SplitGNN: Splitting GNN for Node Classification with Heterogeneous Attention

Xiaolong Xu*
Ant Group
Hangzhou, Zhejiang, China
yiyin.xxl@antgroup.com

Lingjuan Lyu†
Ant Group
Hangzhou, Zhejiang, China
ingjuanlvsmile@gmail.com

Yihong Dong†
Ant Group
Hangzhou, Zhejiang, China
dongyihongbill@gmail.com

Yicheng Lu
Ant Group
Hangzhou, Zhejiang, China
sier.lyc@antgroup.com

Weiqiang Wang
Ant Group
Hangzhou, Zhejiang, China
weiqiang.wwq@antgroup.com

Hong Jin
Ant Group
Hangzhou, Zhejiang, China
jinhong.jh@antgroup.com

ABSTRACT

With the frequent happening of privacy leakage and the enactment of privacy laws across different countries, data owners are reluctant to directly share their raw data and labels with any other party. In reality, a lot of these raw data are stored in the graph database, especially for finance. For collaboratively building graph neural networks (GNNs), federated learning (FL) may not be an ideal choice for the vertically partitioned setting where privacy and efficiency are the main concerns. Moreover, almost all the existing federated GNNs are mainly designed for homogeneous graphs, which simplify various types of relations as the same type, thus largely limits their performance. We bridge this gap by proposing a split learning-based GNN (SplitGNN), where this model is divided into two sub-models: the local GNN model includes all the private data related computation to generate local node embeddings, whereas the global model calculates global embeddings by aggregating all the participants' local embeddings. Our SplitGNN allows the isolated heterogeneous neighborhood to be collaboratively utilized. To better capture representations, we propose a novel Heterogeneous Attention (HAT) algorithm and use both node-based and path-based attention mechanisms to learn various types of nodes and edges with multi-hop relation features. We demonstrate the effectiveness of our SplitGNN on node classification tasks for two standard public datasets and the real-world dataset. Extensive experimental results validate that our proposed SplitGNN significantly outperforms the state-of-the-art (SOTA) methods.

ACM Reference Format:
Xiaolong Xu, Lingjuan Lyu, Yihong Dong, Yicheng Lu, Weiqiang Wang, and Hong Jin. 2022. SplitGNN: Splitting GNN for Node Classification with Heterogeneous Attention . In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

*Corresponding author
†This work is completed during the tenure of Ant Group

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Graph neural networks (GNNs) have attracted great interest in a variety of applications. GNNs can roughly be classified into two classes based on the graph structure: *heterogeneous graph* and *homogeneous graph*. Generally, the heterogeneous graph contains multiple node- and relation-types. On the contrary, in the homogeneous graph, nodes are objects of the same entity type and links are relationships from the same relation type. For the homogeneous graph, various effective methods have been investigated. For example, GraphSAGE [1] can derive the information of neighbors using an aggregation function. Graph convolutional network (GCN) [2] conducts the average pooling for each node's neighbors and then uses both linear projection and non-linear activation operations. Graph attention networks (GAT) [3] adopts an effective attention mechanism and achieves more powerful representation empirically. However, these methods cannot perform well in heterogeneous graphs, where abundant features are lying on edges (*e.g.* view frequency, watch duration, and publication year, *et al.*).

On the other hand, training accurate GNN models requires a wealth of high-quality graph-structured data, including rich node features and complete adjacent information. However, in practice, due to business competition and regulatory restrictions, such information could possibly be isolated by different participants, who are unwilling to share their information, plaguing many practical applications, such as fraud detection over banks and social network recommendation over platforms. Such data isolation problems present a serious challenge for the development of GNN models.

Federated learning (FL) and split learning (SL) are two promising distributed machine learning (ML) approaches that have gained attention due to their inherent privacy-aware capabilities that allow participants to collaboratively learn models without disclosing their raw data. Several recent works have attempted to build federated GNNs when data are horizontally partitioned [4, 5]. However, few works have studied the problem of GNN when data are vertically partitioned, which popularly exists in practice. In a vertically partitioned setting, both features and edges are distributed across different participants. For example, assume there are three participants ($\mathcal{A}$,

$\mathcal{B}$, and C) and they have four same nodes. The node features are vertically split, i.e., $\mathcal{A}$ has $f1$, $f2$, and $f3$, $\mathcal{B}$ has $f4$ and $f5$, and C has $f6$ and $f7$. Meanwhile, $\mathcal{A}$, $\mathcal{B}$, and C may have their own edges. For instance, $\mathcal{A}$ has social relation between nodes while $\mathcal{B}$ and C have the payment relation between nodes. We also assume $\mathcal{A}$ is the party that has the node labels. The problem becomes how to collaboratively build federated GNN models by using the graph data of A, B, and C. In this paper, we take the vertically partitioned setting for example and present how to collaboratively build GNN models by leveraging the privacy and efficiency advantage of split learning [6].

Unlike previous privacy-preserving machine learning models that assume only nodes are held by different parties but samples have no relationship, our task is more challenging because GNN relies on the samples' relationships, which are also kept by different participants. More recently, Zhou *et al*. [7] have attempted federated GNN models under the vertically setting, however, their work was tested on small-scale homogeneous graph datasets, limiting their practicability in large-scale industrial applications. Moreover, the graph topology was still exploited locally, the model performance may be substantially reduced when the dataset is largely decentralized. To date, a mature solution for federated GNN models under the vertically partitioned setting is still missing. To fill in this gap and facilitate modeling heterogeneous relational graphs in the vertically partitioned setting, in this paper, we propose a novel framework, i.e., SplitGNN, for node classification across multiple heterogeneous graphs by splitting the whole GNN across different participants.

In summary, the main contributions of this paper include [1]:

- We present a novel approach, named SplitGNN, which combines split learning with federated learning and eliminates their inherent drawbacks. To learn various types of nodes and edges in the heterogeneous graphs under the vertically partitioned setting, we design a novel base model named Heterogeneous Attention (HAT), which uses both node-based and path-based attention mechanisms by considering multi-hop relations.
- We evaluate the different interactive layer strategies for the server to operate local node embeddings from participants, then test the SplitGNN performance on different data distributions and analyze the communication effectiveness of our proposed method compared with federated learning.
- Extensive experiment results on the standard open-source datasets and the real-world dataset validate the superiority of our proposed SplitGNN on node classification task.

2 SPLITGNN

In this section, we first introduce our threat model, followed by an overview of our proposed SplitGNN. We then present how to generate local node embeddings on each participant, and how to derive global node embeddings on the server. Finally, we show how to ensure the privacy of local node embeddings, and propose a new heterogeneous attention algorithm to better capture the information in heterogeneous graphs. The workflow of SplitGNN is shown as Algorithm 1.

2.1 Threat Model

The security model can be generally categorized into two types, i.e., honest-but-curious (semi-honest) model and malicious model. Although the semi-honest setting is less harsh than the malicious setting, it is more practical and has better efficiency than the latter. Hence, we consider semi-honest adversaries. That is, participants and the server strictly follow the protocol, but they also use all intermediate computation results to infer as much information as possible. We also assume that the server does not collude with any party. This security setting is commonly considered in many existing works [8]. We remark that this is a reasonable assumption since the server can be played by authorities such as governments or replaced by a trusted execution environment (TEE) [9].

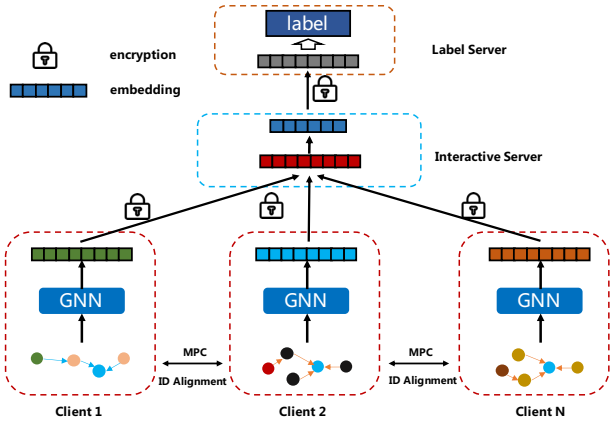

Figure 1: Workflow for SplitGNN.

2.2 Overview of SplitGNN

In our framework, participants are willing to cooperatively train a global model with the aid of a server storing a fraction of the model, but are reluctant to directly share raw data and labels, because not only the raw samples (e.g., chest X-ray images) but also their ground-truth labels (e.g., lung cancer diagnosis) are privacy-sensitive. In actual operation, the label server is the same as one of the participants who want to enhance the classification model's capabilities with data from other participants. In an NN model, each raw sample is fed into the input layer, and its ground-truth label is compared with the model's prediction for loss calculation at the output layer. Therefore, to preserve the privacy of each sample-and-label pair, both input and output layers should be stored by each participant, while the rest of the layers can be offloaded to the server, resulting in a tripartite SplitGNN. This is in stark contrast to the standard bipartite Split model where only the input layer is stored at each participant, while the remaining layers can be offloaded to the server.

In more detail, in our tripartite SplitGNN, the forward steps at each layer can be divided into three steps: it first calculates local embedding at each participant individually with private data. Then,

[1] 1. The data set does not contain any Personal Identifiable Information (PII) 2. The data set is desensitized and encrypted 3. Adequate data protection was carried out during the experiment to prevent the risk of data copy leakage, and the data set was destroyed after the experiment 4. The data set is only used for academic research, it does not represent any real business situation)

Algorithm 1 Heterogeneous Attention (HAT)

Input: the graph G = $(\mathcal{V}, \mathcal{E})$, Node features $\mathbf{f}_i$ and Edge features $\mathbf{e}_i$, the set of pre-defined metapaths and all original relations $\{\rho_1, \rho_2, ..., \rho_n\}$, Hops number N and multi-heads number M
Output: the result of node classification ($\hat{y}$), node embedding, relation embedding

while $n = 1, 2, ..., N$ **do**
  while ρ_i in $\{\rho_0, \rho_1, ..., \rho_n\}$ **do**
    Feature Transformation $\mathbf{f}_i^l$, $\mathbf{e}_i^l$;
    Feature Combination $h_i^{\rho,l}$ according to Eq. 1;
    while m=1,2,...,M **do**
      Computing node attention coefficients $\alpha_{i,j}^{\rho,l,m}$ adopting Eq. 1;
    end while
    Node embedding Fusion to get $\mathbf{z}_i^{\rho,l}$ as Eq. 3;
  end while
  while m=1,2,...,M **do**
    Calculating path attention coefficients $\beta_i^{\rho,l,m}$ based on Eq. 6.
  end while
  Getting $\mathbf{f}i^{l+1}$ as node embedding in this layer by using Eq. 5.
end while
return node embeddings.

the semi-honest server collects non-private local embeddings to compute global embedding. In the end, the server returns the final hidden layer to the party that has labels to compute prediction and loss. Participants and the server perform forward and back propagations to complete model training and prediction, during which the private data (i.e., features, edges, and labels) are always kept by participants themselves.

Compared with our SplitGNN, the other relevant work Zhou [7] exhibits the following weaknesses:

- The full dataset is used in each iteration, which incurs much more communication cost;
- Data holders send their local node embeddings in plaintext form to the server, which may incur privacy leakage;
- The server transmits the last hidden layer output in plaintext to the data holder who has the label, which also poses privacy issues.

2.3 ID alignment

The first step in collaborative modeling under vertical data split setting is secure entity alignment, also known as Private Set Intersection (PSI). That is, in each communication round, participants align their entities (nodes) without exposing those that do not overlap with each other. The server should record all the overlapped entities without knowing the details of any entity. Therefore, we need a privacy-preserving protocol to conduct ID alignment. In this way, the server will only aggregate the overlapped entity embeddings from participants. Note that training and test sets for SplitGNN need to be privately aligned among participants before the training and test process start.

2.4 Local Node Embeddings

For public datasets (ACM and IMDB), in our work, we adopt the pre-trained BERT [10] to generate node embeddings. For instance, the movies' names are encoded as 768 embeddings by using the pre-trained BERT and deal the same operation with paper nodes in the ACM dataset. In this way, we can get rich semantic information of nodes, and keep similar nodes closer. In SplitGNN, participants generate initial node embeddings using their own node features, individually. For participant $i \in \mathcal{P}$, this can be done by $\mathbf{h}_0^i = (\mathbf{x}^i)^T \cdot \mathbf{W}^i$, where $\mathbf{x}^i$ and $\mathbf{W}^i$ are node features and weight matrix of participant i. For local node embeddings, similar to the existing GNNs, we perform multi-hop neighborhood aggregation on graphs using private edge information individually. Under the data isolated setting, neighborhood aggregation should be done by participants separately, rather than cooperatively, to protect the private edge information. This is because one may infer the neighborhood information of v given the neighborhood aggregation results of k-hop ($h_v^k(i)$) and $(k+1)$-hop ($h_v^{k+1}(i)$), if neighborhood aggregation is done by participants together. A special case is $h_v^k(i) = h_v^{k+1}(i)$, where it is likely that v is an isolated node in the graph of participant i. Therefore, to protect graph privacy, we let participants perform multi-hop neighborhood aggregation separately using their own graphs.

For $\forall v \in V$ at each participant, neighborhood aggregation is the same as the traditional GNN. Take GraphSAGE as an example, GraphSAGE introduces aggregator functions to update hidden embeddings by sampling and aggregating features from a node's local neighborhood: the aggregator functions AGG are of three types, i.e., Mean, LSTM, and Pooling.

After participants generate local node embeddings, they need to send their local node embeddings to a semi-honest server for combination and further computations. Although the local node embeddings h_v hide raw information of local graphs, it may still encode the sensitive information of local graphs, hence participants need to either encrypt or perturb the extracted embeddings before sending to the server for further global computation.

2.5 Details of HAT

In this section, we will introduce how the HAT works and explain the equations in it.

2.5.1 Metapath Sampling With Relation Features. In Heterogeneous Attention (HAT), both pre-defined metapaths and original relations are taken to generate input subgraphs. For instance, in IMDB dataset, original relations (*e.g.* actor, director) and metapaths (*e.g.* MDM, MAM) are adopted to sample subgraphs as inputs. To capture more complex representations, when using pre-defined metapaths, we concatenate all the features of nodes and edges lying on the metapath. An example is shown in Figure 2. For the graph sampling process, all the original relations of neighbors will be considered to generate input subgraphs. By contrast, random sampling will miss some important information [11], leading to weak model performance.

2.5.2 BERT Encoding. For machine learning, model performance usually depends on the input features heavily. Previous works that encode all words (description of each node) by adopting the one-hot method to generate node features will drive node features to a sparse

Algorithm 2 Privacy-preserving SplitGNN for node label prediction (forward propagation)

Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E}^i)$ and node features $\{\mathbf{f}_{i,v}, \forall v \in \mathcal{V}\}$ on participant $i, i \in \mathcal{P} = \{1, ..., I\}$; depth K; aggregator functions $\text{HAT}_k, \forall k \in \{1, ..., K\}$; weight matrices $\mathbf{W}_i^k, \forall i \in \mathcal{P}, \forall k \in \{1, ..., K\}$; max layer L; weight matrices $\mathbf{W}_l, \forall l \in \{0, ..., L\}$; non-linearity σ; neighborhood functions $\mathcal{N}^i : v \to 2^{\mathcal{V}}, \forall i \in \mathcal{P}$; node labels on participant $p \in \mathcal{P}$ and $c \in C$; $\rho \in \Phi$ (relation sets)

Output: Node label predictions $\{\hat{y}_{vc}, \forall v \in \mathcal{V}, \forall c \in C\}$ on participant p

P1: private feature and edge related computations by participants **participants**:

while $i \in \mathcal{P}$ in parallel **do**
  while $k = 1$ to $K, v \in \mathcal{V}, \rho \in \Phi$ **do**
    $\mathbf{f}_v^k(i) \leftarrow \text{HAT}_k(\{\mathbf{f}_u^{k-1}(i, u), \forall u \in \mathcal{N}^i(v)\})$
  end while
  $\mathbf{h}_v^K(i) = \mathbf{f}_v^K(i), \forall v \in \mathcal{V}$ and sends (publishs) the to server
end while

P2: server-side layer computations by the interactive server as shown in Figure 1

while $v \in \mathcal{V}$ **do**
  aggregates the local node embeddings from participants $\mathbf{h}_v^K =$ Aggregate $(\{\mathbf{h}_v^K(i), \forall i \in \mathcal{P}\})$
  forward propagation based on the global node embeddings $\mathbf{z}_L = \sigma(\mathbf{W}_{L-1} \cdot \sigma(...\sigma(\mathbf{W}_0 \cdot \mathbf{h}_v^K)))$
  sends $\mathbf{z}_L$ to participant p
end while

P3: private label related computations by participant who has label **participant** p: makes prediction by $\hat{y}_{vc} \leftarrow \text{softmax}(\mathbf{W}_L \cdot \mathbf{z}_L), \forall v \in \mathcal{V}, \forall c \in C$

space. Instead, in our work, we transform the descriptions (names, titles) of nodes to a node feature vector by adopting a pre-trained BERT model [10]. We then derive the representation for each node by averaging the produced node features weighted by each word's attention. In this way, we can make similar sentences become closer in vector distance. Throughout this work, we take node feature dimension as 764. Additionally, the relation embedding is trainable, which can be learned by the back-propagation technique.

2.5.3 Node Attention.

$$\begin{aligned} h_i^l &= W^{\tau,l}\mathbf{f}_i^l + b^{\tau,l} \\ r_{i,j}^{\rho,l} &= W^{\rho,l}\mathbf{e}_i^l + b^{\rho,l} \\ h_i^{\rho,l} &= \Phi(h_i^l, r_{i,j}^{\rho,l}) \end{aligned} \tag{1}$$

where $\mathbf{f}_i^l$ and $\mathbf{e}_i^l$ refer to node features and edge features, h_i^l and $r_{i,j}^{\rho,l}$ are latent vectors of i-th node and the edge which is the relation between the target node and i-th node in l-th layer of the ρ relation, respectively. $W^{\tau,l} \in \mathbb{R}^{D_n \times d}$ and $W^{\rho,l} \in \mathbb{R}^{D_e \times d}$ stand for transformation weights of the node (type τ) and edge (relation ρ) in the l-th layer respectively.

The function Φ represents the concatenation function which is used to combine node and edge latent vectors. In fact, other functions (*e.g.* linear transformation, element-wise addition) can be adopted as well. $h_i^{\rho,l}$ donates the node hidden status of l-th layer. The feature transformation stage will be processed in each subgraph for $\mathcal{N}_i^\rho$. But for the target node, $h_i^{\rho,l} = h_i^l$.

$$\mathbf{z}_i^{\rho,l} = \mathbf{F}_\rho(h_i^{\rho,l}, h_j^{\rho,l}) \tag{2}$$

Where $\mathbf{F}_\rho$ represents the aggregation function in the ρ relation (or metapath). $\mathbf{z}$ donates node's hidden status embedding after node fusion. After this step, for one specific relation, all nodes' information is treated as one vector which can represent all the information of the target node, neighbor (middle neighbor) nodes, and relations between them. Here, we adopt the attention mechanism to integrate node embeddings into one vector.

$$\mathbf{z}_i^{\rho,l} = \sigma(\Big\|_{m=1}^{M} \sum_{j \in \mathcal{N}_i^\rho} \alpha_{i,j}^{\rho,l,m} h_j^{\rho,l}) \tag{3}$$

Here, M is the multi-head number. In our work, we use Eq. (3) to aggregate node embeddings when processing the fusion stage. Where σ is an active function. To consider the impact of different neighbor (middle neighbor) nodes on the target node, all neighbors have calculated attention coefficients in a given relation ρ.

$$\alpha_{i,j}^{\rho,l,m} = \frac{exp(h_i^{\rho,l} \cdot h_j^{\rho,l})}{\sum_{k \in \mathcal{N}_i^\rho} exp(h_i^{\rho,l} \cdot h_k^{\rho,l})} \tag{4}$$

After this stage, every specific relation (metapath) would be simplified as one hidden state vector which contains all information of its subgraphs.

2.5.4 Path Attention.

In the heterogeneous graph, there may exist multi-relations between two nodes. Based on this, path-level attention is needed which can reflect how the target node is impacted by different relations.

$$\mathbf{f}_i^{l+1} = \sigma(\Big\|_{m=1}^{M} \sum_{\rho} \beta_i^{\rho,l,m} \mathbf{z_i}^{\rho,l}) \tag{5}$$

$$\beta_i^{\rho,l,m} = \frac{exp(\mathbf{q}^{\rho T} \cdot \mathbf{z}_i^{C,l})}{\sum_{\rho'} exp(\mathbf{q}^{\rho' T} \cdot \mathbf{z}_i^{C,l})} \tag{6}$$

where $\beta_i^{\rho,l,m}$, $\mathbf{q}^{\rho T}$ are the path-attention coefficient and the attention vector for ρ relation in the l-th layer, respectively. $\mathbf{z}_i^{C,l}$ is the concatenation of all $\mathbf{z}_i^{\rho,l}$. After this process, we can obtain target all node local embeddings $\mathbf{f}_i^{l+1}$.

2.6 Server Aggregation

After receiving local node embeddings from all participants, the semi-honest server generates global node embeddings by combining local node embeddings. Based on the aggregated global node embeddings, the server can conduct the successive computations, e.g., the non-linear operations such as max-pooling and activation functions in deep network structures. Finally, the server returns the final hidden layer to the party that has labels to compute prediction and loss. The participant who has the label can compute the prediction using the final hidden layer it receives from the server. For node classification task, the Softmax activation function is used for the output layer,

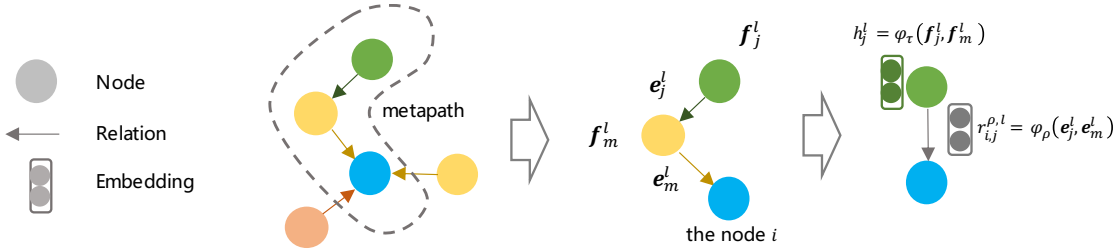

Figure 2: Metapath sampling process: generating latent embedding for neighbor node. Different colors represent different node and relation types. Nodes and relations within dash line is the pre-defined metapath. $\mathbf{f}_j^l$ and $\mathbf{e}_j^l$ stand for node and edge features in l-th layer. In this method, $h_j^l = \varphi_\tau(\mathbf{f}_j^l, \mathbf{f}_m^l)$ and $r_{j,m}^{\rho,l} = \varphi_\rho(\mathbf{e}_j^l, \mathbf{e}_m^l)$. Here, h_j^l and $r_{j,m}^{\rho,l}$ are the same as in Eq. (1). $\varphi_\tau(\mathbf{f}_j^l, \mathbf{f}_m^l) = W^{\tau,l}[\mathbf{f}_j^l, \mathbf{f}_m^l] + b^{\tau,l}$ and $\varphi_\rho(\mathbf{e}_j^l, \mathbf{e}_m^l) = W^{\rho,l}[\mathbf{e}_j^l, \mathbf{e}_m^l] + b^{\rho,l}$. Additionally, τ denotes the type of node, and ρ is the specific relation.

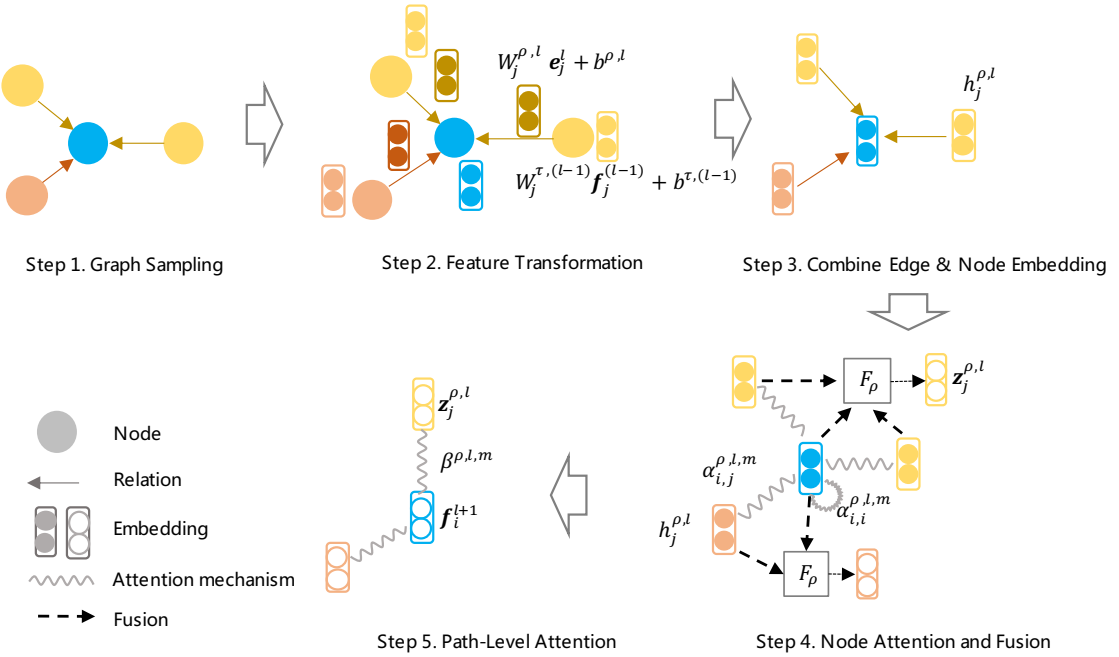

Figure 3: The overall workflow of HAT. The left graph is the original heterogeneous graph. Different colors present different node and relation types. After sampling, it is divided into two subgraphs. In feature transformation, different node types' features and edge features are transferred into the same latent dimension size. In Step 4, the attention mechanism is adopted to extract information from different nodes. Lastly, the path attention mechanism is used to add all relations' information into the final embedding of this layer. After looping all layers, the latent embedding is decoded by MLP.

which is defined as $\text{softmax}(z_c) = \frac{1}{Z}\exp(z_c)$ with $c \in C$ be the node class and $Z = \sum_c \exp(z_c)$.

During each communication round in our framework, the server is responsible for aggregating node embeddings from different participants, conducting aggregation on the overlapped node embeddings. Note that during this procedure, all the private data (including node attributes, edge information, labels, and local model gradients) related computations are carried out by participants locally, the server knows nothing about local data, except the encrypted local embedding.

In terms of embedding aggregation, the combination strategy would be trainable in order to maintain high representational capacity. We introduce three combination strategies, i.e., averaging, concatenation, and weighted averaging as follows. Note that adopting averaging or weighted averaging in contrast to direct concatenation allows the server's input dimension to remain fixed – independent of the number of participants.

Average. The average operator takes the elementwise average of the vectors in $(\{\mathbf{h}_v^K(i), \forall i \in \mathcal{P}\})$, assuming participants contribute

equally to the global node embeddings, i.e.,

$$\mathbf{h}_v^K \leftarrow \text{Average}(\{\mathbf{h}_v^K(1), \mathbf{h}_v^K(2), \cdots, \mathbf{h}_v^K(I)\}) \tag{7}$$

Concatenation. The concatenation operator can fully preserve local node embeddings learnt from different participants. The operation can be written as bellow:

$$\mathbf{h}_v^K \leftarrow \text{Concat}(\{\mathbf{h}_v^K(1), \mathbf{h}_v^K(2), \cdots, \mathbf{h}_v^K(I)\}) \tag{8}$$

Weighted Average. The average strategy treats participants equally. In reality, the local node embeddings from different participants may contribute diversely to the global node embeddings. We propose a weighted averaging strategy to handle this situation. The weighted average operator aims to aggregate the embedding elements from participants through a regression model, whose parameters are learned intelligently during training. Let $\boldsymbol{\omega_i}$ be the weight vector of local node embeddings from participant $i \in \mathcal{P}$, then

$$\mathbf{h}_v^K \leftarrow \boldsymbol{\omega_1} \odot \mathbf{h}_v^K(1) + \boldsymbol{\omega_2} \odot \mathbf{h}_v^K(2) ... + \boldsymbol{\omega_I} \odot \mathbf{h}_v^K(I) \tag{9}$$

where $\odot$ is element-wise multiplication. Regression can handle the situation where the data quality and quantity (feature and edge size) of participants are different from each other.

These different combination operators can utilize local node embeddings in diverse ways, and we will empirically study their effects on model performances in Sec. 3.

We summarize the process of our privacy-preserving SplitGNN in Algorithm 2. P1 describes how participants generate local node embeddings separately (Section 2.4), i.e., perform multi-hop neighborhood aggregation using edge information. P2 corresponds to the computations by the server-side layers, where Line 10 shows how to generate global node embeddings on the server (Section 2.6). P3 shows how the server makes forward propagation to get the last hidden layer. Line 14 shows how the participant who has the label conducts private label-related computations using the last hidden layer.

2.7 Privacy Preservation

Unlike previous works that assume only samples (nodes) are held by different parties and these samples have no relationship, our task is more challenging: for decentralized graph structure data, both nodes and edges are kept by different participants, GNN relies on the relationships between nodes, which makes most of the privacy learning methods designed for conventional datasets infeasible. In particular, we want to ensure that all the forward and backward communications between the participating entities (e.g., the smashed data and gradients between participants and the server) are performed in an encrypted form. To be concrete, we consider privacy from the following perspectives: (1) how to align ID in a privacy-preserving manner; (2) how to aggregate local node embeddings in a privacy-preserving manner.

For privacy-preserving ID alignment, each participant only shares their hashed node index list with the server. The hashed index list is used to index and distinguish nodes from all local graphs to hide the raw index information from the server.

For the privacy-preserving aggregation of local node embeddings, since each local subgraph contains sensitive information about nodes, edges, attributes, and labels, the value of intermediate features has the potential risk to reveal sensitive information about the input data [12]. Henceforth, the intermediate representations should be transmitted in a secure and communication-efficient way. To provide the provable privacy guarantee, various privacy-preserving techniques can be considered, including secure aggregation [13], differential privacy (DP) [14]. To provide high security guarantees against the semi-honest server, we adopt homomorphic encryption (HE) [15] (Section 2.6). In this way, the server can only decrypt the aggregation of local embeddings.

3 EXPERIMENTS

3.1 Dataset, Task and Parameters

We evaluate our SplitGNN on two real-world datasets (ACM and IMDB) with multiple classes. There are multi-classes for both ACM and IMDB. Throughout this work, the Micro F1 score is adopted as the evaluation metric for node classification.

Parameters Setting. We perform batch gradient descent, i.e., we sample a batch of nodes ($B = 512$) in each communication round and report the performance after five communication rounds. We set the L2 regularisation as 1e-4. For all models, we use Relu as the active function of neighbor propagation, and the active function of hidden layers. For the deep neural network on the server, we set the dropout rate to 0.3 and network structure as (d, d, $|C|$), where $d \in \{32, 64, 128\}$ is the dimension of node embeddings and $|C|$ is the number of classes.

Baselines. We compare with the following baselines.

- Standalone. Each participant trains its own GNN on their limited local data without any collaboration. Hence, it cannot utilize information from other participants.
- Entire. All participants pool their local data to a central server, which trains a global GNN on all the combined data. Note that this setting violates privacy protection because data are directly exposed during the procedure of collection.

3.2 Experimental Results

We first summarize the results in Table 1, where SplitGNN$_m$, SplitGNN$_c$, and SplitGNN$_w$ denote SplitGNN with Average, SplitGNN with Concatenation, and Trainable Weighted Average combination strategies. It can be clearly observed that our SplitGNN significantly outperforms the GNNs by using the isolated data and has comparable performance with the traditional GNN by using the entire plain data insecurely. The reason is straightforward: the learned global node representation of SplitGNN is similar to that learned over the combined graph.

We next explore how the interactive layer strategy, number of participants, and partition ratio affect model performance.

3.2.1 Impact of the interactive layer strategy. From Table 1, we find that the concatenation strategy performs the best generally. This is because the concatenation strategy can more fully integrate nodes' information coming from different clients. Moreover, we also find that our HAT works well on all datasets and strategies since it treats different relations with different attention coefficients, and values more important relations with higher weights intelligently.

3.2.2 Impact of the number of participants. We vary the number of participants in {2, 4, 8} and study the performance of SplitGNN. We report the results in Table 2, where we use the ACM

Table 1: Micro F1 Score of different base GNN models (Data is vertically partitioned (Ratio=5:5) among 2 participants A and B).

Dataset	Strategy	GCN	GAT	HAT
ACM	Entire	0.9007	0.9028	0.9158
	Standalone$_A$	0.7075	0.7134	0.7282
	Standalone$_B$	0.7087	0.7146	0.7307
	SplitGNN$_w$	0.8812	0.8837	0.8903
	SplitGNN$_c$	0.8827	0.8864	0.8984
	SplitGNN$_m$	0.8811	0.8816	0.8960
IMDB	Entire	0.5549	0.5396	0.5694
	Standalone$_A$	0.4273	0.4188	0.4429
	Standalone$_B$	0.4209	0.4185	0.4403
	SplitGNN$_w$	0.5385	0.5267	0.5567
	SplitGNN$_c$	0.5489	0.5251	0.5573
	SplitGNN$_m$	0.5433	0.5235	0.5562

dataset and assume participants have even feature and edge data. We find that, as the number of participants increases, the accuracy of all the models decreases, and the gaps widen with the increase of participants. We hypothesize this is because the neighborhood aggregation in SplitGNN is done by each participant individually for privacy concerns, and each participant will have fewer edge data when there are more participants since they split the original edge information evenly. Therefore, when more participants are involved, more information is lost during the neighborhood aggregation procedure.

Table 2: Micro F1 Score of different base GNN models (Dataset: ACM, Strategy: concatenation, Ratio: equal size).

Clients	GCN	GAT	HAT
Entire	0.9007	0.9028	0.9158
2	0.8827	0.8864	0.8984
4	0.8753	0.8778	0.8903
8	0.8654	0.8670	0.8811

3.2.3 Impact of data distribution. In the real scenario, participants may have different data distributions (i.e. domains), exhibiting statistical heterogeneity. For example, distinct participants may speak different languages, take pictures at different locations. Under non-IID setting, it is common that the accuracy and convergence speed of distributed learning may be significantly degraded [16].

To investigate how the data distribution impacts the model performance, we study this by varying the proportion (Prop.) of data (node features and edges) held by A and B in 5:5, 3:7, 1:9. The results on Cora dataset are shown in Table 3. Based on results, we find that with the proportion of data held by A and B is even, i.e., from 1:9 to 5:5, the performances of most strategies tend to decrease. This is because the neighbor aggregation is done by data holders individually, and with a larger proportion of data held by a single holder, it is easier for this party to generate better local node embeddings.

Table 3: Micro F1 Score of different base GNN models (Dataset: ACM, Strategy: concatenation).

Ratio	GCN	GAT	HAT
Entire	0.9007	0.9028	0.9158
5:5	0.8827	0.8864	0.8984
3:7	0.8908	0.8928	0.9059
1:9	0.8998	0.9022	0.9183

3.3 Complexity Analysis

During the training process, we conduct mini-batch training. Communication cost is dependent on both the message size in each communication round and the total number of communication rounds, hence it can be expressed as $O(B*e*R)$, where B, e, and R represent the sampled batch size, embedding size, and communication rounds respectively. For FL, the communication cost is relative to $O(P*N)$, here P and N denote the number of model's parameters and the number of clients, respectively.

Compared with FL, SL is more communication efficient with an increase in the number of participants or model size [17], and decrease with the scale of dataset (There are 12499 and 4780 nodes in ACM and IMDB). In contrast, in FL, the communication efficiency will skyrocket, when the number of participants is large, as shown in Figure 4.

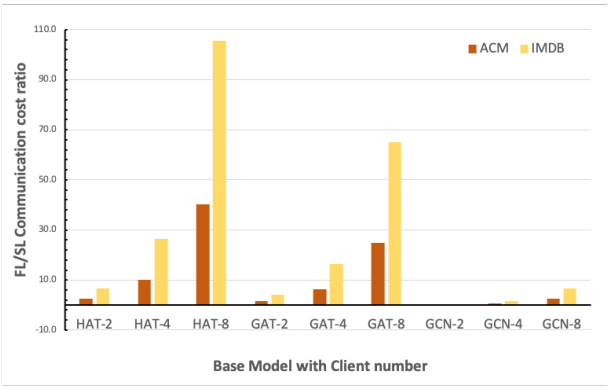

Figure 4: SplitGNN Complexity Analysis.

4 EXPERIMENTS ON REAL DATASET

As introduced above, we find that our SplitGNN is effective and HAT is suitable for heterogeneous graphs. Next, we will adopt them to deal with the real industrial dataset which is a real fintech dataset from Alipay [2] - the world's leading mobile payment platform serving more than 450 millions of users.

4.1 Data Introduction

As shown in Fig. 4, there is a huge difference in the magnitude of the two sides on the real dataset. After the PSI process, party A has 1132511 nodes and 2371270 relations which means all nodes in party B are overlapped with the party A. In addition, labels are provided

[2]https://en.wikipedia.org/wiki/Ant_Financial

by party B with two classes. The proportion of positive and negative samples is about 0.12%. Although the number of the relation types in parts A and B is the same, the meanings they represent are very different. In A, relation types are cash transfer and social relation. But for B, they mean similarity and medium shared.

4.2 Settings and Results

For the real dataset, we set the batch size as 512. Because of the uneven proportion of positive and negative samples, we have tried upsampling for negative samples 3, 5, and 10 times. We find that the 5 times upsampling has the best performance. The hidden size, dropout ratio, and L2 are set as 128, 0.3, and 0.0001, respectively. Besides, as mentioned above, HAT has better capability than other base models. For the real dataset, we adopt HAT as the base model.

Based on Table 5, the results show that the concatenation strategy performs best. This is because the concatenation strategy can fully integrate nodes' information coming from these two participants, as mentioned before.

5 PRELIMINARY AND RELATED WORK

5.1 Federated Learning

Federated learning (FL) enables a multitude of participants to construct a joint ML model without exposing their private training data. In recent years, FL has benefited a wide range of applications such as named entity recognition [18], Native Ad CTR Prediction [19], etc. FL offers a privacy-aware paradigm of model training that does not require data sharing. However, in FL, each participant needs to train a whole model.

One of the standard aggregation methods in FL is FedAvg [20], where parameters of local models are averaged element-wise with weights proportional to the client data size. However, FedAvg is designed for horizontal data split settings, which cannot be directly used in vertical data split setting. Additionally, recent works [21] attempt to extend federated learning to vertical settings under less realistic assumptions, where one party has data, but the other party has labels only. The differences between Horizontal and Vertical settings are listed in Table 6.

5.2 Split Learning

Split learning (SL) [22] aims to tackle with high communication cost between cloud and participant, the limited memory of edge devices, and the high computation of local model. In SL, a whole computation graph of neural network $\mathbf{W}$ is split into two portions: participant-side layers ($\mathbf{W}^{\mathrm{C}}$, from first layer to k layer) and server-side layers ($\mathbf{W}^{\mathrm{S}}$, from $k+1$ layer to output layer). SL allows participants to calculate the private data-related computations individually and get a hidden layer, and then let a server make the rest computations [6]. By connecting the lower layers with shared upper layers stored on a parameter server, each device uploads its NN activations of the split layer (i.e., participant-side network's last layer) to the server to calculate the loss values, and downloads the gradients to update its participant-side network.

SL and FL are all collaborative deep learning techniques. The key difference lies in whether the model is split and trained separately or not. In terms of data privacy, in FL, the server has access to the entire model. In contrast, the server has only access to the server-side portion of the model and the smashed data (i.e., activation vectors of the split layer) from each participant. In contrast to FL, SL provides better privacy due to the model architecture split between the participants and the server. The extracted representations from the split layer are considered to contain less information than the original data, and the inversion from representations is strictly more difficult than recovery from the model or gradient information [23]. Meanwhile, SL provides a more practical paradigm for participants with limited resources (edge devices in IoT), as participants only need to train the first few layers of the split ML network model, instead of running the complete network, which requires significant computation on resource-constrained devices.

Our model differs from the traditional split learning in mainly two aspects. First, we train a GNN rather than a simple neural network. Second, we use cryptographic techniques for participants to aggregate their local node embeddings in a privacy-preserving manner rather than sharing their plain embeddings. Please see the supplementary material for detailed related work.

5.3 Graph Neural Network

GNN extended existing neural networks for processing the data represented in graph domains. The key of GNN is to learn a function f to generate node embeddings $\mathbf{h}_v$ for each node v in a graph based on its own features $\mathbf{x}_v$ and neighbors' features $\mathbf{x}_u, u \in \mathcal{N}(v)$, with $\mathcal{N}(v)$ denoting the neighborhood function in a graph [24]. After this, with the generated node embeddings, one can do further tasks such as classification [25], link prediction [26], and recommendation [27] using deep neural networks. In this paper, we focus on the node classification task. The key to this task is the design of an *aggregator function*, which learns to aggregate feature information from the node's local neighborhood [28]. To date, different types of aggregator functions have been proposed, e.g., convolution-based [29], gated mechanism based [30], and attention based [31].

For node representation, there is a range of methods with or without metapaths. For node representation with metapaths, Metapath2Vec [32] is used to generate node embeddings which can be treated as features for the next step. Heterogeneous information networks to vector (HIN2vec) [33] carries out multiple prediction training tasks to learn representations of nodes and metapaths of a heterogeneous graph. Given a metapath, HERec [34] converts a heterogeneous graph into a homogeneous graph. Based on metapaths and neighbors, HERec applies a DeepWalk model [35] to learn the node embedding of the target type. Moreover, the metapath is used to generate subgraphs for graph neural networks. Wang *et al*. introduce heterogeneous graph attention network (HAN) [36], which uses the metapath to sample neighbors and all input nodes will be of the same type. On the other hand, there are many methods without metapaths. For example, in RGCN [37], every relation (including self-loop) is aggregated in one block which will be added together to get the final embedding. Heterogeneous graph neural networks (HetGNN) [38] use the random walk to sample neighbors. HetGNN then aggregates messages considering the effects of different node types. Ziniu *et al*. [39] propose heterogeneous graph transformer (HGT) that designs node- and edge-type dependent parameters to characterize the heterogeneous attention over each edge.

Although all the above methods can learn node embedding, they have overlooked the relation importance in heterogeneous graphs.

Table 4: Node and Edge in Real Dataset.

Party	Node	Relation	Relation type	Node Features	Edge Features
A	19206184	14132228265	4	155	2
B	1132511	105392	2	76	2

Table 5: Micro F1 Score of different base GNN models (Dataset: Real Data).

Strategy	HAT
Entire	0.8601
SplitGNN$_c$	0.8574
SplitGNN$_w$	0.8493
SplitGNN$_m$	0.8367

Table 6: Graph-structured data partition

Mode	Nodes	Edges	Features
Horizontal	different	different	aligned
Vertical	aligned	not limited	not limited

This gap can however be filled by other methods. For example, Weighted-GCN (WGCN) [40] assigns a learnable scalar weight to each relation and multiplies an incoming "message" by this weight. However, WGCN only learns a scalar, rather than relation embeddings.

5.4 Additively Homomorphic Encryption

A homomorphic encryption scheme allows arithmetic operations to be directly performed on ciphertexts, which is equivalent to a specific linear algebraic manipulation of the plaintext. Existing homomorphic encryption techniques can be categorized into: 1) fully homomorphic encryption, 2) somewhat homomorphic encryption, and 3) partially homomorphic encryption. Fully homomorphic encryption can support arbitrary computation on ciphertexts but is less efficient [41]. On the other hand, somewhat homomorphic encryption and partially homomorphic encryption are more efficient but are specified by a limited number of operations [42–45]. Well-known partially homomorphic encryption schemes include RSA [43], El Gamal [44], Paillier [45], etc. The homomorphic properties can be described as:

$$E_{pk}(m_1 + m_2) = c_1 \oplus c_2$$
$$E_{pk}(a \cdot m_1) = a \otimes c_1$$

where a is a constant, m_1, m_2 are the plaintexts that need to be encrypted, c_1, c_2 are the ciphertext of m_1, m_2 respectively.

5.5 Background for Heterogeneous Graph

Definition 1 (Heterogeneous Graph [46]). A directed graph can be written as $\mathrm{G} = (\mathcal{V}, \mathcal{E})$. Here, $\mathcal{V}$ and $\mathcal{E}$ are sets of multiple type nodes and multiple type relations belonging to G. Moreover, heterogeneous graph is associated with a node mapping function ϕ: $\mathcal{V} \rightarrow \mathcal{A}$ and a link mapping function χ: $\mathcal{E} \rightarrow \mathcal{R}$. $\mathcal{A}$ and $\mathcal{R}$ represent the sets of pre-defined node types and relation types. In addition, it should be satisfied with the requirement $|\mathcal{A}| + |\mathcal{R}| >= 2$.

EXAMPLE 1. *The sample of IMDB dataset consists of three node types, i.e., director (D), moive (M) and actor (A) and two relation types, i.e., direct and act. All nodes and relations assemble sets $\mathcal{V}$ and $\mathcal{E}$, respectively. Also, in this heterogeneous graph, $|\mathcal{A}| + |\mathcal{R}| >= 2$ holds.*

Definition 2 (Metapath [47]). A metapath ρ is defined as a path lying on heterogeneous graph G in the form of $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \cdots \xrightarrow{R_l} A_{l+1}$ (also written as $A_1A_2 \cdots A_{l+1}$), which describes a composite relation $R = R_1 \circ R_2 \circ \cdots R_l$ between A_1 and A_{l+1}, where $\circ$ refers to the composition operator on relations. For a heterogeneous graph, two nodes can be connected by the pre-defined metapaths. Such metapaths can be considered as heuristic methods and the rules of the definition of metapaths typically come from the expert's experience.

EXAMPLE 2. *Movie A and movie B share the same director presenting as $Movie-Director-Movie$ (MDM) by metapath. Similarly, actor A and actor B can be written as $Actor - Movie - Director - Movie - Actor$ ($AMDMA$) which indicates these two actors are connected by the same director D.*

Definition 3 (Metapath neighbors and middle neighbors [36]). Given a specific metapath ρ and a node i in a heterogeneous graph, the metapath neighbor $\mathcal{N}_i^\rho$ of metapath ρ for node i is defined by the set of last nodes connected with i by metapath ρ. Let S_i^ρ denote the set of nodes lying on the metapath ρ of i. The **middle neighbor** is defined by $\mathcal{P}_i^\rho = S_i^\rho - \mathcal{N}_i^\rho - \{i\}$.

EXAMPLE 3. *If we define a metapath between actor A and actor B by $Actor - Movie - Director - Movie - Actor$ ($AMDMA$). We say actor B is a metapath neighbor of actor A, and {movie A, director A, movie B} is the set of middle nodes within this pre-defined metapath.*

6 CONCLUSION AND DISCUSSION

In this paper, we have presented a novel approach, named SplitGNN, which splits the computation graph of GNN by leaving the private data-related computations to participants and delegating the rest computations to the server. To learn various types of nodes and edges in the heterogeneous graphs under the vertically partitioned setting, we have proposed a novel algorithm called HAT. We also conducted the communication efficiency experiment, the result of which demonstrated our SplitGNN incurs lower communication costs than FL.

In addition to HE considered in our work, previous works have applied DP on the local node embeddings [12, 48] when the server generates global node embeddings, to further protect potential information leakage. Compared to the encryption-based schemes, DP-based approaches are more computationally efficient [49], however, it may hurt the utility. Therefore, we leave the comparative performance analysis of SplitGNN with the integration of DP to future work.

REFERENCES

[1] H. Will, Y. Zhitao, and L. Jure, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates and Inc., 2017, pp. 1024–1034.

[2] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017.

[3] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018.

[4] L. Zheng, J. Zhou, C. Chen, B. Wu, L. Wang, and B. Zhang, "Asfgnn: Automated separated-federated graph neural network," *Peer-to-Peer Networking and Applications*, vol. 14, no. 3, pp. 1692–1704, 2021.

[5] C. Shan, H. Jiao, and J. Fu, "Towards representation identical privacy-preserving graph neural network via split learning," *arXiv preprint arXiv:2107.05917*, 2021.

[6] S. A. Osia, A. S. Shamsabadi, S. Sajadmanesh, A. Taheri, K. Katevas, H. R. Rabiee, N. D. Lane, and H. Haddadi, "A hybrid deep learning architecture for privacy-preserving mobile analytics," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4505–4518, 2020.

[7] J. Zhou, C. Chen, L. Zheng, X. Zheng, B. Wu, Z. Liu, and L. Wang, "Privacy-preserving graph neural network for node classification," *arXiv e-prints*, pp. arXiv–2005, 2020.

[8] S. Hardy, W. Henecka, H. Ivey-Law, R. Nock, G. Patrini, G. Smith, and B. Thorne, "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption," *arXiv preprint arXiv:1711.10677*, 2017.

[9] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptol. ePrint Arch.*, vol. 2016, no. 86, pp. 1–118, 2016.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[11] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 631–636. [Online]. Available: https://doi.org/10.1145/1150402.1150479

[12] L. Lyu, Y. Li, X. He, and T. Xiao, "Towards differentially private text representations," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020, pp. 1813–1816.

[13] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1175–1191.

[14] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," *CoRR, arXiv:1712.07557*, 2017.

[15] Y. Aono, T. Hayashi, L. Wang, S. Moriai *et al.*, "Privacy-preserving deep learning via additively homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2017.

[16] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[17] A. Singh, P. Vepakomma, O. Gupta, and R. Raskar, "Detailed comparison of communication efficiency of split learning and federated learning," *arXiv preprint arXiv:1909.09145*, 2019.

[18] S. Ge, F. Wu, C. Wu, T. Qi, Y. Huang, and X. Xie, "Fedner: Medical named entity recognition with federated learning," *arXiv preprint arXiv:2003.09288*, 2020.

[19] C. Wu, F. Wu, T. Di, Y. Huang, and X. Xie, "Fedctr: Federated native ad ctr prediction with multi-platform user behavior data," *arXiv preprint arXiv:2007.12135*, 2020.

[20] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, 2017, pp. 1273–1282.

[21] Y. Zhang and H. Zhu, "Additively homomorphical encryption based deep neural network for asymmetrically collaborative machine learning," *arXiv preprint arXiv:2007.06849*, 2020.

[22] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, 2018.

[23] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients–how easy is it to break privacy in federated learning?" *arXiv preprint arXiv:2003.14053*, 2020.

[24] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv:1901.00596*, 2019.

[25] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[26] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 5165–5175.

[27] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 974–983.

[28] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," *arXiv preprint arXiv:1812.08434*, 2018.

[29] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1024–1034.

[30] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[31] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[32] D. Yuxiao, C. N. V., and S. Ananthram, "Metapath2vec: Scalable representation learning for heterogeneous networks," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17. New York and NY and USA: Association for Computing Machinery, 2017, p. 135–144. [Online]. Available: https://doi.org/10.1145/3097983.3098036

[33] F. Tao-yang, L. Wang-Chien, and L. Zhen, "Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, ser. CIKM '17. New York and NY and USA: Association for Computing Machinery, 2017, p. 1797–1806. [Online]. Available: https://doi.org/10.1145/3132847.3132953

[34] C. Shi, B. Hu, W. X. Zhao, and P. S. Yu, "Heterogeneous information network embedding for recommendation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 2, pp. 357–370, 2019.

[35] P. Bryan, A.-R. Rami, and S. Steven, "Deepwalk: Online learning of social representations," ser. KDD '14. New York and NY and USA: Association for Computing Machinery, 2014, p. 701–710. [Online]. Available: https://doi.org/10.1145/2623330.2623732

[36] Wang, Xiao, Ji, Houye, Shi, Chuan, Wang, Bai, Ye, Yanfang, Cui, Peng, Yu, and P. S, "Heterogeneous graph attention network," in *The World Wide Web Conference*, ser. WWW '19. New York and NY and USA: Association for Computing Machinery, 2019, p. 2022–2032. [Online]. Available: https://doi.org/10.1145/3308558.3313562

[37] S. Michael, K. T. N., and B. Peter, "Modeling relational data with graph convolutional networks," in *The Semantic Web*, Gangemi, Aldo, and N. Roberto, Eds. Cham: Springer International Publishing, 2018, pp. 593–607.

[38] Z. Chuxu and S. Dongjin, "Heterogeneous graph neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery; Data Mining*, ser. KDD '19. New York and NY and USA: Association for Computing Machinery, 2019, p. 793–803. [Online]. Available: https://doi.org/10.1145/3292500.3330961

[39] H. Ziniu, D. Yuxiao, W. Kuansan, and S. Yizhou, "Heterogeneous graph transformer," in *Proceedings of The Web Conference 2020*, ser. WWW '20. New York and NY and USA: Association for Computing Machinery, 2020, p. 2704–2710. [Online]. Available: https://doi.org/10.1145/3366423.3380027

[40] C. Shang, Y. Tang, J. Huang, J. Bi, X. He, and B. Zhou, "End-to-end structure-aware convolutional networks for knowledge base completion," 2018.

[41] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, ser. STOC '09. New York, NY, USA: ACM, 2009, pp. 169–178.

[42] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Annual Cryptology Conference*. Springer, 2012, pp. 643–662.

[43] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

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

[45] P. Paillier *et al.*, "Public-key cryptosystems based on composite degree residuosity classes," in *Eurocrypt*, vol. 99. Springer, 1999, pp. 223–238.

[46] S. Yizhou and H. Jiawei, "Mining heterogeneous information networks: A structural analysis approach," *SIGKDD Explor. Newsl.*, vol. 14, no. 2, p. 20–28, apr 2013. [Online]. Available: https://doi.org/10.1145/2481244.2481248

[47] S. Yizhou, H. Jiawei, Y. Xifeng, Y. P. S., and W. Tianyi, "Pathsim: Meta path-based top-k similarity search in heterogeneous information networks," *Proc. VLDB Endow.*, vol. 4, no. 11, p. 992–1003, aug 2011. [Online]. Available: https://doi.org/10.14778/3402707.3402736

[48] Y. Li, T. Baldwin, and T. Cohn, "Towards robust and privacy-preserving text representations," in *Proceedings of ACL*, 2018, pp. 25–30.

[49] S. Sajadmanesh and D. Gatica-Perez, "Locally private graph neural networks," *arXiv preprint arXiv:2006.05535*, 2020.