is the customized multiplication and comparison protocol, which substantially boosts execution efficiency. More precisely, (1) we design a new multiplication protocol based on pseudo-random functions, which is not only suitable for practical HFL scenarios with cross-client communication constraints, but also is efficient, e.g., only communicating 3 elements in an ϵ-bit ring. (2) We provide a customized comparison protocol based on the advanced adder, such as parallel prefix adder (PPA) (Harris, 2003). We utilize an adder to evaluate comparison operations because the adder contains only AND and XOR gates, where AND can be efficiently computed based on our multiplication protocol and XOR is cost-free. Besides, we provide formal security analysis for the designed protocols, and evaluate GuardHFL on different datasets (SVHN, CIFAR10, Tiny ImageNet), system configurations (IID and Non-IID training sets) and heterogeneous models. Extensive experiments demonstrate that GuardHFL outperforms the alternative instantiations based on the state-of-the-art techniques by about 4.4 ∼ 75.6× while ensuring the model utility.

2. Background

2.1. Heterogeneous Federated Learning

We briefly review the workflow of the standard HFL training paradigm (Li & Wang, 2019), where clients independently design their own unique models. Due to such model heterogeneity, clients cannot directly share model parameters with each other as in the traditional FL. Instead, they learn the knowledge of other models via a query-response mechanism, which is similar to the knowledge distillation technique (Hinton et al., 2015). To be more precise, each client $P_A$ (called the querying party) performs three-phase operations collaboratively with a server. (1) Local training: $P_A$ first trains the local model on his private dataset. (2) Querying: The server selects $C$ fraction of clients as the responding parties $P_A$ who provide predictions given the auxiliary querying dataset. The server receives the predictions results from these $P_A$, computes the aggregated result and returns it back to $P_Q$. (3) Local re-training: $P_Q$ then retrains the local model based on the private dataset, as well as the query samples and corresponding predictions.

GuardHFL is in line with the above paradigm with the additional benefit of privacy protection. The only difference lies in the acquisition of auxiliary query samples in the querying stage. In general HFL, there is a large public auxiliary dataset (used as query samples) that every party can access. However, considering the privacy limitation, such a dataset is hard to collect in real-world scenarios such as healthcare. To tackle this problem, in GuardHFL, each party can locally construct a synthesized querying set based on his private training samples, by utilizing existing data augmentation strategies (refer to Section 3.4).

2.2. Threat Model

As described in Section 1, in the querying phase of HFL, the query samples, response predictions and model parameters may contain sensitive information that is of interest to adversaries. In line with prior works (Phong et al., 2018; Sun & Lyu, 2021; Choquette-Choo et al., 2021), we consider an honest-but-curious adversary setting (Goldreich, 2009), where each entity (including the clients and the server) strictly follows the specification of the designed protocol but attempts to infer more knowledge about this private information of other clients. Moreover, to maintain its reputation and provide more services, the server does not collude with any clients, namely that an attacker either corrupts the server or a subset of clients but not both.

Security is modeled in the simulation paradigm (Canetti, 2001), which defines a real interaction and an ideal interaction. In the real interaction, the parties execute protocols according to the specification in the presence of an adversary $A$ and the environment $Z$. In the ideal interaction, the parties send their inputs to an ideal functionality that faithfully executes the operation. Secure inference requires that no environment can computationally distinguish between real and ideal interactions. The protocols in GuardHFL invoke multiple sub-protocols, and we use the hybrid model to describe them similar to prior works (Rathee et al., 2020; 2021). This is analogous to the real interaction, except that sub-protocols are replaced by the corresponding ideal functionalities. By convention, a protocol invoking a functionality $F$ is referred
to as the “F-hybrid model”.

2.3. Extend existing secure querying solutions to HFL

To provide privacy guarantees against honest-but-curious adversaries in Section 2.2, the clients and the server need to securely execute the querying process. Although this process consists of three entities (i.e., P_Q, the server and P_A), it is non-trivial to directly extend existing secure 3-party computation protocols (3PC) (Wagh et al., 2019; 2021; Knott et al., 2021; Tan et al., 2021) to instantiate this process. The main reason is the incapability of direct communication between P_Q and P_A in realistic HFL scenarios (Bonawitz et al., 2017; Bell et al., 2020), which hinders the usage of these 3PC solutions in HFL, unless we redesign the underlying protocols and make substantial modifications to their corresponding implementations. On the other hand, we can extend state-of-the-art 2PC solutions (Rathee et al., 2020; Huang et al., 2022) into this process via using the server as the communication medium with adaptive protocol modifications (refer to Appendix B.4). Unfortunately, as mentioned in Section 1, such extensions come at the cost of heavy computational and communication complexity. Motivated by these challenges, we design various lightweight and customized protocols to improve the efficiency of the secure querying phase, which show significant performance gains over extending the advanced 2PC schemes to HFL.

2.4. Cryptographic Primitives

Secret sharing. GuardHFL adopts the 2-out-of-2 arithmetic sharing scheme (Shamir, 1979; Demmler et al., 2015) over a ring \( \mathbb{Z}_{2^f} \). Specifically, the sharing algorithm takes \( x \) as input and outputs random shares \( [x]_0 \) and \( [x]_1 \) such that \( x = [x]_0 + [x]_1 \mod 2^f \). The reconstruction algorithm takes the two shares as input and outputs \( x = [x]_0 + [x]_1 \mod 2^f \). Besides, the boolean sharing is also employed in GuardHFL, where \( x \in \mathbb{Z}_2 \) is shared as \( [x]_0 \) and \( [x]_1 \) satisfying \( [x]_0 \oplus [x]_1 = x \). Arithmetic operations can be evaluated on secret-shared values. Given two secret-shared values \([x]_i\) and \([y]_i\) owned by two parties, addition and subtraction operations \([z]_i = [x]_i \pm [y]_i \) in \( \mathbb{Z}_{2^f} \) can be realized locally without any communication, i.e., each party \( P_i \) computes \([z]_i = [x]_i \pm [y]_i \mod 2^f \) for \( i \in \{0, 1\} \). In Appendix B.2, we review existing protocols for multiplication operations.

Pseudo-random Function. A pseudo-random function \( y \leftarrow \text{PRF}(Sk, x) \) is a deterministic function that takes a uniformly random seed \( Sk \) and a payload \( x \) as input and outputs a fixed-length pseudo-random string \( y \). The security of PRFs ensures that the output is indistinguishable from the uniform distribution. In GuardHFL, PRFs enable two parties to generate the same pseudo-random values without communication.

Algorithm 1 The GuardHFL framework

Input: Each client \( P_j, j \in [n] \), holds a private dataset \( D_j \) and a customized local model \( M_j \). \( \text{iter} \) is the number of iterations. \( B \) is the number of query samples and \( C \) is the set of selected responding parties in the current query-response phase.

Output: Trained models \( M_j, j \in [n] \).

1: for each \( j \in [n] \) do
2: \( P_j \) locally trains the local model \( M_j \) on \( D_j \) using the stochastic gradient descent optimization.
3: end for
4: for each \( \text{iter} \) do
5: for each querying party \( P^q_j, j \in [n] \) do
6: \( P^q_j \) randomly samples query data \( \{x_b\}_{b \in [B]} \) from the auxiliary querying dataset that is generated via the data argumentation strategies described in Section 3.4.
7: for each responding party \( P^r_i, i \in C \) do
8: \( P^q_j \) secret-shares \( \{x_b\}_{b \in [B]} \) with \( P^r_i \) and the server, based on the protocol \( \Pi_{\text{Share}} \) in Section 3.1.
9: \( P^r_i \) and the server jointly perform the secure model prediction protocol in Section 3.2.
10: \( P^r_i \) secret-shares the predictions \( \{[y^r_b]\}_{b \in [B]} \) to \( P^q_j \) and the server.
11: end for
12: \( P^q_j \) obtains \( \{y_b\}_{b \in [B]} \), where \( y_b = \sum_{i \in C} y^r_b[i] \), via the protocol \( \Pi_{\text{Agg}} \) in Section 3.3 with the server.
13: \( P^q_j \) re-trains \( M_j \) based on the query dataset \( \{x_b, y_b\}_{b \in [B]} \) and \( D_j \).
14: end for
15: end for

3. GuardHFL

GuardHFL is built upon standard HFL systems as discussed in Section 2.1 and enhances their privacy protection with cryptographic techniques. Figure 1 shows the overview of GuardHFL and the detailed description is given in Algorithm 1. Similar to vanilla HFL, it includes three phases: local training, secure querying and local re-training. Since local training and local re-training are standard HFL training processes without privacy issues, below we focus on formalizing our core construction, i.e., secure querying. As detailed in Section 2.3, extending existing secure querying solutions to HFL introduces expensive overhead due to the usage of heavy cryptographic primitives and the lack of customized protocols. To tackle this challenge, we propose a tailored secure querying scheme utilizing lightweight secret sharing and PRF techniques, which is decomposed into three steps: secure query-data sharing, secure model prediction and secure result aggregation.
In general, $P_Q$ first constructs querying samples locally using data argumentation strategies (Section 3.4). Since querying samples imply the semantic information of private training data, they cannot be directly exposed to the server and $P_A$ for prediction. Therefore, GuardHFL secret-shares query samples to the server and $P_A$ using the designed secure query-data sharing protocol (Section 3.1). Then given the secret-shared samples, $P_A$, $P_Q$ and the server jointly execute the proposed secure model prediction scheme (Section 3.2) to obtain the secret-shared inference logits. After that, the secure result aggregation protocol (Section 3.3) comes in handy, which takes as input the secret-shared logits and returns the aggregated results to $P_Q$.

### 3.1. Secure Query-data Sharing

To perform secure model prediction based on secret sharing techniques, $P_Q$ first secret-shares the query data $x$ with the server and $P_A$. Considering the communication constraint between $P_Q$ and $P_A$, we utilize PRFs to share $x$. Specifically, we first construct PRF seeds in pairs for $P_Q$, $P_A$ and the server, denoted as $Sk_{QA}$, $Sk_{SA}$, and $Sk_{SQ}$, which are used to generate the same random values between two parties without communication (refer to Figure 2 in Appendix B.2). After that, $P_Q$ can share $x$ using the protocol $\Pi_{Share}$ as shown in Figure 2. In particular, $P_Q$ non-interactively shares $[x]_0 = r$ with $P_A$ using PRFs on the seed $Sk_{QA}$. Then $P_Q$ computes $[x]_1 = x - r$ and sends it to the server.

**Theorem 3.1.** The protocol $\Pi_{Share}$ in Figure 2 securely realizes the functionality $F_{Share}$ in Table 5 in the $F_{PRE}$ hybrid model.

**Proof.** The formal proof is provided in Appendix C. □

### 3.2. Secure Model Prediction

In this step, the server and $P_A$ execute secure model prediction on the secret-shared query data with the assistance of $P_Q$. Similar to prior secure prediction schemes (Rathee et al., 2020; Huang et al., 2022), neural networks include three types of layers: linear layers, ReLU and MaxPool-

![Diagram](image)

**Figure 1.** The high-level view of GuardHFL

![Diagram](image)

**Figure 2.** Secure query-data sharing protocol $\Pi_{Share}$
Remark. Similar to Rathee et al. (2020); Huang et al. (2022), to be compatible with cryptographic protocols, we use the fixed-point representation, where the truncation technique is needed to prevent values from overflowing after each multiplication operation. Consistent with existing works (Mishra et al., 2020; Wagh et al., 2019), we use the truncation method from Mohassel & Zhang (2017). This method simply truncates the extra least significant bit (LSB) of a fixed-point value, albeit at the cost of a 1-bit error of the fractional part with the probability of $2^{\ell-1}$. Here, $\ell$ is the fractional prediction, and $\ell$ is the size of the secret-sharing ring. In GuardHFL, $\ell = 20$ and $\ell = 64$, thus an error of about $10^{-6}$ may occur with the probability of $2^{10}$, which is negligible.

**Theorem 3.2.** The protocol $\Pi_{\text{Matmul}}$ in Figure 3 securely realizes the functionality $F_{\text{Matmul}}$ in Table 5 in the $F_{\text{PRF}}$-hybrid model.

**Proof.** The formal proof is provided in Appendix C. □

![Algorithm 2 Secure MSB Protocol $\Pi_{\text{msb}}$](image)

**Algorithm 2 Secure MSB Protocol $\Pi_{\text{msb}}$**

**Input:** The arithmetic shares $[x]$  
**Output:** The boolean shares $[\text{MSB}(x)]^B$

1. $P_A$ and the server initiate vectors $g^*$ and $p^*$ with size $\ell$, where $g^*_i$ and $p^*_i$ are the $i$-th positions of $g^*$ and $p^*$ respectively.
2. Let $e_1, \ldots, e_1$ and $f_1, \ldots, f_1$ denote the bit strings of $[x]_0$ and $[x]_1$ respectively.
3. For $i \in [\ell]$, $P_A, P_Q$ and the server invoke an instance of $\Pi_{\text{Matmul}}$ with inputs $e_i$ and $f_i$ to obtain $[g^*_i]^B$.
4. For $i \in [\ell]$, $P_A$ sets $[p^*_i]^B = e_i$ and the server sets $[p^*_i]^B = f_i$.
5. for $r \in [2, \log \ell + 1]$ do
   6. if $r = 2$ then
      7. For $i \in [2, \frac{\ell}{2}]$, $P_A, P_Q$ and the server invoke two instances of $\Pi_{\text{Matmul}}$ with inputs $[g^*_{\ell-i-1}]^B$ and $[p^*_{\ell-i-1}]^B$ to obtain $[t_i]^B$. Then the server and $P_A$ set $[g^*_i]^B = [g^*_i]^B \oplus [t_i]^B$.
   8. Else
      9. For $i \in [1, \frac{\ell}{2}]$, $P_A, P_Q$ and the server invoke two instances of $\Pi_{\text{Matmul}}$ with inputs $[p^*_i]^B$ and $[p^*_{\ell-i-1}]^B$ to obtain $[t_i]^B$.
   10. End if
11. End for
12. $P_A$ sets $[\text{MSB}(x)]^B = e_{\ell} \oplus [g^*_0]^B$ and the server sets $[\text{MSB}(x)]^B_1 = f_{\ell} \oplus [g^*_0]^B$.

ReLU. The ReLU activation can be redefined as ReLU($x$) = $x - (1 + \text{MSB}(x))$, where MSB($x$) equals 0 if $x \geq 0$ and 1 otherwise. Thus, the evaluation of ReLU consists of a MSB (i.e., comparison) operation, followed by a multiplication operation. Below, we first provide a customized MSB protocol built on the advancedadder as parallel prefix adder (PPA) (Harris, 2003), and then describe the subsequent multiplication implementation.

**Customized MSB evaluation.** Given that $[x]_0 = e_1|| \ldots ||e_1$ and $[x]_1 = f_1|| \ldots ||f_1$, an $\ell$-bit adder is applied to perform the binary addition $e_i + f_i$ for each $i \in [\ell]$ to produce the carry bits $c_\ell, \ldots, c_1$. Thus, the MSB of $x$ can be learned via MSB($x$) = $e_\ell \oplus f_\ell \oplus c_\ell$, and the key task is to compute $c_\ell$. Obviously, we have $c_\ell = c_{\ell-1} \land (e_{\ell-1} \lor f_{\ell-1}) \land (e_{\ell-1} \land f_{\ell-1})$. Further, PPA defines a set of carry signal tuples $\{(g^*_i, p^*_i)\}_{i \in [\ell]}$, and sets $g^*_0 = e_1 \land f_1, p^*_0 = e_1 \lor f_1$ for each $i \in [\ell]$. Then, $c_\ell$ can be expressed as $c_\ell = g^*_{\ell-1} \lor (p^*_{\ell-1} \land g^*_{\ell-1})$.

The PPA-based solution was also used in existing works (Mohassel & Rindal, 2018; Patra et al., 2021), but in GuardHFL we give customized design for better efficiency.
are computed locally by $P_A$ and the server, respectively, and the last two items will be obtained by invoking the protocol $\Pi_{\text{Matmul}}$ twice. Moreover, for the evaluation of $y_i^0 = e_i \land f_i$ with $i \in [\ell]$, the parties only need to jointly invoke the protocol $\Pi_{\text{Matmul}}$ once to obtain $[y_i^0]$ since the server and $P_A$ own $f_i$ and $e_i$, respectively. Overall, this method contains $3\ell - 4$ AND gates, which totally requires $15\ell - 24$ bits of communication within $\log \ell + 1$ communication rounds. Algorithm 2 gives the detailed construction of our MSB protocol $\Pi_{\text{msb}}$.

**Theorem 3.3.** The protocol $\Pi_{\text{msb}}$ in Algorithm 2 securely realizes the functionality $\mathcal{F}_{\text{msb}}$ in Table 5 in the $\mathcal{F}_{\text{Matmul}}$-hybrid model.

**Proof.** The formal proof is provided in Appendix C. □

After obtaining $[\text{MSB}(x)]^P_i$, we need to compute $[x] \cdot (1 \ominus [\text{MSB}(x)]^P_i)$, i.e., the secret shares of ReLU($x$). Given that $z_0 = [\text{MSB}(x)]^P_i$ and $z_1 = 1 \ominus [\text{MSB}(x)]^P_i$, we have $\text{ReLU}(x) = ([x_0] + [x_1])(z_0 + z_1 - 2z_0z_1) = z_0[x_0] + z_1[x_1] + z_1(1 - 2z_0)[x_0] + z_0(1 - 2z_1)[x_1]$. The first two terms can be computed locally by $P_A$ and the server respectively, while the latter two terms are evaluated using our multiplication protocol $\Pi_{\text{Matmul}}$. Taking $z_1(1 - 2z_0)[x_0]$ as an example, the protocol $\Pi_{\text{Matmul}}$ takes as input $t_0 = (1 - 2z_0)[x_0]$ from $P_A$ and $z_1$ from the server, and outputs $[t_0z_1]_0$ to $P_A$ and $[t_0z_1]_1$ to the server. Finally, $P_A$ and the server learn $y = [\text{ReLU}(x)]$. The detailed secure ReLU protocol $\Pi_{\text{ReLU}}$ is shown in Figure 4.

**Theorem 3.4.** The protocol $\Pi_{\text{ReLU}}$ in Figure 4 securely realizes the functionality $\mathcal{F}_{\text{ReLU}}$ in Table 5 in the $\mathcal{F}_{\text{msb}}, \mathcal{F}_{\text{Matmul}}$-hybrid model.

**Proof.** The formal proof is provided in Appendix C. □

**Maxpooling.** Maxpooling can be evaluated using the protocol $\Pi_{\text{ReLU}}$ as well as a tree-based round optimization that recursively partitions the values into two halves and then compares the elements of each half. Precisely, the parties arrange the input of $m$ elements into a 2-ary tree with the depth of $\log m$, and evaluate the tree in a top-down fashion. In each comparison of two secret-shared elements $[x]$ and $[y]$, we observe that $\text{max}([x], [y]) = \text{ReLU}([x] - [y]) + [y]$. Hence the complexity of Maxpooling mainly comes from the evaluation of $m - 1$ ReLU. Besides, as illustrated in Wagh et al. (2019); Mishra et al. (2020), AvgPooling can be evaluated locally without communication.

**3.3. Secure Result Aggregation**

After the secure prediction on a sample, the predicted logit $[x_i]$ is secret-shared between the server and each responding party $P_A^i$, where $i \in [C]$ and $C$ is the set of responding parties in the current query-response phase. To prevent privacy leakage from a single prediction (Salem et al., 2019; Ganju et al., 2018; Yang et al., 2019), we return the aggregated logit to $P_Q$ via the secure aggregation protocol $\Pi_{\text{Agg}}$ in Figure 5. Specifically, $P_A^i$ and $P_Q$ first generate a random value $r_i$ based on PRFs. Then each $P_A^i$ computes and sends $[x_i]_0 - r_i$ to the server. The server sums all received values and sends the masked aggregation to $P_Q$, which will reconstruct the aggregated logits of the query sample. Notice that our secure aggregation protocol can be extended to output the aggregated label rather than the logit, using the above $\Pi_{\text{ReLU}}$ protocol.

**Theorem 3.5.** The protocol $\Pi_{\text{Agg}}$ in Figure 5 securely realizes the functionality $\mathcal{F}_{\text{Agg}}$ in Table 5 in the $\mathcal{F}_{\text{PRF}}$-hybrid model.

**Proof.** The formal proof is provided in Appendix C. □

![Figure 4. Secure ReLU protocol $\Pi_{\text{ReLU}}$](image)

![Figure 5. Secure result aggregation protocol $\Pi_{\text{Agg}}$](image)

**3.4. Discussion**

**Query data construction.** Unlike existing HFL works relying on auxiliary datasets as the query data (Choquette-Choo et al., 2021; Lin et al., 2020), we demonstrate the feasibility of model knowledge transfer in GuardHFL by constructing a synthesized query set based on private training data, to alleviate potential limitations (e.g., privacy, acquisition...
and storage) of public auxiliary datasets. A simple solution is to directly use the private training data to query, like well-studied knowledge distillation (Hinton et al., 2015). Moreover, we also construct a synthesized dataset via the mixup method (Zhang et al., 2018) (refer to Appendix A.2). The synthesized dataset construction is a universal and modular method, and it can be readily extended with advanced data augmentation strategies, such as cutout (DeVries & Taylor, 2017) and cutmix (Yun et al., 2019). Note that this process does not reveal any private information, since the samples are constructed locally by the querying party based on the local training data, without involving any other parties and their private datasets. We present some exploration and experiments in Appendix A.2 and Figure 10(c).

**GPU-friendly evaluation.** Our scheme is friendly with GPUs and can be processed by highly-optimized CUDA kernels (Tan et al., 2021). As discussed above, the cryptographic protocols of GuardHFL only involve simple vectorized arithmetic operations, rather than homomorphic encryption and garbled circuits in prior works (Rathee et al., 2020; Huang et al., 2022; Choquette-Choo et al., 2021). As a result, GuardHFL is suitable for batch querying (i.e., executing multiple querying at the same time) with a lower amortized cost. We evaluate the designed protocols on GPUs in Section 4.1 and show the advantage of GPU acceleration over CPUs in Figure 6.

### 4. Evaluation

**Datasets and models.** We evaluate GuardHFL on three image datasets (SVHN, CIFAR10 and Tiny ImageNet). By default, we assume independent and identically distributed (IID) training data among clients. We also simulate disjoint Non-IID training data via the Dirichlet distribution $\text{Dir}(\alpha)$ in Lin et al. (2020). The value of $\alpha$ controls the degree of Non-IID-ness, where a smaller $\alpha$ indicates a higher degree of Non-IID-ness. Moreover, we simulate the heterogeneity property in HFL. In particular, for SVHN and CIFAR10, we set the number of clients $n = 50$ and use VGG-7, ResNet-8 and ResNet-10 as the clients’ local models. For Tiny ImageNet, we set $n = 10$ and use ResNet-14, ResNet-16, and ResNet-18 architectures. Each model architecture is used by $n/3$ clients. Besides, the query data are constructed via two methods as shown in Section 3.4: using the private training data (Q-priv) or synthesizing samples (Q-syn) via mixup (Zhang et al., 2018).

**Experimental configuration.** Each of the entities, i.e., $P_Q$, $P_A$, and the server, is run on the Ubuntu 18.4 system with Intel(R) 562 Xeon(R) CPU E5-2620 v4(2.10 GHz) and 16 GB of RAM and NVIDIA 1080Ti GPU. Following existing works (Rathee et al., 2020; Tan et al., 2021), we set the secret-sharing protocols over a 64-bit ring $\mathbb{Z}_{2^{64}}$, and encode inputs using a fixed-point representation with 20-bit precision. The security parameter $\kappa$ is 128 in the instantiation of PRFs. Unless otherwise stated, we only report the performance on the GPU accelerator. More experimental setup is given in Appendix A.1.

#### 4.1. Efficiency

We report the efficiency of GuardHFL and compare it with CaPC (Choquette-Choo et al., 2021) and HFL instantiations based on state-of-the-art secure querying protocols (Rathee et al., 2020; Huang et al., 2022).

**End-to-end performance.** We show the extra overhead introduced by GuardHFL compared with the vanilla HFL system in the plaintext environment. This is caused by the secure querying phase, which consists of three steps described in Section 3. Table 1 reports the runtime of each step for different models and datasets. We observe that the cost is dominated by the secure model prediction step. Specifically, it takes 16.9 minutes to evaluate 5000 query samples securely on VGG-7 and CIFAR10, and only 11.32 second and 0.3 second are spent on the secure query-data sharing and secure result aggregation steps. More time is required to evaluate Tiny ImageNet because of larger input sizes and model architectures.

**Comparison with CaPC.** As described in Section 1, similar to GuardHFL, CaPC (Choquette-Choo et al., 2021) was proposed to support private collaborative learning utilizing the secure querying scheme (Boemer et al., 2019b), but

<table>
<thead>
<tr>
<th>Dataset</th>
<th># of Queries</th>
<th>1. Query data sharing</th>
<th>2. Secure prediction</th>
<th>3. Result aggregation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIFAR10 (SVHN)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>5.08</td>
<td>205.48</td>
<td>270.78</td>
<td>305.48</td>
</tr>
<tr>
<td>2500</td>
<td>7.16</td>
<td>511.63</td>
<td>657.83</td>
<td>758.16</td>
</tr>
<tr>
<td>5000</td>
<td>11.32</td>
<td>1019.12</td>
<td>1346.79</td>
<td>1521.23</td>
</tr>
<tr>
<td>Tiny ImageNet</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>9.87</td>
<td>2700.96</td>
<td>2971.47</td>
<td>3084.81</td>
</tr>
<tr>
<td>2500</td>
<td>18.78</td>
<td>6815.69</td>
<td>7217.28</td>
<td>7503.50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>CryptoNets</th>
<th>CryptoNets-ReLU</th>
<th>MLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>GuardHFL</td>
<td>CAPC</td>
<td>GuardHFL</td>
<td>CAPC</td>
</tr>
<tr>
<td>BS=128</td>
<td>0.03</td>
<td>17.75</td>
<td>0.24</td>
</tr>
<tr>
<td>BS=256</td>
<td>0.05</td>
<td>17.56</td>
<td>0.31</td>
</tr>
<tr>
<td>BS=512</td>
<td>0.07</td>
<td>17.62</td>
<td>0.50</td>
</tr>
<tr>
<td>BS=1024</td>
<td>0.13</td>
<td>17.77</td>
<td>0.89</td>
</tr>
</tbody>
</table>

To clearly illustrate the efficiency of GuardHFL, unless otherwise specified, we only report the overhead of one pair of querying and responding parties, as well as the server, in each iteration as described in Section 3.
with the unrealistic cross-client communication. In Table 2, we compare the secure querying process of GuardHFL with CaPC. Following CaPC’s setup, we evaluate three small-scale models (CryptoNets (Gilad-Bachrach et al., 2016), CryptoNets-ReLU (Gilad-Bachrach et al., 2016) and MLP (Boemer et al., 2019b)) on MNIST. We observe that GuardHFL is two orders of magnitude faster than CaPC on these three models. In terms of communication overhead, we provide a theoretical comparison. (1) For linear layers, CaPC requires to communicate 2 homomorphic ciphertexts within 2 rounds. GuardHFL needs communicating 3 ring elements (each with 64-bit). Note that the size of ciphertexts is much larger than the size of the ring elements. (2) For non-linear layers, e.g., ReLU, CaPC adopts the GC technique that requires 2 rounds with $8\ell\lambda - 4\lambda$ communication bits ($\lambda = 128$ and $\ell = 64$ in our setting) (Rathee et al., 2020). GuardHFL only requires communicating $15\ell - 3\log\ell - 12$ bits, a $70\times$ improvement over CaPC.

**Comparison with alternative instantiations.** To further demonstrate the efficiency of GuardHFL, we instantiate HFL based on advanced secure inference schemes, including Cheetah (Huang et al., 2022) and CryptTFlow2 (Rathee et al., 2020), using the methods described in Appendix B.4. Table 3 reports the comparison of the secure querying phase over CIFAR10. We observe that GuardHFL achieves a significant efficiency improvement on three heterogeneous models. For example, GuardHFL requires $57.4\sim 75.6\times$ less runtime and $8.6\sim 12.7\times$ less communication compared to CryptTFlow2. This is because the latter needs heavy HE-based multiplication and OT-based comparison operations within multi-communication rounds. Moreover, as shown in Section 2.3, extending 3PC protocols such as CryptGPU (Tan et al., 2021) to HFL is non-trivial. Nevertheless, since CryptGPU is one of the most advanced protocols under GPU analogs, we also compare with it assuming no communication limitation. We would like to mention that despite such an unfair comparison, GuardHFL still has performance advantages, i.e., roughly $2.1\times$ and $2.0\times$ in runtime and communication overhead, respectively.

**Impact of GPU acceleration.** To explore the impact of GPU acceleration, we evaluate GuardHFL on both CPU and GPU settings with different batch sizes of query data.

**Figure 6.** The runtime of GuardHFL on CIFAR10 under CPU/GPU with varied batch sizes of query data.

**Figure 7.** Accuracy curves of each heterogeneous model in GuardHFL as the number of iterations increases.

Figure 6 reports the results of VGG-style and ResNet-style networks on CIFAR10, where the GPU-based setting is always superior to the CPU analogs. As the batch size increases, the advantage of GPU-based protocols becomes more pronounced.

**4.2. Accuracy**

We report the accuracy of each heterogeneous model in GuardHFL, and explore the impact of various factors on the model accuracy such as the Non-IID setting, and the number of query data.

**End-to-end model accuracy.** Table 4 reports the model accuracy on three datasets in GuardHFL. We observe that for SVHN and CIFAR10, using Q-priv to query can increase the accuracy by about 4%, while the accuracy gain is about 10% when using 10K query samples with Q-syn. The main reason is that synthetic samples could provide a good coverage of the manifold of natural data. We also observe that more synthetic query data can achieve better performance from Table 4. Furthermore, with an increased number of participating clients, the accuracy improves slightly. Figure 7 shows the accuracy curves versus the number of iterations. We use SVHN and CIFAR10 as examples, as they converge much faster with better readable curves than Tiny ImageNet. We can observe that each heterogeneous model on both datasets can converge well based on two types of query data, and Q-syn shows better performance.
Table 4. The model accuracy of three datasets in GuardHFL on different ratios of participating clients (0.6, 0.8 and 1), and querying strategies (Q-priv and Q-syn).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>SVHN</th>
<th>CIFAR10</th>
<th>Tiny ImageNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ratio of clients</td>
<td>0.6</td>
<td>0.8</td>
<td>1</td>
</tr>
<tr>
<td>Before GuardHFL</td>
<td>7.54</td>
<td>56.86</td>
<td>22.26</td>
</tr>
<tr>
<td>Q-priv 2.5K</td>
<td>79.43</td>
<td>79.56</td>
<td>80.29</td>
</tr>
<tr>
<td>Q-syn 5.0K</td>
<td>80.69</td>
<td>80.32</td>
<td>81.69</td>
</tr>
<tr>
<td>Q-syn 7.5K</td>
<td>83.32</td>
<td>83.52</td>
<td>83.82</td>
</tr>
<tr>
<td>Q-syn 10K</td>
<td>84.54</td>
<td>84.78</td>
<td>85.12</td>
</tr>
</tbody>
</table>

Impact of Non-IID datasets. We illustrate the impact of Non-IID data on model accuracy in Figure 8, using CIFAR10 as an example. Figures 8(a), 8(b) and 8(c) visualize the distributions of Non-IID samples among clients with different Dir(α). When α = 100, the distribution is close to uniform sampling. When α = 0.5, the sample distribution of each class among clients is extremely uneven. From Figure 8(d) we observe that the higher the degree of Non-IID-ness, the lower the accuracy of models. Notably, GuardHFL can still significantly improve the performance of models under the Non-IID environment.

Impact of other factors. Due to space constraints, we report other experimental results in Appendix A.2. Briefly, Figure 9 shows the accuracy of each heterogeneous model with different numbers of query data. As the number of query data increases, accuracy increases by about 5%. Figures 10(a) and 10(b) illustrate the impact of different numbers of private training data on SVHN and CIFAR10. We observe that as the number of training data increases, the model performance is on the rise. Figure 10(c) details the impact of different query data construction methods. The results show that existing data augmentation strategies, such as Cutmix, Cutout, and Mixup, are effective for query data construction and improve the model accuracy.

5. Conclusion
We propose GuardHFL, an efficient and private HFL framework to formally provide the privacy guarantee of query samples, model parameters and response predictions. The core construction of GuardHFL is a customized secure querying scheme, in which two efficient multiplication and comparison protocols are designed based on lightweight secret sharing and PRF techniques. Extensive experiments demonstrate that GuardHFL outperforms prior art in both communication and runtime performance.

We consider the following future directions. (1) The communication cost of GuardHFL, which is also the limitation of the standard HFL paradigm, will be further improved. One possible mitigation is to extend the insight of the k-regular graph in FL (Bell et al., 2020) to HFL, and carefully design protocols from scratch. The main idea is that in FL it is enough for each party to speak to k < n − 1 other parties via the server, where n is the number of parties. (2) The security of GuardHFL will be improved to defeat more powerful malicious adversaries who may deviate from the protocol specifications. Unfortunately, even using the best-known techniques, the overhead will be increased by several orders of magnitude. We leave these improvements as future work.

Acknowledgements
The authors would like to thank the anonymous reviewers for their insightful comments. This work is supported by the National Key R&D Program of China under Grant 2022YFB3103500, the Key-Area Research and Development Program of Guangdong Province under Grant 2020B0101360001, the National Natural Science Foundation of China under Grants 6202016013 and 61972454, the Fundamental Research Funds for Chinese Central Universities under Grant ZYGX2020ZB027, the Natural Science Foundation of Chongqing, China under Grant cstc2019jcyjmsxmX0322, and Singapore Ministry of Education (MOE) AcRF Tier 2 MOE-T2EP20121-0006.
References


GuardHFL: Privacy Guardian for Heterogeneous Federated Learning


A. More Details on Experiment Evaluation

A.1. Experimental Setup

Datasets. We evaluate GuardHFL on the following standard datasets for image classification: (1) SVHN is a real-world image dataset obtained from house numbers in Google Street View images, which contains 600,000 32×32 RGB images of printed digits from 0 to 9. (2) CIFAR10 consists of 60,000 32×32 RGB images in 10 classes. There are 50,000 training images and 10,000 test images. (3) Tiny ImageNet contains 100,000 images of 200 classes downsized to 64×64 colored images. Each class has 500 training images, 50 validation images and 50 test images.

Training procedure. At the local training phase, each client trains the local model from scratch using stochastic gradient descent optimization. For SVHN, CIFAR10, and Tiny ImageNet, the loss function is cross-entropy with the learning rate of 0.5, 0.1, 0.01, respectively. Besides, the batch size is 256, 64 and 64, respectively. When the clients retrain the local model at the local retraining step, they use Adam optimizer for 50 epochs with learning rate of 2e-3 decayed by a factor of 0.1 on 25 epochs, where the batch size is 256 on SVHN, and 64 on both CIFAR10 and Tiny ImageNet.

A.2. Experimental Results

Impact of the number of query data. Figure 9 shows the accuracy of each heterogeneous model with different numbers of query data. We observe that GuardHFL consistently improves the model accuracy on the above datasets and heterogeneous models. Specifically, as the number of query data increases (from 2.5K to 10K), the accuracy of all three models increases by about 5%.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVHN</td>
<td>VGG7</td>
<td>70</td>
</tr>
<tr>
<td>CIFAR10</td>
<td>ResNet8</td>
<td>65</td>
</tr>
<tr>
<td>Tiny ImageNet</td>
<td>ResNet10</td>
<td>35</td>
</tr>
</tbody>
</table>

Figure 9. The accuracy of each heterogeneous model as the number of query data increases. Dashed lines represent the model accuracy before GuardHFL.

Impact of the number of private training data. Figures 10(a) and 10(b) illustrate the model accuracy of GuardHFL under different number of private training data on SVHN and CIFAR10. We can observe that as the number of training data increases, the model performance is on the rise. The main reason is that models can learn better on more training data and can construct more synthetic samples to query, so as to promote the transfer of model knowledge.

Impact of query data construction strategies. Figure 10(c) gives the model accuracy under three advanced data augmentation strategies, including cutmix (Yun et al., 2019), cutout (DeVries & Taylor, 2017), and mixup (Zhang et al., 2018). Cutmix (Yun et al., 2019) can be formulated as $\tilde{x}_{i,j} = M \cdot x_i + (1 - M) \cdot x_j$, where $M \in \{0, 1\}^{W \times H}$ is a binary mask matrix of size $W \times H$ to indicate the location of dropping out and filling from the two images $x_i$ and $x_j$. Cutout (DeVries & Taylor, 2017) augments the dataset with partially occluded versions of original samples. Mixup (Zhang et al., 2018) constructs synthetic samples by a convex combination on two images $x_i$ and $x_j$ with different coefficients $\lambda$, in which $\tilde{x}_{i,j} = \lambda \cdot x_i + (1 - \lambda) \cdot x_j$. We observe that those strategies are good choices for the query data construction in GuardHFL.

B. More Details on the Designed Scheme

B.1. Graphic depiction of end-to-end secure prediction scheme

Figure 11 gives a graphic depiction to illustrate the end-to-end secure prediction scheme, where the input is a secret-shared sample $[x]$. The whole process maintains the following invariant: the server and $P_A$ begin with secret shares of the
Figure 10. The model accuracy under different number of training data and query data construction methods on SVHN and CIFAR10.

input and after each layer, end with secret shares (over the same ring) of the output. The honest-but-curious security of GuardHFL will follow trivially from sequential composibility of individual layers. To be specific, $[x]$ first passes through a convolutional layer that can be formalized as the secure matrix multiplication operation $\omega_1 [x]$ ($\omega_1$ is the parameter) using the protocol $\Pi_{\text{matmul}}$ in Figure 3. The secret-shared outputs of this layer, i.e., $[y_1]_0$ and $[y_1]_1$, are obtained by $P_A$ and the server, respectively. For the subsequent ReLU layer, the protocol $\Pi_{\text{ReLU}}$ in Figure 4 is executed to return $[y_2]_0$ and $[y_2]_1$ to $P_A$ and the server respectively. Then Maxpooling on $[y_2]$ can be evaluated via the protocol $\Pi_{\text{ReLU}}$ as described in Section 3.2, to output the secret-shared values $[y_3]_0$ and $[y_3]_1$. When the secure prediction reaches the final fully-connected layer with inputs $[y_{n-1}]_0$ and $[y_{n-1}]_1$, the protocol $\Pi_{\text{matmul}}$ is executed. In the end, $P_A$ and the server obtain the secret-shared predicted logit, i.e., $[\text{logit}]_0$ and $[\text{logit}]_1$, respectively.

![Figure 11. End-to-end secure model prediction protocol. Green boxes represent linear layers (including convolutional/fully connected layers), and blue boxes represent non-linear layers (including ReLU/Maxpooling layers).](image)

### B.2. More details on cryptographic protocols

#### Secret sharing and Beaver’s multiplication protocol

As shown in Section 2.4, GuardHFL utilizes the arithmetic secret sharing primitive to protect the privacy of sensitive information. Given two secret-shared values $[x]$ and $[y]$ owned by two parties $P_i$, $i \in \{0, 1\}$, addition and subtraction operations ($[z] = [x] \pm [y]$ in $\mathbb{Z}_{2^t}$) can be realized locally without any communication, i.e., each $P_i$ computes $[z]_i = [x]_i \pm [y]_i \mod 2^t$. Besides, multiplication operation, i.e., $[z] = [x][y]$, is evaluated using Beaver’s multiplication triples (Demmler et al., 2015), where each triple refers to $(a, b, c)$ with the constraint $c = ab$. The triples are generated by cryptographic techniques (Demmler et al., 2015) or a trusted dealer (Riazi et al., 2018). Specifically, the multiplication over secret-sharing values can be evaluated in the following:

$$z = xy = ([x]_0 + [x]_1)([y]_0 + [y]_1) = \frac{P_0}{[x]_0[y]_0 + [x]_1[y]_1 + [x]_0[y]_1 + [x]_1[y]_0}$$

(2)

where for $i \in \{0, 1\}$, $[x]_i[y]_i$ can be computed locally, and $[x]_i[y]_{i-1}$ will be evaluated as follows. Taking $[x]_0[y]_1$ as an example, assuming $P_0$ and $P_1$ already hold $(a, [c]_0)$ and $(b, [c]_1)$, respectively. $P_0$ first sends $[x]_0 + a$ to $P_1$, while $P_1$ sends $[y]_1 + b$. Then the multiplication $[x]_0[y]_1$ can be computed locally by $P_0$ and $P_1$.
With non-trivial adaptation, existing secure 2-party querying schemes (Mishra et al., 2020; Rathee et al., 2020; Huang et al., 2022) can be extended to the communication-limited HFL setting. However, as shown in Section 4.1, such extension

B.3. Distinguish GuardHFL from other private settings.

GuardHFL is in line with the standard HFL paradigm (Li & Wang, 2019) with the additional benefit of privacy protection. As declared in the Introduction, GuardHFL is the first-of-its-kind privacy-preserving HFL framework, which is different from existing privacy-preserving training efforts. The latter can be divided into two categories: (1) privacy-preserving federated learning (Bonawitz et al., 2017; Bell et al., 2020), and (2) secure multi-party training (Tan et al., 2021; Keller & Sun, 2022). In the following, we give a detailed analysis.

Comparison to privacy-preserving federated learning. In the privacy-preserving federated learning (FL), each clients locally computes the gradient with his private database, and then a secure aggregation protocol is executed at the server side for aggregating the local gradients and updating the global model. However, as described in the Introduction, secure gradient aggregation cannot be realized in HFL, due to the heterogeneity of the clients’ models. Instead, GuardHFL follow a general HFL training paradigm (Li & Wang, 2019), which contains three steps: local training, querying, and local re-training. GuardHFL focuses on solving the privacy issue caused by the querying stage, and mainly proposes a query datasets generation (refer to Section 3.4) and a secure querying protocol (refer to Section 3.1 - Section 3.3).

Comparison to secure multi-party training. Secure multi-party training is typically an outsourced training setting, where resource-constrained clients outsource the entire training task to non-colluding multiple servers in a privacy-preserving manner. It requires a secure training protocol to finally yield a well-trained model. Different from secure multi-party training, GuardHFL enables clients to collaboratively and securely train their own customized models that may differ in size and structure. Moreover, as discussed above, the general HFL paradigm contains three steps: local training, querying and local re-training, where the local training and re-training stages are evaluated locally without revealing privacy. Therefore, the privacy-preserving HFL requires an HFL-friendly secure querying protocol (i.e., a customized inference protocol).

B.4. Extend existing 2PC protocols to HFL

With non-trivial adaptation, existing secure 2-party querying schemes (Mishra et al., 2020; Rathee et al., 2020; Huang et al., 2022) can be extended to the communication-limited HFL setting. However, as shown in Section 4.1, such extension
introduces expensive communication and computation overheads compared with our GuardHFL. In the following we divide these schemes into three categories, i.e., pure OT-based protocols, pure HE-based protocols, and hybrid protocols, and give the corresponding extension designs.

To extend the pure OT-based secure querying protocols such as CrypTFlow2 (Rathee et al., 2020) into HFL, \( P_Q \) first secret-shares query samples to the server and \( P_A \) using our protocol \( \Pi_\text{share} \) in Section 3.1. Then the server and \( P_A \) execute secure prediction based on the method proposed in Rathee et al. (2020). After that, adopting our secure aggregation protocol \( \Pi_\text{agg} \) in Section 3.3, the aggregated predictions will be returned to \( P_Q \). Although the OT-based schemes can be extended to HFL by combining the designed protocols in GuardHFL, it requires too many communication rounds due to the usage of OT primitives.

To extend the pure HE-based secure querying protocols (Gilad-Bachrach et al., 2016; Lee et al., 2021) to HFL, \( P_Q \) first encrypts the query samples and asks the server to pass them to \( P_A \). After that, \( P_A \) evaluates secure prediction non-interactively in the ciphertext environment. Then \( P_A \) sends encrypted predictions to the server. The server aggregates these encrypted predictions utilizing the additive homomorphism of HE and sends the aggregated results to \( P_Q \). Although it is trivial to extend the schemes equipped with the HE-based scheme to the communication-limited setting, they have two key problems: 1) activation functions need to be approximated as low-degree polynomials, which leads to serious accuracy loss; 2) the HE-based secure prediction protocol is difficult to extend to large-scale models due to the inherent high computation complexity.

For hybrid secure querying protocols that evaluates linear layers using HE and non-linear layers using OT or GC, such as Cheetah (Huang et al., 2022), we discuss the extension algorithm for each layer separately. For the linear layer, 1) \( P_Q \) encrypts query samples with HE and sends the ciphertext to \( P_A \) through the server. 2) \( P_A \) evaluates linear layers locally, and returns the encrypted masked outputs to \( P_Q \) through the server. 3) \( P_Q \) decrypts it to obtain the masked outputs, which are then sent to the server. As a result, the masked outputs of linear layers are secret-shared between the server and \( P_A \). For the non-linear layer, given that the server and \( P_A \) hold shares of the linear layer’s outputs, the two parties invoke the OT-based protocols to evaluate non-linear functions. Therefore, such an extension comes at the cost of heavy computational and communication complexity.

In summary, although existing 2PC protocols can be extended to the HFL setting with the cross-communication restriction, they sacrifice efficiency due to the lack of customized protocols and the adoption of heavy cryptographic primitives. Therefore, it is necessary to design an efficient cryptographic framework for HFL. And GuardHFL shows better adaptability and efficiency in the natural HFL scenarios.

C. Security analysis

Intuitively, GuardHFL only reveals the aggregated prediction to \( P_Q \) without the responding parties’ model parameters, and the server and \( P_A \) learn zero information about the querying parties’ data. This is because all intermediate sensitive values are secret-shared. Next, we give a formal analysis.

Our security proof follows the standard ideal-world/real-world paradigm (Canetti, 2001): in the real world, three parties (i.e., the server, \( P_Q \), and \( P_A \)) interact according to the protocol specification, and in the ideal world, they have access to an ideal functionality shown in Table 5. When a protocol invokes another sub-protocol, we use the \( \mathcal{F} \)-hybrid model for the security proof by replacing the sub-protocol with the corresponding functionality. Note that our proof works in the \( \mathcal{F}_\text{PRF} \)-hybrid model where \( \mathcal{F}_\text{PRF} \) represents the ideal functionality corresponding to the PRF protocol. The executions in both worlds are coordinated by the environment \( \text{Env} \), who chooses the inputs to parties and plays the role of a distinguisher between the real and ideal executions. We will show that the real-world distribution is computationally indistinguishable to the ideal-world distribution.

**Theorem C.1.** \( \Pi_\text{share} \) securely realizes the functionality \( \mathcal{F}_\text{share} \) in the \( \mathcal{F}_\text{PRF} \)-hybrid model.

**Proof.** Note that \( P_Q \) and \( P_A \) receive no messages in \( \Pi_\text{share} \), and hence the protocol is trivially secure against the corruption of \( P_Q \) and \( P_A \). Next, the only message that the server receives is the value \( [x]_1 \). However, \( [x]_1 = x - r \), where given the security of PRF, \( r \) is a random value unknown to the server. Thus, the distribution of \( [x]_1 \) is uniformly random from the server’s view and the information learned by the server can be simulated.

\footnote{To be more precise, this step is for the input layer. In the hidden layer, one of the input shares of the linear layer should be encrypted by the server and sent to \( P_A \).}
Theorem C.2. $\Pi_{\text{Matmul}}$ securely realizes the functionality $F_{\text{Matmul}}$ in the $F_{\text{PRF}}$-hybrid model.

Proof. Note that $P_Q$ receives no messages in $\Pi_{\text{Matmul}}$, and hence the protocol is trivially secure against corruption of $P_Q$. The only message that $P_A$ receives is the value $[x]_1 - b$. However, given the security of PRF, $b$ is a random value unknown to $P_A$. Thus, the distribution of $[x]_1 - b$ is computationally indistinguishable from a uniformly random distribution in $P_A$’s view, and the information learned by $P_A$ can be simulated. Next, during the protocol, the server learns $[c]_1$ and $w + a$. However, the distribution of $[c]_1$ and $w + a$ is computationally indistinguishable from a uniformly random distribution in the server’s view, since given the security of PRF, $a$ and $[c]_1$ are random values unknown to the server. Thus, the information learned by the server can be simulated.

Theorem C.3. $\Pi_{\text{msb}}$ securely realizes the functionality $F_{\text{msb}}$ in the $F_{\text{Matmul}}$-hybrid model.

Proof. Note that as shown in Section 3.2, $\Pi_{\text{msb}}$ just consists of AND gates, which is instantiated by the protocol $\Pi_{\text{Matmul}}$. Therefore, the msb protocol is trivially secure in the $F_{\text{Matmul}}$-hybrid model.

Theorem C.4. $\Pi_{\text{ReLU}}$ securely realizes the functionality $F_{\text{ReLU}}$ in the $(F_{\text{Matmul}}, F_{\text{msb}})$-hybrid model.

Proof. Note that as shown in Section 3.2, $\Pi_{\text{ReLU}}$ consists of $\Pi_{\text{msb}}$ and $\Pi_{\text{Matmul}}$. Therefore, the ReLU protocol is trivially secure in the $(F_{\text{Matmul}}, F_{\text{msb}})$-hybrid model.

Theorem C.5. $\Pi_{\text{Agg}}$ securely realizes the functionality $F_{\text{Agg}}$ in the $F_{\text{PRF}}$-hybrid model.

Proof. Note that $P_A$ receives no messages in $\Pi_{\text{Agg}}$, and hence the aggregation protocol is trivially secure against the corruption of $P_A$. Next, the only message that the server receives is the value $[x_i]_0 - r_i$. However, given the security of

<table>
<thead>
<tr>
<th>Table 5. The ideal functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input sharing functionality $F_{\text{Share}}$:</strong></td>
</tr>
<tr>
<td><strong>Input:</strong> $P_Q$: query data $x$.</td>
</tr>
<tr>
<td><strong>Output:</strong> $P_A$: $[x]_0 \in \mathbb{Z}_2^\ell$. Server: $[x]_1 = x - [x]_0 \mod 2^\ell$.</td>
</tr>
<tr>
<td><strong>Matrix multiplication functionality $F_{\text{Matmul}}$:</strong></td>
</tr>
<tr>
<td><strong>Input:</strong> Server: $[x]_1 \in \mathbb{Z}_2^\ell$. $P_A$: $[x]_0 \in \mathbb{Z}_2^\ell$, model parameter $\omega$.</td>
</tr>
<tr>
<td><strong>Output:</strong> Server: $[y]_1 \in \mathbb{Z}_2^\ell$. $P_A$: $[y]_0 = \omega x - [y]_1 \mod 2^\ell$.</td>
</tr>
<tr>
<td><strong>MSB functionality $F_{\text{msb}}$:</strong></td>
</tr>
<tr>
<td><strong>Input:</strong> Server: $[x]_1 \in \mathbb{Z}_2^\ell$. $P_A$: $[x]_0 \in \mathbb{Z}_2^\ell$.</td>
</tr>
<tr>
<td><strong>Output:</strong> Server: $[\text{msb}(x)]^B \in \mathbb{Z}_2^\ell$. $P_A$: $[\text{msb}(x)]^B_0 = \text{msb}(x) \oplus [\text{msb}(x)]^B_1 \mod 2$.</td>
</tr>
<tr>
<td><strong>ReLU functionality $F_{\text{ReLU}}$:</strong></td>
</tr>
<tr>
<td><strong>Input:</strong> Server: $[x]_1 \in \mathbb{Z}_2^\ell$. $P_A$: $[x]_0 \in \mathbb{Z}_2^\ell$.</td>
</tr>
<tr>
<td><strong>Output:</strong> Server: $[y]_1 \in \mathbb{Z}_2^\ell$. $P_A$: $[y]_0 = \text{ReLU}(x) - [y]_1 \mod 2^\ell$.</td>
</tr>
<tr>
<td><strong>Result aggregation functionality $F_{\text{Agg}}$:</strong></td>
</tr>
<tr>
<td><strong>Input:</strong> Server: $[x_i]_1 \in \mathbb{Z}_2^\ell$, $i \in [C]$. $P_A^i$: $[x_i]_0 \in \mathbb{Z}_2^\ell$.</td>
</tr>
<tr>
<td><strong>Output:</strong> $P_Q$: $y = \text{softmax}(\sum_{i=1}^{\lvert C \rvert} x_i)$.</td>
</tr>
</tbody>
</table>
Table 6. Comparison with prior works on properties necessary for federated learning

<table>
<thead>
<tr>
<th>Framework</th>
<th>Privacy</th>
<th>Usability</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data</td>
<td>Model</td>
<td>w/o Dataset</td>
</tr>
<tr>
<td></td>
<td>Privacy</td>
<td>Privacy</td>
<td>Heterogeneity</td>
</tr>
<tr>
<td>Bonawitz et al. (2017)</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Bell et al. (2020)</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Sav et al. (2021)</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Jayaraman &amp; Wang (2018)</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Li &amp; Wang (2019)</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Choquette-Choo et al. (2021)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Lin et al. (2020)</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Sun &amp; Lyu (2021)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Diao et al. (2021)</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>GuardHFL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

PRF, $r_i$ is a random value unknown to the server. Thus, the distribution of $[x_i]_0 - r_i$ is computationally indistinguishable from a uniformly random distribution in the server’s view and the information learned by the server can be simulated. After the aggregation, $PQ$ only learns the aggregated result $\sum_{i \in [C]} x_i$, but is unknown to each $x_i$. Therefore, the aggregation protocol is secure assuming the aggregation result will not reveal privacy.

D. Related Work

D.1. Heterogeneous federated learning

Federated learning (FL) achieves collaboration among clients via sharing model gradients. While successful, it still faces many challenges, among which, of particular importance is the heterogeneity that appear in all aspects of the learning process. This consists of model heterogeneity (Li & Wang, 2019) and statistical heterogeneity (Zhu et al., 2021). Statistical heterogeneity means that parties’ data comes from distinct distributions (i.e., Non-IID data), which may induce deflected local optimum. Solving the statistical heterogeneity has been extensively studied, such as Dinh et al. (2020); Zhu et al. (2021); Yurochkin et al. (2019); Fallah et al. (2020); Yoon et al. (2021), and is out of the scope of this work. Nevertheless, GuardHFL may help to alleviate the statistical heterogeneity due to the customized model design and the knowledge distillation-based aggregation rule.

Our work mainly focuses on the model heterogeneity that has been explored in recent works (Li & Wang, 2019; Lin et al., 2020; Choquette-Choo et al., 2021). In particular, Li & Wang (2019) proposed the first FL framework FedMD supporting heterogeneous models by combining transfer learning and knowledge distillation techniques. They first used a public dataset to pre-train the model and transferred to the task of private datasets. After that, to exchange the knowledge, each party used the public data and the aggregated predictions from others as carriers for knowledge distillation. To further improve model accuracy, Lin et al. (2020) proposed FedDF, similar to FedMD, which also used the model distillation technique for knowledge sharing. The difference is that they first performed FedAvg on parties’ local models and integrated knowledge distillation on the aggregated model. The dependence on model averaging leads to limited model heterogeneity. Later, Diao et al. (2021) focused on heterogeneous parties equipped with different computation and communication capabilities. In their framework, each party only updated a subset of global model parameters through varying the width of hidden channels, which reduces the computation and communication complexity of local models. However, this approach only learns a single global model, rather than unique models designed by parties. Moreover, as described in the Introduction, HFL suffers from several privacy issues, which are not considered in the above works. To address the privacy concern, GuardHFL provides end-to-end privacy-preserving HFL services.

The privacy-preserving techniques (i.e., secure aggregation) have been studied in federated learning (Bonawitz et al., 2017; Bell et al., 2020; Sav et al., 2021; Jayaraman & Wang, 2018). However, these techniques can not be directly extended to privacy-preserving HFL. Recently, Sun & Lyu (2021) proposed a noise-free differential privacy solution for HFL to guarantee each party’s privacy. However, as shown in Jayaraman & Evans (2019), there is a huge gap between the upper bounds on
privacy loss analyzed by advanced mechanisms and the effective privacy loss. Thus, differentially private mechanisms offer undesirable utility-privacy trade-offs. To further formally guarantee the privacy, Choquette-Choo et al. (2021) proposed CaPC, leveraging hybrid cryptographic primitives to realize confidential and private collaborative learning. Specifically, parties learn from each other collaboratively utilizing a secure inference strategy based on 2PC and HE protocols and a private aggregation method. As noted in the Introduction, the usage of heavy cryptography in CaPC leads to huge efficiency and communication overheads.

In summary, we give a comparison between prior FL works and GuardHFL in Table 6.

D.2. Secure neural network prediction

Since secure prediction is a critical component of GuardHFL, we briefly introduce its recent progress. Neural networks present a challenge to cryptographic protocols due to their unique structure and exploitative combination of linear computations and non-linear activation functions. In real scenarios, model prediction can be viewed as a two-party computation case, where one party with private query data wants to obtain prediction results from the other party who owns the model. During the whole process, the cryptographic protocols, typically HE and secure multi-party computation (MPC), are applied to ensure the confidentiality of the private data and model parameters.

Many existing works (Boemer et al., 2019b; Gilad-Bachrach et al., 2016; Brutzkus et al., 2019) support pure HE protocols for secure predictions. Typically, nGraph-HE (Boemer et al., 2019b,a) allows linear computations using the CKKS HE scheme. However, since a solution that builds upon HE protocols should be restricted to compute low degree polynomials, the non-polynomial activation functions, such as Maxpooling and ReLU, are forced to be evaluated in the clear by the party who owns private query data. This leaks the feature maps, from which adversaries may deduce the model weights. To solve this problem, Gilad-Bachrach et al. (2016) and Chen et al. (2019) use low-degree polynomial approximation to estimate non-linear functions. Unfortunately, such approximations affect the inference accuracy, and lead to huge computation overhead.

On the other hand, several libraries (Mohassel & Zhang, 2017; Knott et al., 2021; Wagh et al., 2019) employ primarily MPC techniques in secure predictions, which provide linear and non-linear protocols through the usage of oblivious transfer (OT), garbled circuit (GC) and secret sharing. In particular, CryptTen (Knott et al., 2021) performs linear operations based on \( n \)-out-of-\( n \) arithmetic secret sharing over the ring \( \mathbb{Z}_2 \). However, it requires a trusted third party to assist the secure prediction process, which is unrealistic in the real-world setting. CrytpGPU (Tan et al., 2021) builds on CrypTen, working in the 3-party setting based on the replicated secret sharing primitive. Although the scalability is poor, it introduces an interface to losslessly embed cryptographic operations over secret-shared values in a discrete domain into floating-point operations, which can implement the whole inference process on the GPU. Recently, Keller & Sun (2022) proposed a secure quantized training protocol that outperforms CrypGPU in the cryptographic performance. Unfortunately, this work cannot be applied in HFL and is not comparable to GuardHFL. The main reasons are: (1) GuardHFL and Keller & Sun (2022) are concerned with completely different tasks. GuardHFL builds on the standard HFL setting, where multiple parties collaboratively train individual models with the assistance of a server. Keller & Sun (2022) focuses on the outsourced training scenario, i.e., multiple servers jointly execute standard model training algorithm to obtain a well-trained model. (2) Moreover, the protocols in Keller & Sun (2022) are designed for a specific network architecture, i.e., quantized neural networks, which cannot be applied to the general models in GuardHFL. Therefore, Keller & Sun (2022) and GuardHFL are two fully orthogonal works.

In addition, many works focus on hybrid protocols, in which they combine the advantages of HE and MPC to improve prediction efficiency (Juvekar et al., 2018; Mishra et al., 2020; Rathee et al., 2020; Huang et al., 2022). For example, HE-transformer (Boemer et al., 2019a) employs nGraph-HE (Boemer et al., 2019b) for the evaluation of linear operations, and utilizes GCs of the ABY framework (Demmler et al., 2015) for the evaluation non-linear functions. However, GC is inefficient especially for large networks with thousands of parameters, since non-linear operations cannot be parallelized between query samples. After that, CrypTFlow2 (Rathee et al., 2020) implements two types of protocols for linear operations, i.e., the HE-based method and OT-based method. For non-linear layers, they also design efficient protocols based on OTs. More recently, Cheetah (Huang et al., 2022) improves CrypTFlow2 with customized HE-based linear protocols and improved OT-based non-linear protocols, which achieves the state-of-the-art efficiency. Nevertheless, as shown in Section 4.1, directly extending the protocols in Cheetah into HFL cannot obtain satisfying performance. Therefore, we propose GuardHFL, which avoids the adoption of heavy cryptographic tools like HE and OT, and only employs secret sharing and PRFs to achieve high efficiency, confidentiality and practicability.