

HiFGL: A Hierarchical Framework for Cross-silo Cross-device Federated Graph Learning

Zhuoning Guo

zhuoning.guo@connect.ust.hk

The Hong Kong University of Science and Technology
(Guangzhou), The Hong Kong University of Science and
Technology

Qiang Yang

qyang@cse.ust.hk

The Hong Kong University of Science and Technology

Duanyi Yao

dyao@connect.ust.hk

The Hong Kong University of Science and Technology

Hao Liu*

liuh@ust.hk

The Hong Kong University of Science and Technology
(Guangzhou), The Hong Kong University of Science and
Technology

ABSTRACT

Federated Graph Learning (FGL) has emerged as a promising way to learn high-quality representations from distributed graph data with privacy preservation. Despite considerable efforts have been made for FGL under either cross-device or cross-silo paradigm, how to effectively capture graph knowledge in a more complicated cross-silo cross-device environment remains an under-explored problem. However, this task is challenging because of the inherent hierarchy and heterogeneity of decentralized clients, diversified privacy constraints in different clients, and the cross-client graph integrity requirement. To this end, in this paper, we propose a Hierarchical Federated Graph Learning (HiFGL) framework for cross-silo cross-device FGL. Specifically, we devise a unified hierarchical architecture to safeguard federated GNN training on heterogeneous clients while ensuring graph integrity. Moreover, we propose a Secret Message Passing (SecMP) scheme to shield unauthorized access to subgraph-level and node-level sensitive information simultaneously. Theoretical analysis proves that HiFGL achieves multi-level privacy preservation with complexity guarantees. Extensive experiments on real-world datasets validate the superiority of the proposed framework against several baselines. Furthermore, HiFGL's versatile nature allows for its application in either solely cross-silo or cross-device settings, further broadening its utility in real-world FGL applications.

CCS CONCEPTS

• **Theory of computation** → Graph algorithms analysis; • **Computing methodologies** → Distributed artificial intelligence; • **Security and privacy** → Database and storage security.

* Corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
KDD '24, August 25–29, 2024, Barcelona, Spain.

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0490-1/24/08
<https://doi.org/10.1145/3637528.3671660>

KEYWORDS

federated graph learning, graph neural network, multi-level privacy preservation

ACM Reference Format:

Zhuoning Guo, Duanyi Yao, Qiang Yang, and Hao Liu*. 2024. HiFGL: A Hierarchical Framework for Cross-silo Cross-device Federated Graph Learning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain*. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3637528.3671660>

1 INTRODUCTION

Federated Learning (FL) has emerged as a transformative approach by enabling multiple parties to contribute to a shared machine learning model without the need for direct data exchange [34, 36]. Along this line, Federated Graph Learning (FGL) has been proposed to collaboratively train the Graph Neural Network (GNN) to extract distributed knowledge from interconnected subgraphs held privately by each party. Recently, FGL has been adopted to a wide range of application domains, such as finance [29], recommender system [20], and transportation [38].

Existing FGL approaches predominantly revolve around two paradigms [9]. 1) *Cross-silo FGL* [2, 31, 40], as illustrated in Figure 1 (a), formulates data silos as clients, each of which possessing a subgraph consists of nodes and connected edges. This paradigm is particularly relevant for institutions wishing to maintain private subgraphs while contributing to a global graph structure, such as cross-platform recommendation for E-commerce [16]. 2) *Cross-device FGL* [23, 25, 33], as shown in Figure 1 (b), regards each device as a client, which holding a node and its associated edges. It is more suitable for scenarios where numerous devices maintain private connections within a global graph, e.g., user-centric social networks [27]. Table 1 summarizes the privacy, utility, and efficiency tradeoff of existing FGL approaches.

Despite the success of the above two paradigms in unleashing the power of isolated graph data, none of them can be directly adopted to fully address the complexities of a mixed cross-silo cross-device environment, as demonstrated in Figure 1 (c). In real-world scenarios, institutions and users may have common privacy concerns but with varying privacy and utility requirements [42, 43].

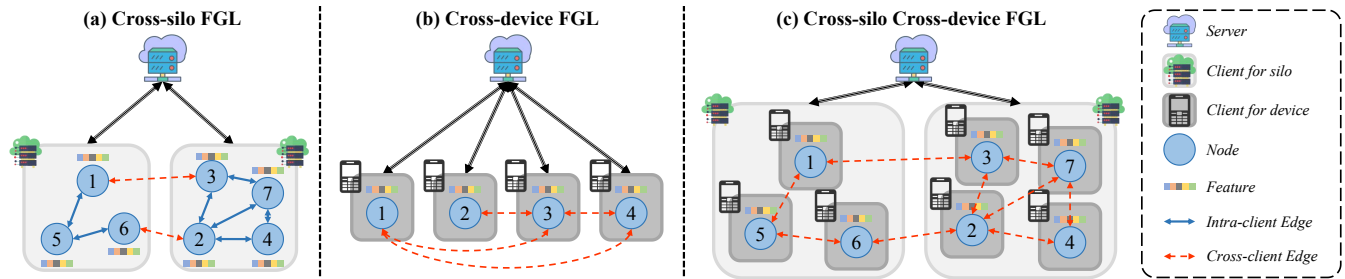


Figure 1: Illustration of three FGL paradigms: cross-silo, cross-device, and cross-silo cross-device FGL.

We illustrate the use case of FGL under the cross-silo cross-device setting via the following real-world application.

EXAMPLE 1. Anomaly detection in financial transactions. In this scenario, banks oversee customer transactions, forming a federated graph where each bank controls a subgraph of accounts (i.e., nodes) and transactions (i.e., edges). The collaborative analysis of the cross-bank federated graph can enhance the anomaly detection task. However, the transaction records are confidential, and regulations forbid the banks to disclose the subgraph structure. Meanwhile, customers are also unwilling to expose their sensitive information to institutions to avoid personal privacy leakage.

To bridge the gap, we investigate the cross-silo cross-device federated graph learning problem, where institutions and customer devices collaboratively optimize a GNN model under their diverged privacy constraints. However, three major technical challenges arise. **1) Inherent hierarchy and heterogeneity of decentralized clients.** The participants in cross-silo cross-device FGL naturally form a hierarchy, where an institution client may have a more comprehensive local structure of devices, and devices may preserve their local sensitive features. Besides, the cross-silo cross-device FGL involves clients with varying computational capabilities. Such hierarchy and heterogeneity pose significant challenges in designing a unified FGL framework that can effectively operate across a varied landscape. **2) Diversified privacy constraints in different clients.** As depicted in Table 1, privacy concerns in FGL vary across different types of clients. In a word, cross-silo FGL focuses more on subgraph-level privacy (i.e., the privacy of structures), while cross-device FGL weighs more on node-level privacy (i.e., the privacy of features). The varied privacy requirements necessitate a flexible approach that can adapt to the specific needs of different types of clients while ensuring the overall utility of the federated graph learning process. **3) Cross-client graph integrity.** In FGL, each client contributes to a portion of the overall graph. Maintaining the integrity of graph data across multiple clients is critical to the utility of the joint model. However, it is a non-trivial task to protect the cross-client graph information without sacrificing the model performance. For example, FedSage+ [40] protects cross-silo structures by generatively approximating edges across local subgraphs, while Glint [19] chooses to expose node embeddings to guarantee model utility. It is challenging to preserve graph privacy without sacrificing graph integrity.

In this paper, we propose a unified cross-silo cross-device framework, **Hierarchical Federated Graph Learning (HiFGL)**, for multi-level privacy preservation (i.e., both subgraph-level and node-level)

without sacrificing graph information integrity. Specifically, we first construct a hierarchical architecture comprising three key components: device-client, silo-client, and server. The hierarchical architecture enables federated graph learning with the flexibility of applying diversified privacy preservation strategies on different types of clients. Moreover, we propose a *Secret Message Passing (SecMP)* scheme for multi-level privacy protection. In particular, a Neighbor-Agnostic Aggregation protocol and a Hierarchical Lagrangian Embedding protocol are proposed to reduce subgraph-level and node-level privacy leakage, respectively. Furthermore, a tailored resource-efficient optimization algorithm is introduced for the unified framework. Notably, HiFGL can also be applied in solely cross-silo or cross-device scenarios, and is compatible with diverse FL algorithms (e.g., FedAvg [22], FedProx [15]) and GNN variants (e.g., GCN [13], GraphSage [7]), further broaden its utility in real-world FGL applications.

The main contributions of our work are listed as follows: 1) To our knowledge, HiFGL is the first framework tailored for the cross-silo cross-device federated graph learning problem, which has rarely been studied before. 2) We construct a hierarchical architecture to enable flexible privacy-preservation of heterogeneous clients without sacrificing graph integrity. 3) We propose a secret message passing scheme to simultaneously safeguard subgraph-level and node-level privacy against semi-honest adversaries during collaborative GNN training. 4) We theoretically analyze the privacy and complexity of HiFGL. The results show that our methods not only preserve subgraph-level and node-level privacy but also achieve information integrity with guaranteed complexity. 5) We conduct extensive experiments on real-world graph datasets, and the results demonstrate that HiFGL outperforms state-of-the-art baselines in learning more effective GNN models.

2 RELATED WORK

In this section, we review state-of-the-art federated graph learning approaches, including two paradigms: cross-silo and cross-device.

2.1 Cross-silo Federated Graph Learning

A common scenario of cross-silo FGL is that institutions collaboratively learn models while keeping local data private. Besides various GNN modules for improving effectiveness, existing cross-silo FGL works have different strategies for processing graph structure data.

The first strategy is to drop the cross-client edges to prevent data leakage across clients. An intuitive way is that each client holds a GNN (e.g., GCN [13]) and a subgraph without cross-client edges and

Table 1: Comparison of typical FGL works and HiFGL in the dimension of *Multi-level Privacy Preservation (including Subgraph-level and Node-level)*, *Information Integrity*, and *Efficiency*. We evaluate properties by *Low*, *Medium*, or *High* by considering how much a framework satisfies requirements relatively among baselines.

Model	FGL Paradigm	Multi-level Privacy Preservation		Information Integrity	Efficiency
		<i>Subgraph-level</i>	<i>Node-level</i>		
GCN [13]+FedAvg [22]	Cross-silo	Medium	Low	Medium	High
FedSage+ [40]	Cross-silo	Medium	Low	Medium	Low
FedPUB [2]	Cross-silo	Medium	Low	Medium	Medium
GraphFL [31]	Cross-silo	Medium	Low	Medium	Medium
FedGraph [4]	Cross-silo	High	Low	High	Low
Glint [19]	Cross-silo	Low	High	High	Medium
PPSGCN [39]	Cross-silo	Medium	Low	High	Low
FedCog [14]	Cross-silo	Medium	Low	High	Low
CNFGNN [23]	Cross-device	-	Medium	High	Medium
FedPerGNN [33]	Cross-device	-	High	High	Low
FedWalk [25]	Cross-device	-	High	High	Medium
Lumos [26]	Cross-device	-	High	High	Low
SemiDFEGL [27]	Cross-device	-	High	High	Medium
HiFGL (Ours)	Cross-silo Cross-device (Both)	High	High	High	Medium

trains a global model through FedAvg [22], a custom FL framework. In addition, FedSage+ [40] generates local nodes' neighbor features to offset the ignorance of cross-client edges due to subgraph-level privacy preservation, which improves predicting ability with significant extra training costs. FedPUB [2] focuses on the prediction improvement via personalized masked graph convolutional network, where their pairwise similarity is measured by subgraph representation. GraphFL [31] solves the semi-supervised graph learning problem on federated unconnected subgraphs, where node label domains vary and are not identically distributed. Another strategy is to maintain cross-client edges either stored by the server or clients. In this way, we have to face either less practicability or more privacy leakage. For example, FedGraph [4] uses a central server to keep cross-client edges and federally trains GNNs with graph sampling through huge communication for expanding neighbors between clients and the server. Unfortunately, the framework is not practical in the real world [18]. Glint [19] decentralized trains graph convolutional networks, where nodes are fully aware of cross-client neighbors. PPSGCN [39] leverages the graph sampling method to enhance efficiency and scalability, which hides node information but exposes nodes across clients. FedCog [14] decouples subgraphs according to intra- or cross-client edges to construct a border graph for each client, which is a bipartite graph between internal nodes and external nodes, with two separated graph learning operations.

We propose quantifying subgraph-level privacy leakage in Appendix A.1. The results demonstrate the unsolved issue of balancing subgraph-level privacy protection and cross-client graph integrity. Our approach aims to achieve two objectives simultaneously.

2.2 Cross-device Federated Graph Learning

Cross-device FGL assumes users hold private data and learn models without accessing private data. To name a few, CNFGNN [23] combines the spatiotemporal GNN model with FL, where the data

storage scheme hides the original information of nodes but exposes hidden states and neighboring nodes. FedPerGNN [33] applies GNNs for recommender systems where a user keeps a local user-item subgraph. A privacy-preserving graph expansion method anonymously acquires neighbor information with high communication costs. FedWalk [25] adopts the random walk algorithm for federated graph node embedding learning with node-level visibility for covering raw graph information. Lumos [26] utilizes local differential privacy and zero-knowledge protocol to protect node features' and degrees' privacy among decentralized devices. SemiDFEGL [27] collaborates ego graph-corresponded devices via a peer-to-peer manner for scalability improvement and communication reduction on recommendation tasks. In summary, existing works either utilize inefficient privacy-preserving schemes to align pairwise relationships or rely on a server with extreme representation exposure to achieve cross-client edge integrity. Our framework constructs a three-level architecture with multiple privacy preservation protocols to achieve effective and secure FGL.

3 PRELIMINARIES

3.1 Federated Graph Definition

In FGL, graphs are required to be distributively stored with privacy preservation. We focus on the scenario where a complete graph consists of subgraphs without overlapped nodes and nodes have their features and edges [40]. Let v denote a node associated with a multi-dimensional feature vector h_v , an edge $e = (v_i, v_j)$ is defined as a directed linkage from v_i to v_j . A subgraph is defined as $\mathbb{G}_s = (\mathbb{V}_s, \mathbb{E}_s)$, where \mathbb{V}_s is the node set and \mathbb{E}_s is the edge set. Note $\exists v_j \notin \mathbb{V}_s, e = (v_i, v_j) \in \mathbb{E}_s$, i.e., a node in a subgraph can connect with the nodes from other subgraphs. Then, we define the federated graph as a union of subgraphs under a FL setting.

DEFINITION 1. Federated Graph. Federated graph is denoted as $\mathbb{G} = (\mathbb{V}, \mathbb{E}) = \{\mathbb{G}_s | s = 1, 2, \dots, |\mathbb{G}|\}$, where \mathbb{V} is the node set, \mathbb{E} is the edge set, $\mathbb{G}_s = (\mathbb{V}_s, \mathbb{E}_s)$ is the s -th subgraph, and $|\mathbb{G}|$ is

the number of subgraphs. \mathbb{G} satisfies 1) $\mathbb{V}_1 \cup \mathbb{V}_2 \cup \dots \cup \mathbb{V}_{|\mathbb{G}|} = \mathbb{V}$, 2) $\mathbb{E}_1 \cup \mathbb{E}_2 \cup \dots \cup \mathbb{E}_{|\mathbb{G}|} \subseteq \mathbb{E}$, 3) $\mathbb{V}_i \cap \mathbb{V}_j = \emptyset, \forall 1 \leq i \leq |\mathbb{G}|, 1 \leq j \leq |\mathbb{G}|, i \neq j$, 4) $\mathbb{E}_i \cap \mathbb{E}_j = \emptyset, \forall 1 \leq i \leq |\mathbb{G}|, 1 \leq j \leq |\mathbb{G}|, i \neq j$.

In this work, we associate each node with a device and each subgraph with a data silo. Under the cross-silo and cross-device setting, nodes and graph structures may be placed in multiple heterogeneous clients for federated graph learning.

3.2 Problem Formulation

In this work, we aim to collaboratively train GNNs over distributed graph data in a federated way, which is formally defined below.

PROBLEM 1. Cross-silo Cross-device Federated Graph Learning. Given a federated graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$ consisting of the subgraph set $\{\mathbb{G}_s\}_1^{|\mathbb{G}|}$. We aim to learn the parameter θ_s of s -th silo-client for the global GNN model \mathcal{G} to minimize the global loss for downstream predictive or regressive tasks,

$$\{\theta_1, \dots, \theta_{|\mathbb{G}|}\} = \arg \min \sum \mathcal{L}(\mathcal{G}(\mathbb{G}_s; \theta_s), Y_s), \quad (1)$$

where $i = 1, 2, \dots, |\mathbb{G}|$ and \mathcal{L} denotes the loss function between estimation $\mathcal{G}(\mathbb{G}_s; \theta_s)$ and ground truth Y_s .

3.3 Threat Model

Here we define the threat model for cross-silo cross-device FGL, which concurrently considers nodes' and subgraphs' privacy. Specifically, we assume all parties are *semi-honest*, i.e., honest-but-curious, which means the adversary will try its best to obtain private information but not break protocols or cause malicious damage to the model's ability. We first define nodes' and subgraphs' privacy as node-level privacy and subgraph-level privacy, respectively.

DEFINITION 2. Node-level Privacy. A node usually represents a device of a user in FGL applications, which should not leak the privacy of the raw features and adjacent neighbors to other participants, i.e., devices and data silos.

DEFINITION 3. Subgraph-level Privacy. A subgraph corresponds to a data silo of an institution in FGL applications, which should not expose its subgraph to other data silos and devices.

As illustrated in Example 1, where banks correspond to subgraphs and accounts correspond to nodes, the corresponding participants in cross-silo cross-device FGL are inherently heterogeneous. Then, we introduce the potential attack types between heterogeneous clients, including node-node attack, subgraph-node attack, and subgraph-subgraph attack.

DEFINITION 4. Node-Node Attack. A node acts as an attacker, and another node acts as a defender. Nodes only see their features thus they hope to obtain neighbor embeddings to infer more information.

DEFINITION 5. Subgraph-Node Attack. A subgraph acts as an attacker, and a node acts as a defender. Subgraphs have no access to nodes' features and neighbors, and thus subgraphs are desired to utilize this information by cooperating with nodes. This cooperation may raise data leakage from nodes to subgraphs.

DEFINITION 6. Subgraph-Subgraph Attack. A subgraph acts as an attacker, and another subgraph acts as a defender. The adversary hopes to acquire other subgraphs as well as cross-client edges to enhance their subgraph.

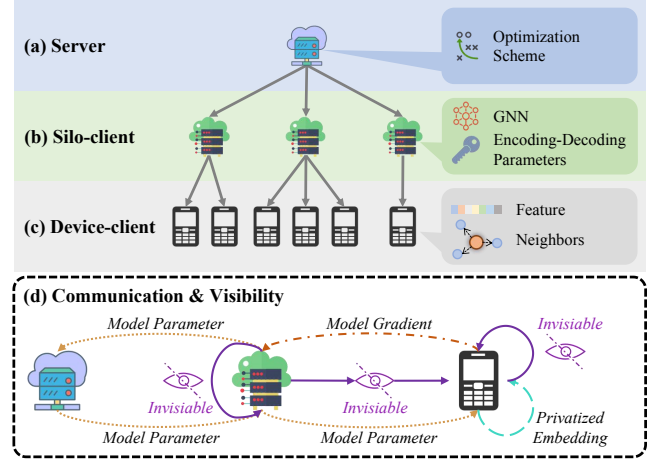


Figure 2: The architecture of the Hierarchical Federated Graph Learning framework.

Note we don't assume nodes will attack subgraphs. Due to the hierarchy of heterogeneous clients, the device can access the model from their corresponding data silo.

Overall, in this work, we aim to preserve node-level privacy against node-node attacks and subgraph-node attacks, and preserve subgraph-level privacy against subgraph-subgraph attacks.

4 FRAMEWORK

In this section, we present the proposed Hierarchical Federated Graph Learning (HiFGL) framework in detail. In particular, we first present the hierarchical architecture for heterogeneous clients. Then, we introduce the Secret Message Passing (SecMP) scheme to achieve multi-level privacy preservation. Finally, we detail the optimization scheme tailored for the framework.

4.1 The Hierarchical Architecture

We construct the hierarchical architecture of HiFGL consisting of three modules, i.e., device-client, silo-client, and server, in bottom-up order, as shown in Figure 2. Generally, under this hierarchy, modules are set to administrate the ones at a subordinate level and keep information private against the ones at a superordinate level¹. Specifically, the device-client holds local data and cooperates with its silo-client for learning without data exposure. A silo-client preserves local models that are federally optimized with the server, administrates device-clients containing graph data, and determines the privatization method. The server executes the federated optimization method for FL. In the following parts, we detail three modules and their communication and visibility.

4.1.1 Device-client. The device-client has two roles, including storing the ego graph of each node and computing node gradients with the silo-client, as shown in Figure 2 (c).

1) Storage. A device-client stores an ego graph consisting of the target node's features and its neighbors.

¹In this paper, we use "the device-client's silo-client" to denote "the silo-client that administrates the device-client" and "the silo-client's device-client" to denote "the device-clients administrated by the silo-client".

DEFINITION 7. Ego Graph. An ego graph stores j -th node's feature $h_j^{(0)}$, label y_j , and 1-hop directed neighbors. We denote the ego graph as $\tilde{g}_j = (h_j^{(0)}, \mathcal{N}_j)$.

We store data separately in each device for two reasons. First, storing data at the device level can prevent node features from being seen directly by adversaries at the silo or server level. Second, cross-client edges are risky for exposing the silo's node identities. Suppose a directed cross-client edge connects a source and a target node from two silos. However, storing the edge in any one of the two silos will violate subgraph-level privacy requirements because there will be at least one node to be seen by a silo-client that is not its own. To address this challenge, we propose allowing two devices corresponding to the two nodes to have access to this cross-client edge in order to reduce the visibility of silos.

To construct ego graphs, we allow any two devices to connect in a peer-to-peer manner to ensure that the connected edges are only known by these two device-clients. For instance, two users can build a friend relationship without the awareness of the social network administrator. In the semi-honest setting, we consider the connected edges to be true. In this way, we can construct the graph in a privacy-preserving and decentralized way.

2) Gradients computation Considering the limited computing power and local data size, instead of optimizing the full parameters of the local model, the device-clients are only required to calculate gradients associated with their local data. Specifically, for a model \mathcal{G} parameterized by θ , the j -th device-client computes the corresponding gradients by $\nabla_j \theta = \nabla f_l(\mathcal{G}_\theta(h_j^{(0)}), y_j)$, where $f_l(\cdot)$ is the loss function. These gradients are not considered as private information and can be shared with the corresponding silo-client.

4.1.2 Silo-client. As shown in Figure 2 (b), the silo-client is a module between the server and the device-client, which is in charge of local model optimization based on data in its device-clients. We introduce two roles of each silo-client C_i , including storage and model update.

1) Storage. The silo-client stores three elements, the subgraph, the local model, and encoding-decoding parameters. First, C_i indirectly keeps a federated subgraph \tilde{G}_i consisting of federated ego graphs distributed in its device-clients $\mathbb{D}_i = \{D_j | j = 1, 2, \dots\}$. We define the federated subgraph as below.

DEFINITION 8. Federated Subgraph. Federated subgraph is denoted as $\tilde{G} = \{D_j | j = 1, 2, \dots\}$, where each D_j stores a \tilde{g}_j , the j -th ego graph whose node v_j is in subgraph G . \tilde{G} only keeps the identities of the subordinate device-clients, but does not store any features and neighbor information.

The federated subgraph avoids exposure of node features and graph structures and therefore protects device-level privacy in the federated learning process.

Moreover, since the silo-client usually has more powerful computing resources, we let C_i keep the local GNN \mathcal{G}_i for optimization. Additionally, C_i also preserves a set of individual encoding-decoding parameters μ_i generated by a polynomial function \mathcal{F} for privatizing node embeddings. Specifically, we define the parameters $\mu = \{\alpha_1, \dots, \alpha_{T+1}, \beta_1, \dots, \beta_{T+1}, z_2, \dots, z_{T+1}\}$, where $\alpha_1, \dots, \alpha_{T+1}$,

$\beta_1, \dots, \beta_{T+1}$ are $2T + 2$ distinct elements from the finite field F^2 , satisfying $\{\alpha_1, \dots, \alpha_{T+1}\} \cap \{\beta_1, \dots, \beta_{T+1}\} = \emptyset$, and z_2, \dots, z_{T+1} are T uniformly distributed vector.

2) Model update. The silo-client C_i is responsible to optimize \mathcal{G}_i based on gradients computed by device-clients in \mathbb{D} . Specifically, C_i collects the gradients set $\{\nabla_j \theta_i | D_j \in \mathbb{D}_i\}$ and compute the average gradients as $\nabla \theta_i = \frac{1}{|\mathbb{D}_i|} \sum_{D_j \in \mathbb{D}_i} \nabla_j \theta_i$, where θ_i is parameters of \mathcal{G}_i . The averaged gradients can be further utilized for optimization on \mathcal{G}_i . For example, we can leverage gradient descent as $\theta_i = \theta_i - \rho \nabla \theta_i$ where ρ is the learning rate.

4.1.3 Server. The server coordinates the entire framework as shown in Figure 2 (a), whose role is to enable federated optimization schemes (e.g., FedAvg [22]) among silo-clients. In practice, the server can be an administrator of silo-clients to supervise their sensitive activities in optimization. For instance, in a finance scenario, the server can be a banking authority (e.g., European Banking Authority), silo-clients are banks, and device-clients are different users. The federation is under the authority's control to prevent potential attacks among banks and users.

4.1.4 Communication and Data Visibility. The cross-level communication and data visibility among three different modules are shown in Figure 2 (d). First, the server and silo-clients can transfer model parameters during federated optimization. Second, silo-clients are forbidden to exchange information, including device identities, models, and encoding-decoding parameters. Third, during the joint learning process, the device-client can access the model of the silo-client and send local gradients to the silo-client for model update, while the silo-client can not directly access the device-client's data. Last, device-clients communicate with each other through privatized embeddings without exposure of node features and neighborhood information.

4.2 Secret Message Passing

We further propose a novel graph learning method, SecMP, to preserve multi-level privacy without information loss. In particular, we develop the *neighbor-agnostic aggregation* and *hierarchical Lagrangian embedding* against potential subgraph-node and subgraph-subgraph attacks. Specifically, the neighbor-agnostic aggregation decomposes the feature extraction function in conventional GNN into two individual steps to mutually cover neighbor information between any connected node pair, preventing subgraph-level sensitive information exchange. Meanwhile, the hierarchical Lagrangian embedding utilizes Lagrange polynomial functions to mask sharing node embeddings during message passing recoverably.

4.2.1 Neighbor-agnostic aggregation. The aggregation step in GNNs may leak subgraph-level privacy, since such a process requires accessing structural information from cross-client neighbors [10, 13, 30]. Take GCN [13] as an example, the aggregation process can be denoted by $h_u = \sum_{v \in \mathcal{N}_u} (1/(\sqrt{|\mathcal{N}_u|} \sqrt{|\mathcal{N}_v|})) h_v \mathbf{W} + b$, where h is node embedding, b and \mathbf{W} are learnable parameters, and $|\mathcal{N}|$ are the number of neighbors. Calculating $|\mathcal{N}_u|$ and $|\mathcal{N}_v|$ requires knowing precise numbers of u and v , which is risky if the computation only inside either device-client u or v .

²Following [37], we assume that F is a finite field with 11 elements.

To address this problem, we devise the neighbor-agnostic aggregation to split GNN aggregating operations into source-side and target-side steps, which makes the aggregation process agnostic with neighbors' information. Specifically, we operate a message passing from node v to node u in the source device-client and target device-client that correspond to two nodes, respectively. First, device-client v will calculate a source-side function on h_v to get \hat{h}_v and pass it to device-client u . Then, device-client u will calculate a target-side function to update its embedding. The source-side function can be defined as

$$\hat{h}_v = (1/\sqrt{|\mathcal{N}_v|}) \cdot h_v, \quad (2)$$

and the target-side function can be defined as

$$h_u = b + (1/\sqrt{|\mathcal{N}_u|}) \sum_{v \in \mathcal{N}_u} \hat{h}_v \cdot \mathbf{W}. \quad (3)$$

In this way, we can preserve the privacy of both device-client u and v 's neighbor lists by delegating operation-related private data to be computed within device-clients.

4.2.2 Hierarchical Lagrangian embedding. Sharing embeddings with nodes' neighbors may induce severe node-level privacy leakage. Therefore, we privatize device-clients' information without information loss by introducing Lagrange Coded Computing (LCC) [37], a secret sharing technique in GNN optimization. Specifically, our method follows an encoding-decoding workflow based on a group of parameters μ introduced in Section 4.1.2. In the encoding step, we use Lagrange interpolation polynomial to encode data to coded versions. In the decoding step, the aggregated coded vectors can be decoded without any information loss.

Encoding. Any ego device-clients D_s edged with a target device-client D_t of silo-client C will access μ and use them for constructing a polynomial function based on D_s 's embedding³ h_v as

$$g_v(x) = h_v \cdot \prod_{k \in [T+1] \setminus \{1\}} \frac{x - \beta_k}{\beta_1 - \beta_k} + \sum_{j=2}^{T+1} z_j \cdot \prod_{k \in [T+1] \setminus \{j\}} \frac{x - \beta_k}{\beta_j - \beta_k}, \quad (4)$$

D_s generates $T+1$ coded embeddings $\tilde{h}_v = g_v(\alpha)$, $\alpha \in \alpha_1, \dots, \alpha_{T+1}$.

Decoding. D_t will receive $T+1$ coded embeddings⁴ h_1, \dots, h_{T+1} from each source neighbor device-client $D_s \in \mathcal{N}_{D_t}$. D_t will delegate the aggregated coded embedding to C for decoding by calculating values of the Lagrange interpolated polynomial function at $x = \beta_1$.

4.2.3 Overall workflow. We summarize the overall workflow of SecMP that incorporates neighbor-agnostic aggregation and hierarchical Lagrangian embedding for subgraph-level and node-level privacy protection. As illustrated in Figure 3, SecMP follows general GNN pipeline [6] in three major steps, *i.e.*, *privatized message*, *secure aggregation*, and *neighbor-agnostic update*.

1) Privatized message. We first emit encoded embedding to the target device-client. In this way, device-clients are only aware of neighbors' coded embeddings instead of any concrete features. This step includes three sub-steps. a) Device-clients project raw features into node embeddings on the source-side. b) Device-clients request encoding-decoding parameters from neighbors' silo-clients. For each neighbor, the device-client will have one corresponding group

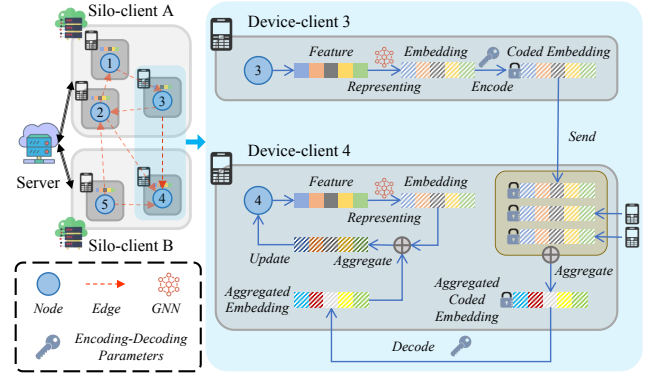


Figure 3: Overall workflow of Secret Message Passing.

of parameters for encoding. c) Device-clients encode embeddings and send encoded embeddings to both intra-client and cross-client target neighbor device-clients.

2) Secure aggregation. Then, the encoded embeddings from multiple neighbors are aggregated to derive a unified embedding. The silo-client can decode the aggregated embeddings without accessing the embeddings of each individual node, while device-clients cannot decode encrypted embeddings, therefore protecting individual embeddings. Two sub-steps of secure aggregation are elaborated below. a) Device-clients aggregate received encoded embeddings into a unified embedding. b) Device-clients send the aggregated embeddings to their silo-clients, which can use its encoding-decoding parameters to decode the aggregated embedding and send it back to device-clients.

3) Neighbor-agnostic update. After secure aggregation, the target device-clients obtain the aggregated embeddings computed from the source side, and update their embeddings without requiring access to neighbors' structure information.

Note that an exceptional case is that when a node has only one neighbor, the corresponding device-client can get the node embedding after aggregation because the decoded aggregated embedding is exactly the embedding of the neighbor. A possible solution is to perturb embeddings by Differential Privacy (DP) techniques [5] to cover sensitive information. The DP-based masking operation also prevents attacks that infer original information by gradients or embeddings [44]. Besides, we build HiFGL on the Trusted Execution Environment (TEE) [24] and ensure the transition is anonymous for any receivers to eliminate exposure.

4.3 Privacy-preserving Optimization

Then, we present the privacy-preserving optimization algorithm for HiFGL. Without loss generality, we assume the local model is GCN for the node classification task, and the federated optimization algorithm is FedAvg. The full pipeline is reported in Appendix A.2. We specify two critical steps in training, including local optimization and federated optimization.

Local optimization. We optimize local models based on updated embeddings computed by SecMP. Specifically, we first calculate estimated classification probability as $\hat{y} = \text{SoftMax}(h^{(k)})$ where k is the index of the last GNN layer. Then each device-client applies cross entropy loss on predicted labels \hat{y} and true labels y to calculate

³The embedding can be computed by a source-side function as Equation 2.

⁴The embedding can be computed by a target-side function as Equation 3.

the loss value $l_j = \sum_q y_q \log \hat{y}_q$ and generate gradients ∇l_j . After that, gradients from device-clients will be transmitted to their silo-clients. Lastly, silo-clients average them as gradients of this round as $\nabla \mathcal{L}_i = (1/|\mathbb{D}_i|) \sum_{D_j \in \mathbb{D}_i} \nabla l_j = (1/|\mathbb{D}_i|) \sum_{D_j \in \mathbb{D}_i} \nabla \sum_q y_q \log \hat{y}_q$, for local optimization on GNN parameters $\theta_i^{(n+1)}$.

Federated optimization. When silo-clients finish their local optimization, the server will leverage FedAvg to average local models as the new global model as $\theta^{(n+1)} = (1/|\mathbb{C}|) \sum_{C_i \in \mathbb{C}} \theta_i^{(n+1)}$. Updated parameters are the initial ones of the next round of optimization. Federated optimization is recurrently executed until the performance of each silo-client tends to converge.

5 ANALYSIS

5.1 Privacy Analysis

Subgraph-level privacy. In our solution, we preserve privacy by developing a hierarchical structure and a secure learning method. Specifically, for storage structure, edges are preserved between two device-clients corresponding to the two nodes instead of within the silo-clients, restraining features' exposure to other silo-clients. Besides, to protect against privacy leakage during message passing, the neighbor-agnostic strategy ensures that the target device-client only gets the encoded embedding and sends the aggregated one to its silo-client for decoding, where the silo-client cannot know the raw embedding of neighbors. In this way, 0% subgraph-level privacy is leaked when executing SecMP according to the quantifying metric defined in Appendix A.1.

Node-level privacy. We analyze the privacy leakage of node embeddings h of device-clients. Based on Equation 4, we select $\alpha_i \in \alpha_1, \dots, \alpha_{T+1}$ to encode h as $\tilde{h} = g(\alpha_i) = (h, z_2, \dots, z_{T+1}) \cdot U_i$, where $g(\beta_i) = z_i$ and $U \in \mathbb{F}^{(T+1) \times (T+1)}$ is the encoding matrix defined as

$$U_{i,j} = \prod_{k \in [T+1] \setminus \{i\}} \frac{\alpha_j - \beta_k}{\beta_i - \beta_k}. \quad (5)$$

Any neighbor device-client will receive an encoded embedding $\tilde{h} = hU_t^{\text{top}} + ZU_t^{\text{bottom}}$, where $t \in [T]$, $Z = (z_2, \dots, z_{T+1})$, and $U^{\text{top}} \in \mathbb{F}^T$, $U^{\text{bottom}} \in \mathbb{F}^{T \times T}$ are the t -th columns top and bottom submatrices in U .

LEMMA 5.1. *The $T \times T$ matrix U^{bottom} is invertible.*

Please refer Appendix A.3 for Lemma 5.1. Since U^{bottom} is invertible, we can mask encoded data hU_t^{top} as Z is uniformly randomized, and h cannot be directly decoded for any $T \geq 1$, i.e., U_t^{bottom} exists.

5.2 Complexity Analysis

Here we analyze the communication, encoding and decoding, and space complexity of HiFGL. We denote ξ as the size of model parameters, d as the dimension of coded or non-coded embeddings h or \tilde{h} , γ as the dimension of output prediction.

Communication complexity. We analyze three types of communication complexity guarantees of HiFGL. The communication complexity is $\mathcal{O}(dT)$ between a pair of connected device-clients. For a silo-client with its device-clients, the communication complexity for message passing and decoding is $\mathcal{O}(2|\mathbb{D}|\xi)$ and $\mathcal{O}(d|V_i| + \sum_{v_j \in V_i} dT|N_j|)$, respectively, where $|\mathbb{D}|$ is the number of the silo-client's device-clients. Last, the communication complexity between

Table 2: Overall information of datasets

Datasets Split Method	Cora Random	CiteSeer Random	PubMed Random
# of Silo-clients	5	5	5
# of Node	542	665	3943
# of Intra-client Edges	431	183	1772
# of Cross-client Edges	4199	3637	35461
# of Classes	7	6	3
Partition	6/2/2	6/2/2	6/2/2

silo-clients and the server is $\mathcal{O}(2\xi)$ for each silo-client and $\mathcal{O}(2|\mathbb{C}|\xi)$ for the server. More details are provided in Appendix A.4.1. **Encoding and decoding complexity.** The encoding and decoding complexity and be approximately guaranteed both as $\mathcal{O}(dT)$ according to analysis in Appendix A.4.2. **Space complexity.** As analyzed in Appendix A.4.3, the space complexity of HiFGL is $\mathcal{O}(|\mathbb{G}| \cdot (\xi + 3T + 2) + |\mathbb{E}| + |\mathbb{H}|)$ and we guarantee the increased complexity $\delta_{\mathcal{S}}$ satisfying $\mathcal{O}(|\mathbb{G}|\xi) \leq \mathcal{O}(\Delta_{\mathcal{S}}) \leq \mathcal{O}(|\mathbb{G}|\xi + |\mathbb{E}| \setminus (\mathbb{E}_1 \cup \mathbb{E}_2 \cup \dots))$.

6 EXPERIMENTS

6.1 Experimental Setup

Here we conduct experiments for evaluation on HiFGL. We first introduce experimental setups including datasets, baselines, metrics, and implementation details.

Datasets. We leverage popular graph datasets for node classification tasks, including Cora [28], CiteSeer [28], and PubMed [28]. Following previous FGL benchmarks [32, 35], we split the above graph datasets randomly to 5 subgraphs with comparable node numbers. Details of datasets are presented in Table 2.

Baselines. For framework-level experiments, we compare HiFGL with five baseline frameworks (Local, FedAvg [22], FedProx [15], FedPer [1], and Global) incorporated with two popular GNNs variants (GCN [13] and GraphSage [7]) as backbones. Besides, we also test performance of state-of-the-art FGL methods, including FedPerGNN [33], FedSage+ [40], and FED-PUB [2]. More information on baseline models is listed in Appendix A.5.

Metrics. We evaluate node classification performance by accuracy (ACC), i.e., the global percentage of accurately predicted samples, rather than the average of silo-client accuracy. In particular, we design a new metric, i.e., *Graph Information Gain*, to show how much graph information has been modeled as $\text{Gain}(\star\text{-}\mathcal{G}) = \frac{\text{ACC}(\star\text{-}\mathcal{G}) - \text{ACC}(\text{Local-MLP})}{\text{ACC}(\text{Global-}\mathcal{G}) - \text{ACC}(\text{Local-MLP})}$, where $\star\text{-}\mathcal{G}$ denotes a GNN backbone model under a framework.

Implementation details. We set the hidden dimension for GNNs and multi-layer perceptron as 64, and the input and output dimensions depend on the raw feature dimensions and the number of classes for each dataset. We implement all GNNs with 2 layers and train them for maximum 50 epochs or two hours with Adam [12] optimizer where the learning rate is set as 0.01 and multiplies gamma 0.9 for every 4 epoch. The T in embedding privatization is set as 1. The HiFGL framework is implemented based on PyTorch, and PyTorch-Lightning runs on the machine with Intel Xeon Gold 6148 @ 2.40GHz, V100 GPU and 64G memory. Our codes are open-sourced at <https://github.com/usail-hkust/HiFGL>.

Table 3: The prediction ACC and graph information gain of different FGL frameworks.

ACC	Cora			CiteSeer			PubMed		
	Mean	Std	Gain	Mean	Std	Gain	Mean	Std	Gain
Local-MLP	0.5698	±0.0071	+0%	0.6450	±0.0061	+0%	0.8051	+0.0006	+0%
Local-GCN	0.8095	±0.0149	+80.14%	0.7429	±0.0135	+75.77%	0.8525	±0.0073	+89.10%
FedAvg-GCN	0.8358	±0.0135	+88.93%	0.7601	±0.0152	+89.09%	0.8603	±0.0095	+103.76%
HiFGL-GCN	0.8555	±0.0162	+95.52%	0.7724	±0.0108	+98.61%	0.8626	±0.0064	+108.08%
Global-GCN	0.8689	±0.0182	+100%	0.7742	±0.0115	+100%	0.8583	±0.0033	+100%
Local-GraphSage	0.6207	±0.0103	+17.03%	0.6125	±0.0077	-25.04%	0.8221	±0.0101	+29.72%
FedAvg-GraphSage	0.8095	±0.0123	+80.22%	0.7656	±0.0139	+92.91%	0.8444	±0.0098	+68.71%
HiFGL-GraphSage	0.8642	±0.0288	+98.53%	0.7791	±0.0112	+103.31%	0.8504	±0.0169	+79.20%
Global-GraphSage	0.8686	±0.0215	+100%	0.7748	±0.0127	+100%	0.8623	±0.0058	+100%

6.2 Overall Prediction Performance

In this subsection, we show the results of the node classification task including HiFGL and baseline frameworks and algorithms. Specifically, we first compare HiFGL with three different frameworks to demonstrate the increased predictive ability brought by information integrity. Second, we conduct experiments to illustrate that for FGL, information integrity is more important than any other algorithm-level improvement.

6.2.1 Framework-level results. We investigate predictive knowledge retrieved under different frameworks measured by Graph Information Gain. First, we define the predictability lower bound and upper bound as 1) *Lower bound (0%)*: the performance of *Local-MLP*, which only separately learns the knowledge from raw features through a 2-layer multi-layer perceptron; 2) *Upper bound (100%)*: the performance of trained 2-layer backbone GNN (GCN and GraphSage) on the *Global* setting, which extracts both knowledge from raw features and graph information. Then, we consider the gap between the lower and upper bound as graph information knowledge. Therefore, we test GNNs under different frameworks and collect the mean and standard deviation of ACC for five times experiments and graph information gain in Table 3. Results show that under HiFGL, GCN and GraphSage have more graph information gain over than Local and FedAvg and comparable with under Global. Specifically, on three datasets, HiFGL-GCN outperforms 6.59%, 9.52%, and 4.32% than FedAvg-GCN, and HiFGL-GraphSage outperforms 18.31%, 10.40%, and 10.49% than FedAvg-GraphSage. The model improvement demonstrates that besides ensuring privacy and complexity, HiFGL enhances information integrity to achieve better accuracy approximately equal ones without FL settings because the graph information gain of HiFGL is obtained from preserved cross-client edges, which offers rich relational knowledge to learn more effective embeddings.

6.2.2 Algorithm-level results. The compared results among different FGL methods are depicted in Table 4. We compare HiFGL with three optimization schemes, *i.e.*, FedAvg, FedProx, and FedPer, with two GNNs, *i.e.*, GCN and GraphSage. Besides, we involve FedPerGNN, FedSage+, and FED-PUB, which design tailored advanced graph learning modules for FGL. We observe that under HiFGL, GCN, and GraphSage defeat all tested methods. Specifically, with the integrated information preserved by the HiFGL, GCN achieves

Table 4: The prediction ACC of different FGL algorithms.

	Cora	CiteSeer	PubMed
FedAvg-GCN	0.8358	0.7601	0.8603
FedProx-GCN	0.8238	0.7612	0.8305
FedPer-GCN	0.8202	0.7403	0.8471
FedAvg-GraphSage	0.8295	0.7540	0.8524
FedProx-GraphSage	0.8256	0.7593	0.8391
FedPer-GraphSage	0.8174	0.7625	0.8439
FedPerGNN	0.8351	0.7488	0.8346
FedSage+	0.7767	0.7567	0.8394
FED-PUB	0.8370	0.7579	0.8457
HiFGL-GCN	0.8555	0.7724	0.8626
HiFGL-GraphSage	0.8642	0.7791	0.8504

an accuracy of 0.8555, over 2.3%, 3.8%, and 4.3% improvement than FedAvg, FedProx, and FedPer on Cora dataset, respectively. Similarly, GraphSage gets 4.1%, 4.6%, and 5.7% improvement. Significant performance promotion is attained on CiteSeer and PubMed datasets. The reason is that for GCN and GraphSage that highly depend on structure information, graph integrity can offer great benefits. In addition, against methods with more advanced GNNs, HiFGL wins on prediction accuracy only by incorporating simple backbones. For example, under HiFGL, GraphSage surpasses FedPerGNN, FedSage+, and FED-PUB by 4.0%, 3.0%, and 2.8%, respectively, on CiteSeer. These experimental results demonstrate that graph integrity can bring much effectiveness improvement than GNN modification including high-order relationships incorporated in FedPerGNN, missing neighbor generation in FedSage+, and community-aware personalization in FED-PUB.

6.3 Efficiency Results

We study the efficiency by evaluating time and memory cost through varying two GNN hyper-parameters, *i.e.*, layer numbers and hidden dimensions, and take HiFGL-GCN as the testing model.

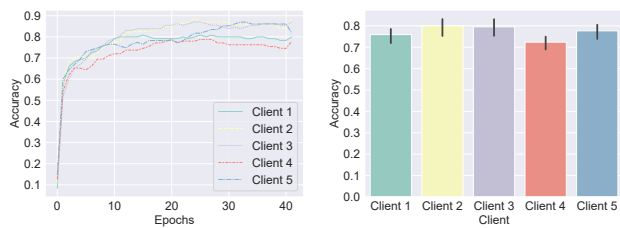
GNN layers. First, we set GNN ranging from 1 to 16 layers, with a hidden dimension of 64, on the CiteSeer dataset. We display results in Table 5 that the highest ACC occurs in 2-layer GCN, and the performance decreases for more and fewer layers. The training time

Table 5: Training results of different GNN layers.

# of GNN Layers	1	2	4	8	16
ACC (%)	73.04	77.91	70.87	39.84	26.02
Epoch Time (s)	6.5	4.3	5.6	8.4	16.4
Memory (GB)	3.41	3.45	3.49	3.59	3.80

Table 6: Training results of different hidden dimensions.

Hidden Dimensions	4	8	16	32	64	128
ACC (%)	83.45	85.15	85.33	85.27	85.88	86.39
Epoch Time (s)	17.2	18.05	20.81	19.84	23.49	44.30
Memory (GB)						3.753 ± 0.005



(a) Training ACC of silo-client models. (b) Average ACC of silo-client models.

Figure 4: Local model performance on Cora dataset.

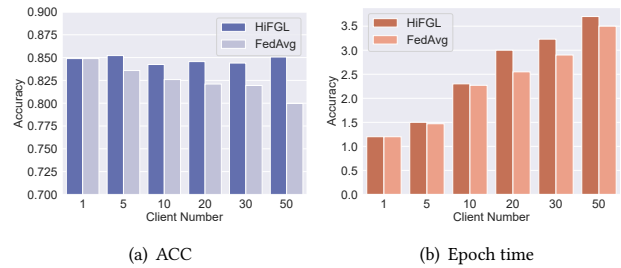
per epoch and the memory both increase along with the growth of the layer number. The results show that 2-layer GCN in the HiFGL framework is the most efficient model for ACC and training time, and it does not need expensive memory for training.

GNN dimension. We also set different hidden dimensions of a 2-layer GCN, ranging from 4 to 128, on the PubMed dataset. Table 6 demonstrates that performance increases with larger hidden dimensions since a more complicated model can catch more intricate patterns while obstructing high efficiency. Besides, the memory for different hidden dimensions is only slightly different because parameters are not dominant for memory costs compared with the other parts of the HiFGL framework under our implementation.

In conclusion, higher performance is not always brought by expensive executive costs that multiple layers of GNNs need a heavy training burden but are predicted poorly. Increasing hidden dimensions of GNNs helps the ACC, but the time consumption grows exponentially. Therefore, we select 2-layer GNNs with the hidden dimension of 64 for our main experiments. Our framework has sacrificed efficiency to achieve more advanced privacy preservation, especially during training stages equipping with SecMP. The imperfection of inefficient processes remains a future development such as faster secret sharing techniques. Fortunately, this work is still an applicable solution for cross-silo cross-device FGL scenarios with a high demand for privacy, scalability, and generalizability.

6.4 Local Prediction Performance

Beyond the global perspective, we also investigate the local performance of each silo-client. Specifically, we collect the ACC of local models in all training epochs and show them in Figure 4, respectively. We discover that local models do not simultaneously

**Figure 5: Sensitivity for different silo-client numbers under the HiFGL and FedAvg frameworks.**

upgrade or degrade during training in Figure 4(a). Instead, the five curves fluctuate in inconsistent upward patterns, which means a tradeoff is made among them during co-optimization directed by the server for global ACC improvement. Moreover, the converged performance of silo-clients is slightly diverse for their data distribution difference in FL. In our example, the distributions of node features and graph structures in silo-clients are diverse naturally, which causes local models to be optimized in different directions during global loss degradation.

6.5 Sensitivity of Silo-client Number

We evaluate the sensitivity of the silo-client number under different frameworks. Specifically, we vary it from 1 to 50 and take GraphSage as the backbone model to find out the variation of ACC in Figure 5(a). Moreover, we also present executive time costs for different silo-client numbers in Figure 5(b). The number hardly influences model performance under HiFGL, while the ACC decreases as the number increases, and models of HiFGL are trained slightly longer than FedAvg. Because for FedAvg, a large number of silo-clients means that fewer time-consuming cross-client message passing operations are taken, and fragmentary information remaining on each silo-client is inadequate for precise training. These results further prove the importance of graph integrity for an FGL framework.

7 CONCLUSION

This paper first studies cross-silo cross-device FGL, where we propose a HiFGL framework based on a novel hierarchical architecture on heterogeneous clients. Moreover, we devise a tailored graph learning algorithm, SecMP, for multi-level privacy-preserving optimization with graph integrity. Theoretical analyses are provided for privacy and efficiency guarantees. Extensive experiments demonstrate the prediction improvement gained from graph integrity on three datasets. HiFGL offers versatility in a wider range of real-world applications, even in solely cross-silo or cross-device settings.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (Grant No.62102110, No.92370204), National Key R&D Program of China (Grant No.2023YFF0725004), Guangzhou-HKUST(GZ) Joint Funding Program (Grant No. 2022A03J00056, No.2023A03J0008), Education Bureau of Guangzhou Municipality.

REFERENCES

- [1] Manoj Ghuhana Arivazhagan, Vinay Aggarwal, Aaditya Kumar Singh, and Sunav Choudhary. 2019. Federated learning with personalization layers. *arXiv preprint arXiv:1912.00818* (2019).
- [2] Jinheon Baek, Wonyong Jeong, Jiongdoo Jin, Jaehong Yoon, and Sung Ju Hwang. 2023. Personalized subgraph federated learning. In *International Conference on Machine Learning*. PMLR, 1396–1415.
- [3] E Berlekamp. 2006. Nonbinary BCH decoding (Abstr.). *IEEE Transactions on Information Theory* 14, 2 (2006), 242–242.
- [4] Fahao Chen, Peng Li, Toshiaki Miyazaki, and Celimuge Wu. 2021. Fedgraph: Federated graph learning with intelligent sampling. *IEEE Transactions on Parallel and Distributed Systems* 33, 8 (2021), 1775–1786.
- [5] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.
- [6] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International conference on machine learning*. PMLR, 1263–1272.
- [7] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).
- [8] Bing He, Dian Zhang, Siyuan Liu, Hao Liu, Dawei Han, and Lionel M Ni. 2018. Profiling driver behavior for personalized insurance pricing and maximal profit. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 1387–1396.
- [9] Chao Huang, Jianwei Huang, and Xin Liu. 2022. Cross-silo federated learning: Challenges and opportunities. *arXiv preprint arXiv:2206.12949* (2022).
- [10] Wei Jin, Yao Ma, Xiaorui Liu, Xianfeng Tang, Suhang Wang, and Jiliang Tang. 2020. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 66–74.
- [11] Kiran S Kedlaya and Christopher Umans. 2011. Fast polynomial factorization and modular composition. *SIAM J. Comput.* 40, 6 (2011), 1767–1802.
- [12] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
- [13] Thomas N Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.
- [14] Runze Lei, Pinghui Wang, Junzhou Zhao, Lin Lan, Jing Tao, Chao Deng, Junlan Feng, Xidian Wang, and Xiaohong Guan. 2023. Federated Learning Over Coupled Graphs. *IEEE Transactions on Parallel and Distributed Systems* 34, 4 (2023), 1159–1172.
- [15] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems* 2 (2020), 429–450.
- [16] Tzu-Heng Lin, Chen Gao, and Yong Li. 2019. Cross: Cross-platform recommendation for social e-commerce. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 515–524.
- [17] Fan Liu, Hao Liu, and Wenzhao Jiang. 2022. Practical adversarial attacks on spatiotemporal traffic forecasting models. *Advances in Neural Information Processing Systems* 35 (2022), 19035–19047.
- [18] Rui Liu, Pengwei Xing, Zichao Deng, Anran Li, Cuntai Guan, and Han Yu. 2024. Federated Graph Neural Networks: Overview, Techniques, and Challenges. *IEEE Transactions on Neural Networks and Learning Systems* (2024).
- [19] Tao Liu, Peng Li, and Yu Gu. 2021. Glint: Decentralized federated graph learning with traffic throttling and flow scheduling. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.
- [20] Zhiwei Liu, Liangwei Yang, Ziwei Fan, Hao Peng, and Philip S Yu. 2022. Federated social recommendation with graph neural network. *ACM Transactions on Intelligent Systems and Technology (TIST)* 13, 4 (2022), 1–24.
- [21] Hui Luo, Jingbo Zhou, Zhifeng Bao, Shuangli Li, J Shane Culpepper, Haochao Ying, Hao Liu, and Hui Xiong. 2020. Spatial object recommendation with hints: When spatial granularity matters. In *Proceedings of the 43rd International ACM SIGIR Conference on research and development in information retrieval*. 781–790.
- [22] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguerre y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.
- [23] Chuizheng Meng, Sirisha Rambhatla, and Yan Liu. 2021. Cross-Node Federated Graph Neural Network for Spatio-Temporal Data Modeling. *knowledge discovery and data mining* (2021).
- [24] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th annual international conference on mobile systems, applications, and services*. 94–108.
- [25] Qiying Pan and Yifei Zhu. 2022. FedWalk: Communication Efficient Federated Unsupervised Node Embedding with Differential Privacy. *knowledge discovery and data mining* (2022).
- [26] Qiying Pan, Yifei Zhu, and Lingyang Chu. 2023. Lumos: Heterogeneity-aware Federated Graph Learning over Decentralized Devices. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 1914–1926.
- [27] Liang Qu, Ningzhi Tang, Ruiqi Zheng, Quoc Viet Hung Nguyen, Zi Huang, Yuhui Shi, and Hongzhi Yin. 2023. Semi-decentralized federated ego graph learning for recommendation. In *Proceedings of the ACM Web Conference 2023*. 339–348.
- [28] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI magazine* 29, 3 (2008), 93–93.
- [29] Toyotaro Suzumura, Yi Zhou, Natahalie Baracaldo, Guangnan Ye, Keith Houck, Ryo Kawahara, Ali Anwar, Lucia Larise Stavarache, Yuji Watanabe, Pablo Loyola, Daniel Klyashtorny, Heiko Ludwig, and Kumar Bhaskaran. 2019. Towards Federated Graph Learning for Collaborative Financial Crimes Detection. *Research Papers in Economics* (2019).
- [30] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.
- [31] Binghui Wang, Ang Li, Meng Pang, Hai Li, and Yiran Chen. 2022. Graphfl: A federated learning framework for semi-supervised node classification on graphs. In *2022 IEEE International Conference on Data Mining (ICDM)*. IEEE, 498–507.
- [32] Zhen Wang, Weirui Kuang, Yuexiang Xie, Liuyi Yao, Yaliang Li, Bolin Ding, and Jingren Zhou. 2022. Federatedscope-gnn: Towards a unified, comprehensive and efficient package for federated graph learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4110–4120.
- [33] Chuhan Wu, Fangzhao Wu, Lingjuan Lyu, Tao Qi, Yongfeng Huang, and Xing Xie. 2022. A federated graph neural network framework for privacy-preserving personalization. *Nature Communications* 13, 1 (2022), 1–10.
- [34] Yuexiang Xie, Zhen Wang, Dawei Gao, Daoyuan Chen, Liuyi Yao, Weirui Kuang, Yaliang Li, Bolin Ding, and Jingren Zhou. 2023. FederatedScope: A Flexible Federated Learning Platform for Heterogeneity. *Proceedings of the VLDB Endowment* 16, 5 (2023), 1059–1072.
- [35] Yuexiang Xie, Zhen Wang, Dawei Gao, Daoyuan Chen, Liuyi Yao, Weirui Kuang, Yaliang Li, Bolin Ding, and Jingren Zhou. 2023. FederatedScope: A Flexible Federated Learning Platform for Heterogeneity. *Proceedings of the VLDB Endowment* 16, 5 (2023), 1059–1072.
- [36] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated Machine Learning: Concept and Applications. *ACM Transactions on Intelligent Systems and Technology* (2019).
- [37] Qian Yu, Songze Li, Netanel Raviv, Seyed Mohammadreza Mousavi Kalan, Mahdi Soltanolkotabi, and Salman A Avestimehr. 2019. Lagrange coded computing: Optimal design for resiliency, security, and privacy. In *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, 1215–1225.
- [38] Xiaoming Yuan, Jiahui Chen, Jiayu Yang, Ning Zhang, Tingting Yang, Tao Han, and Amir Taherkordi. 2022. Fedstn: Graph representation driven federated learning for edge computing enabled urban traffic flow prediction. *IEEE Transactions on Intelligent Transportation Systems* (2022).
- [39] Binchi Zhang, Minnan Luo, Shangbin Feng, Ziqi Liu, Jun Zhou, and Qinghua Zheng. 2021. Ppsgc: A privacy-preserving subgraph sampling based distributed gcn training method. *arXiv preprint arXiv:2110.12906* (2021).
- [40] Ke Zhang, Carl Yang, Xiaoxiao Li, Lichao Sun, and Siu Ming Yiu. 2021. Subgraph federated learning with missing neighbor generation. *Advances in Neural Information Processing Systems* 34 (2021), 6671–6682.
- [41] Weijia Zhang, Hao Liu, Lijun Zha, Hengshu Zhu, Ji Liu, Dejing Dou, and Hui Xiong. 2021. MugRep: A multi-task hierarchical graph representation learning framework for real estate appraisal. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3937–3947.
- [42] Xiaojin Zhang, Hanlin Gu, Lixin Fan, Kai Chen, and Qiang Yang. 2022. No free lunch theorem for security and utility in federated learning. *ACM Transactions on Intelligent Systems and Technology* 14, 1 (2022), 1–35.
- [43] Xiaojin Zhang, Yan Kang, Kai Chen, Lixin Fan, and Qiang Yang. 2023. Trading Off Privacy, Utility, and Efficiency in Federated Learning. *ACM Transactions on Intelligent Systems and Technology* 14, 6 (2023), 1–32.
- [44] Ligeng Zhu, Zhijian Liu, and Song Han. 2019. Deep leakage from gradients. *Advances in neural information processing systems* 32 (2019).

A APPENDIX

A.1 Quantifying Subgraph-level Privacy Leakage

We analyze the subgraph-level privacy leakage of previous cross-client FGL frameworks. Subgraph-level privacy leakage derives from the information sharing across cross-client edges, which lets silo-clients see nodes from others. There are works such as FedSage [40] and FedPUB [2] that drop cross-client edges to sacrifice

Algorithm 1: Training algorithm for GCN based on SecMP.

Input: The graph \mathbb{G} , the server S , $|\mathbb{G}|$ silo-clients \mathbb{C} (each C_i with a local model \mathcal{G}_i), n_d device-clients \mathbb{D} (each D_j with feature $h^{(0)}$ and neighbors \mathcal{N}_{D_j}).

Output: The converged global model parameterized with $\theta^{(n)}$ trained after n epochs $\mathcal{G}_{\theta^{(n)}}$.

- 1 Initialize each silo-client's model parameters with $\theta^{(0)}$ and encoding-decoding parameters
 $\mu = \{\alpha_1, \dots, \alpha_{T+1}, \beta_1, \dots, \beta_{T+1}, z_2, \dots, z_{T+1}\};$
- 2 **while** $\mathcal{G}_{\theta^{(n)}}$ is not converged **do**
- 3 **foreach** $C_i \in \mathbb{C}$ in parallel **do**
- 4 **foreach** $D_j \in \mathbb{D}_i$ in parallel **do**
- 5 $\hat{h}^{(0)} \leftarrow \frac{1}{\sqrt{|\mathcal{N}_j|}} h^{(0)};$
- 6 **foreach** $D_p \in \mathcal{N}_j$ **do**
- 7 Get μ_p from D_p 's silo-client;
- 8 $\tilde{h}^{(0)} \leftarrow \mathcal{F}(\hat{h}^{(0)}; \mu_p);$
- 9 Send coded embedding $\tilde{h}^{(0)}$ to D_p ;
- 10 **end**
- 11 /* After D_j receives all coded embeddings $\tilde{\mathcal{H}}_{\mathcal{N}_j}$ from \mathcal{N}_j */
- 12 $\tilde{h}^{(k)} \leftarrow \sum_{\tilde{h}_p^{(0)} \in \tilde{\mathcal{H}}_{\mathcal{N}_j}} \frac{1}{\sqrt{|\mathcal{N}_j|}} \mathcal{G}_i(\tilde{h}_p^{(0)}; \theta_i);$
- 13 Send $\tilde{h}^{(k)}$ to C_i , and C_i decodes it according to \mathcal{F} and μ_i as $h^{(k)}$ and send it back to D_j ;
- 14 $h^{(k)} \leftarrow h^{(k)}_{\mathcal{N}_j};$
- 15 Calculate estimated probability as $\hat{y} \leftarrow \text{SoftMax}(h^{(k)});$
- 16 Compute loss l_j of prediction \hat{y} and labels y ;
- 17 Send gradient ∇l_j to C_i ;
- 18 **end**
- 19 Optimize $\theta_i^{(n+1)}$ with $\nabla \mathcal{L}_i \leftarrow \frac{1}{|\mathbb{D}_i|} \sum_{D_j \in \mathbb{D}_i} \nabla l_j;$
- 20 Update global parameters as $\theta^{(n+1)} \leftarrow \frac{1}{|\mathbb{C}|} \sum_{C_i \in \mathbb{C}} \theta_i^{(n+1)};$
- 21 **end**

the information integrity to perfectly preserve subgraph-level privacy. However, other works such as Glint [19] choose to permit silo-clients to access the features of the nodes from other silo-clients connected with their own nodes. Here the subgraph-level privacy leakage of i -th silo-client ϵ_p^i can be defined as

$$\epsilon_p^i = \{|\mathbb{D}_j| \mid \mathbb{D}_j \in C_i \text{ and } \mathbb{D}' \notin C_i, \exists \mathbb{D}' \in \mathcal{N}_j\}, \quad (6)$$

where we utilize the number of nodes that possess cross-client edges (*i.e.*, neighbors out of their own silo-clients). Subsequently, we compute the average of ϵ_p^i among silo-clients as the global privacy leakage by $\epsilon_p = \frac{1}{|\mathbb{C}|} \sum_{i=1}^{|\mathbb{C}|} \epsilon_p^i$.

Under this formulation, on experimental datasets introduced in Section 6.1, we observe that privacy leakage is remarkably significant, which is 95.16% on Cora, 88.70% on CiteSeer, and 90.31% on PubMed for 5 silo-clients. Otherwise, 90.69%, 95.21%, and 95.24% edges are lost on Cora, CiteSeer, and PubMed for 5 silo-clients, respectively, if we protect privacy by dropping all cross-client edges.

Therefore, preventing nodes from being seen by other silo-clients is necessary for FGL frameworks.

A.2 Training Pipeline

As an example, here we appoint GCN as the backbone model and FedAvg as the federated optimization method to explain the workflow in Algorithm 1. Specifically, after initialization of model parameters with $\theta^{(0)}$ and encoding-decoding parameters μ , the optimization round will be executed recurrently. Training will be stopped after global convergence.

A.3 Proof of Lemma 5.1

PROOF. Before we give proof, we revisit the selected $2T$ distinct elements which satisfying $\{\alpha_1, \dots, \alpha_{T+1}\} \cap \{\beta_1, \dots, \beta_{T+1}\} = \emptyset$. Thus $\alpha_i - \alpha_j$, $\beta_i - \beta_j$, and $\alpha_i - \beta_j$ are non-zero for any different i and j . Here we start to validate U^{bottom} 's invertibility. First, we multiply every i -th row by $\prod_{k \in [T+1] \setminus \{i\}} (\beta_i - \beta_k) \neq 0$ to obtain

$$U_{i,j}^{\text{bottom}} = \prod_{k \in [T+1] \setminus \{i\}} (\alpha_j - \beta_k). \quad (7)$$

Second, we multiply every j -th column by $\prod_{k \in [T+1]} \frac{1}{\alpha_j - \beta_k} \neq 0$ as

$$U_{i,j}^{\text{bottom}} = \frac{1}{\alpha_j - \beta_i}, \quad (8)$$

Third, we subtract the n -th column from the first $n-1$ columns and extract the common factor as

$$\begin{aligned} U_{n \times n}^{\text{bottom}} &= \begin{bmatrix} \frac{b_1 - b_n}{(a_1 - b_1)(a_1 - b_n)} & \frac{b_2 - b_n}{(a_1 - b_2)(a_1 - b_n)} & \dots & \frac{1}{a_1 - b_n} \\ \frac{b_1 - b_n}{(a_2 - b_1)(a_2 - b_n)} & \frac{b_2 - b_n}{(a_2 - b_2)(a_2 - b_n)} & \dots & \frac{1}{a_2 - b_n} \\ \frac{b_1 - b_n}{(a_n - b_1)(a_n - b_n)} & \frac{b_2 - b_n}{(a_n - b_2)(a_n - b_n)} & \dots & \frac{1}{a_n - b_n} \end{bmatrix} \\ &= \prod_{i=1}^{n-1} (b_i - b_n) \begin{bmatrix} \frac{1}{a_1 - b_1} & \frac{1}{a_1 - b_2} & \dots & 1 \\ \frac{1}{a_2 - b_1} & \frac{1}{a_2 - b_2} & \dots & 1 \\ \vdots & \vdots & \dots & \vdots \\ \frac{1}{a_n - b_1} & \frac{1}{a_n - b_2} & \dots & 1 \end{bmatrix} \end{aligned} \quad (9)$$

Obviously, the values of the n -th column are the same, thus we can subtract the last row from the first $n-1$ rows and extract the common factor as

$$\begin{aligned} U_{n \times n}^{\text{bottom}} &= \frac{\prod_{i=1}^{n-1} (b_i - b_n)}{\prod_{j=1}^n (a_j - b_n)} \begin{bmatrix} \frac{a_n - a_1}{(a_1 - b_1)(a_n - b_1)} & \frac{a_n - a_1}{(a_1 - b_2)(a_n - b_2)} & \dots & 0 \\ \frac{a_n - a_2}{(a_2 - b_1)(a_n - b_1)} & \frac{a_n - a_2}{(a_2 - b_2)(a_n - b_2)} & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ \frac{1}{a_n - b_1} & \frac{1}{a_n - b_2} & \dots & 1 \end{bmatrix} \\ &= \frac{\prod_{i=1}^{n-1} (a_n - a_i)(b_i - b_n)}{\prod_{j=1}^n (a_j - b_n) \prod_{k=1}^{n-1} (a_n - b_k)} \begin{bmatrix} \frac{1}{a_1 - b_1} & \frac{1}{a_1 - b_2} & \dots & \frac{1}{a_1 - b_{n-1}} \\ \frac{1}{a_2 - b_1} & \frac{1}{a_2 - b_2} & \dots & \frac{1}{a_2 - b_{n-1}} \\ \vdots & \vdots & \dots & \vdots \\ \frac{1}{a_{n-1} - b_1} & \frac{1}{a_{n-1} - b_2} & \dots & \frac{1}{a_{n-1} - b_{n-1}} \end{bmatrix} \end{aligned} \quad (10)$$

where we can obtain a recursion formula that

$$U_{n \times n}^{\text{bottom}} = \frac{\prod_{i=1}^{n-1} (a_n - a_i)(b_i - b_n)}{\prod_{j=1}^n (a_j - b_n) \prod_{k=1}^{n-1} (a_n - b_k)} U_{(n-1) \times (n-1)}^{\text{bottom}}. \quad (11)$$

Last, by continuing the recursive process, we can derive the determinant as

$$\det U_{n \times n}^{\text{bottom}} = \frac{\prod_{1 \leq i < j \leq n} (a_j - a_i)(b_i - b_j)}{\prod_{i,j=1}^n (a_i - b_j)} \neq 0. \quad (12)$$

Hence, the matrix U^{bottom} is invertible. \square

A.4 Complexity Analysis Details

Here we analyze communication, encoding and decoding, and space complexity, which are based on the following conditions. The size of encoding-decoding parameters μ is $3T + 2$. Proofs are satisfied with 1) $T \leq 10$, 2) $\gamma \leq 10$, 3) $d \gg T$, and $d \gg \gamma$ based on realistic assumption and experimental datasets.

A.4.1 Communication Complexity. The communication is composed of three parts, including 1) *communication between device-clients*: coded embedding exchange between device-clients and device-clients, 2) *communication between device-clients and silo-clients*: aggregated embedding communication between device-clients and silo-clients, and 3) *communication between silo-clients and the server*: model parameters sharing between silo-clients and the server.

1) *Communication between device-clients.* The communication between any two connected device-clients (from a source device-client to a target device-client) consists of two parts. First, the target device-client passes the $3T + 2$ encoding parameters of its silo-client to the source device-client. Second, after coding by the source device-client, the coded embedding will be sent to the target device-client as K parts by $T + 1$ times at least. Therefore, for any directed edge, the minimum communication complexity between two device-clients can be computed as $O(3T+2+d(T+1)) \approx O(dT)$.

2) *Communication between device-clients and silo-clients.* Each device-client will download the latest model parameters and upload its gradients at each training round. Thus, the communication complexity between a single device-client and the corresponding silo-client can be measured by ξ , the size of the model parameters. In a word, the communication complexity for a device-client with its silo-client is $O(2\xi)$, and for a silo-client with its device-clients is $O(2|\mathbb{D}|\xi)$ that $|\mathbb{D}|$ is the number of the silo-client's device-clients. In the decoding stage, when device-client D_j transmits coded embeddings to the silo-client C_i for decoding, the complexity is $O(dT|N_j|)$ from D_j to C_i and $O(d)$ from C_i to D_j . Totally, the complexity of communication for C_i is $O(d|V_i| + \sum_{v_j \in V_i} dT|N_j|)$ at each round.

3) *Communication between silo-clients and the server.* The communication between silo-clients and the server obeys the traditional FL setting, which can be computed as $O(2\xi)$ for each silo-client and $O(2|\mathbb{C}|\xi)$ for the server.

A.4.2 Encoding and Decoding Complexity. We then present the analysis of Lagrange interpolation-based encoding and decoding complexity. Specifically, according to existing mathematical proof [11], the complexity of interpolating a k -degree polynomial can be evaluated as $O(k \log^2 k \log \log k)$. Therefore, we compute the encoding

complexity as $O(d(T+1) \log^2(T+1) \log \log(T+1)) \approx O(dT)$. Besides, using the proposed technique [3], we can decode the embedding with a complexity as $O(d(T+1) \log^2 T \log \log T) \approx O(dT)$.

A.4.3 Space Complexity. The space complexity \mathcal{S} of HiFGL on a graph $\mathbb{G} = (\mathbb{V}, \mathbb{E}, \mathbb{H})$, is the summation of silo-client's model complexity $O(|\mathbb{G}| \cdot (\xi + 3T + 2))$ and device-client's data (raw features and neighbor device-client pointers) complexity $O(|\mathbb{E}| + |\mathbb{H}|)$, which can be formulated as

$$\mathcal{S} = O(|\mathbb{G}| \cdot (\xi + 3T + 2) + |\mathbb{E}| + |\mathbb{H}|), \quad (13)$$

where $|\mathbb{G}|$ is the number of silo-clients (or subgraphs), $|\mathbb{E}|$ is the number of edges, and $|\mathbb{H}|$ is the raw feature complexity.

Conventionally, in FGL, researchers proposed two schemes for graph storage. 1) Trusted server for cross-client edges and silo-clients for intra-client edges. Its space complexity is $O(|\mathbb{G}| \cdot \xi + |\mathbb{E}| + |\mathbb{H}|)$. 2) Only silo-clients store intra-client edges. Its space complexity is $O(|\mathbb{G}| \cdot \xi + \sum_{\mathbb{G}_i \in \{\mathbb{G}\}} |\mathbb{E}_i| + |\mathbb{H}|)$. In this work, the HiFGL framework only needs tiny extra storage memory $\Delta_{\mathcal{S}}$ compared with two traditional schemes, which can be stated as

$$O(|\mathbb{G}|\xi) \leq O(\Delta_{\mathcal{S}}) \leq O(|\mathbb{G}|\xi + |\mathbb{E} \setminus (\mathbb{E}_1 \cup \mathbb{E}_2 \cup \dots)|), \quad (14)$$

where $|\mathbb{E} \setminus (\mathbb{E}_1 \cup \mathbb{E}_2 \cup \dots)| \leq |\mathbb{E}|$.

A.5 Baseline Information

We involve five baseline frameworks in our experiments, including

- *Local*: models are individually trained on silo-clients.
- *FedAvg* [22]: a collaborative learning framework that averages local model parameters for global optimization.
- *FedProx* [15]: a more robust federated optimization method based on FedAvg with regularization of parameters.
- *FedPer* [1]: a personalized FL algorithm allowing some layers to be free from FedAvg training for better local fitness.
- *Global*: a usual graph training way without distributed or private constraints.

Furthermore, we compare two popular GNNs as backbone models within federated frameworks, including

- *GCN* [13]: a spectral-based graph convolutional network form capturing first-order structure feature with node information for node representation learning.
- *GraphSage* [7]: a spatial-based graph model aggregating sampled neighbor information with ego features for inductive graph learning.

We also leverage 2-layer multi-layer perceptron (MLP) under *Local* frameworks for comparison with GNNs. Besides, we test classification performance on state-of-the-art FGL methods, including

- *FedPerGNN* [33]: a cross-device FL method on personalized federated item recommendation.
- *FedSage+* [40]: a federated inductive graph learning model with neighbor generation for missing edge reconstruction.
- *FED-PUB* [2]: a personalized federated subgraph learning method incorporating community structures and masked graph convolutional networks.