Scalable Federated Learning for Distributed Graphs

Undergraduate Thesis

Submitted in partial fulfillment of the requirements of
BITS F421T Thesis

By

Sarthak Choudhary
ID No. 2019A7TS0112P

Under the supervision of:

Prof. Prateek Saxena
&
Dr. Ashutosh Bhatia

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, PILANI CAMPUS
November 2023

Declaration of Authorship

I, Sarthak Choudhary, declare that this Undergraduate Thesis titled, 'Scalable Federated Learning for Distributed Graphs' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Certificate

This is to certify that the thesis entitled, "*Scalable Federated Learning for Distributed Graphs*" and submitted by Sarthak Choudhary ID No. 2019A7TS0112P in partial fulfillment of the requirements of BITS F421T Thesis embodies the work done by him under my supervision.

Supervisor	*Co-Supervisor*
Prof. Prateek Saxena	Dr. Ashutosh Bhatia
Assoc. Professor,	Asst. Professor,
National University of Singapore	BITS-Pilani Pilani Campus
Date:	Date:

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, PILANI CAMPUS

Abstract

Bachelor of Engineering (Hons.)

Scalable Federated Learning for Distributed Graphs

by Sarthak Choudhary

Federated learning represents a significant advancement in the field of machine learning, offering a unique approach to model training that addresses some of the most pressing challenges facing the field today. By enabling the training of models on decentralized data sources, federated learning allows for the efficient use of resources, preserves data privacy, and improves model performance. Meanwhile, a lot of users data is highly correlated and best captured by a graphical structure allowing Graph Neural Networks (GNNs) be to state of the art solution. We argue the need of a framework to train GNNs in the federated setup where every client hold their own data and the raw data never leaves their devices.

In this Thesis, I discuss Retexo [11], our framework to train every existing Graph Neural Network in federated setup. Specifically, my work focuses on implementation of different GNNs in our framework, facilitating training on distributed setup, and to devise potential ideas to make the framework robust in order to deploy it without any trust between the parties involved.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Graph neural networks (GNNs) are a type of neural network specifically designed for generating representations of graph-structured data that can be utilized for various applications such as classification, regression, and more. GNNs have been successfully employed in a range of real-world scenarios including recommendation systems, financial fraud detection, and language modeling, where they have been shown to provide superior performance compared to other techniques[21, 24].

Up until now, the design of graph neural networks has primarily focused on a centralized setup where the graph data is stored and processed in centralized servers. However, this approach poses significant security and privacy concerns for user data, making it less than ideal [3]. As concerns around security and privacy of centralized graph data continue to grow, services that offer distributed graph processing and storage are becoming increasingly popular. These services enable the processing of graph data across multiple nodes and devices, reducing the risk of a single point of failure or breach. Examples of popular distributed graph processing services include Apache Giraph, Apache Flink, and GraphX. These services have been utilized in a range of industries including social media, finance, and healthcare, and are proving to be a promising solution for large-scale graph processing and analysis. Examples of applications that can benefit from distributed graph processing include social networks like Signal and Mastodon, which avoid storing social graph data on centralized servers. Additionally, graphs that capture location traces, health data, and financial transactions can also be distributed among untrusting parties. By developing machine learning protocols that support distributed graphs, GNN-based applications can be enhanced with improved privacy and data sovereignty.

In this context, we present the challenge of training neural networks for supervised node classification on fully-distributed graphs. In a fully-distributed graph, each node maintains its features and edges locally on a trusted device such as a mobile or an IoT device. The goal is to enable the clients, along with a service provider's server, to collaboratively learn a

supervised machine learning model that can classify nodes based on their features and the graph structure. This type of setup is relevant for various applications, including health monitoring over Bluetooth-proximity networks, providing recommendations in distributed social networks, and social fitness tracking [4, 6, 13].

Currently available GNNs are not explicitly designed for training in a fully-distributed setup. Attempting to directly retrofit them for such a setup results in substantial communication costs. In the fully-distributed setup, each client processor maintains features for only a single node and its neighboring edges, taking partitioning to its extreme. Consider a standard graph convolutional network (GCN) with K trainable layers. In a fully-distributed setup, training this network would require frequent message-passing, involving the propagation of intermediate node representations and layers across clients during each round of training. This creates a significant communication overhead, particularly for bandwidth-constrained clients. Moreover, clients may not always be available for such communication, which further complicates the training process. Therefore, it is essential to develop GNNs with lower client-to-client communication costs.

Our proposed framework, RETEXO, is the first of its kind to enable the training of neural networks that are transformations of existing message-passing GNN architectures in a fully-distributed setup. This means that architectures like the graph convolutional network (GCN) can be transformed into RETEXO-GCN for training purposes. One of the main advantages of RETEXOGNNs over GNNs is that they require significantly fewer rounds of message-passing between clients. This is made possible by disentangling the aggregation of intermediate representations over graph edges from the end-to-end training of the model. RETEXOGNNs are trained layer-by-layer, with the embeddings (logits) output by each trained layer only propagated over the graph edges once every layer has been fully trained. The propagated embeddings are then aggregated in the same manner as existing GNNs, and the resulting aggregated embeddings are used as features to train the next layer. This process reduces the communication overhead between clients and makes it easier to train RETEXOGNNs in a fully-distributed setup.

While RETEXO is primarily designed to tackle the challenges of fully-distributed setup, it is worth noting that the framework can also be adapted to other distributed settings. For example, RETEXO could be employed to train GNNs efficiently in a federated setup where the graph is partitioned into a few large subgraphs, each stored on a different client device. By leveraging the unique features of RETEXO, it is expected that GNNs can be trained effectively and efficiently in a variety of distributed settings, beyond the fully-distributed setup. For instance, efficiently training GNNs in the federated setup, where the graph is split into few large subgraphs with one subgraph stored per client, is being studied extensively [22]. RETEXOGNNs can be cheaper to train than their corresponding GNNs in the federated setup as well due to much fewer rounds. of client-to-client communication.

The rest of this Thesis is organised as follows:

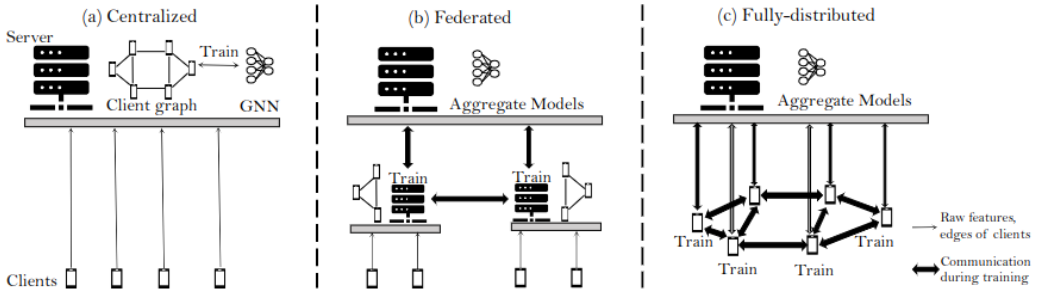

FIGURE 1.1: Training GNNs for supervised node classification across three different setups—centralized, federated and fully-distributed.

- In Background and Problem we discuss different message passing mechanism of GNNs and node classification problem on fully-distributed graphs with it prior setups.
- In GNNs for Fully-Distributed Setup we discuss a possible way to use existing GNNs in the full-distributed setup and problems associated with it.
- The Retexo GNN Architecture talks about the retexo architecture and training procedure.
- Finally Evaluation evaluates the performance of the framework for large homophilous and heterophilous datasets.

Chapter 2

Background and Problem

Consider an graph $\mathcal{G} : (\mathcal{V}, \mathcal{E})$ which has nodes $\mathcal{V}$ and edges $\mathcal{E}$. Each node has attribute or feature vector of I dimensions, given by the node-feature map $\mathbf{F} : \mathcal{V} \rightarrow \mathbb{R}^I$, and its neighbors are denoted as the set $\mathcal{N}(v)$.

2.1 Message Passing Graph Neural Networks

A GNN model $(\mathbf{M} : (v, \mathcal{G}, \mathbf{F})) \rightarrow \mathbb{R}^O)$ takes a node, all features of all nodes, and the graph as an input. It generates the node embeddings for a specified node, with an emphasis on message-passing graph neural networks (GNNs) that are commonly utilized in practical applications. In this approach, each node acquires representations from its m-hop neighbors and combines them with its own representation in each of the m rounds, where m is an integer value between 1 and K. The architecture specifies K as a parameter between 2 and 5. Consequently, the resulting representation of the specified node v after m rounds of message passing can be expressed as:

$$\mathbf{Q}_v^m = \mathbf{f}_\theta^{m-1}\left(\text{COMBINE}\left(\mathbf{Q}_v^{m-1}, \text{AGGR}_{u\in\mathcal{N}(v)}^{m-1}\left(\mathbf{Q}_u^{m-1}\right)\right)\right)$$

Here, $\mathbf{f}_\theta^{m-1}$ is a non-linear function with learnable parameters. COMBINE is usually the concatenate operation, and AGGR represents aggregation functions that may have learnable parameters. AGGR functions vary with different GNNs. Lets say, AGGR is the mean operator in GCNs [10] or a max-pooling layer in GraphSage [7]. In essence, to get the m^{th} representation of a node, a non-linear transformation over an aggregate of the $m-1^{th}$ representations of it neighbors are used.

The goal of training a GNN is to find the values of $\mathbf{f}_\theta^{m-1}$ and AGGR^{m-1} functions $\forall m \in \{1 \cdots K\}$. These parameters are optimized for accuracy on upstream tasks such as supervised node classification, which is our focus as well.

2.2 Prior Setups

Previous studies have explored two distinct approaches for supervised node classification (see Figure 1.1, (a)). The centralized setup, which is the most commonly adopted, involves processing the entire graph, including its node attributes and structure, at a single central server that is trusted and responsible for the costs associated with training the model. By contrast, a more recent approach known as the federated setup for graph learning involves a central server that does not have access to the full graph (see Figure 1.1, (b)). Instead, the graph is divided into sub-graphs and distributed among a limited number of clients, usually ranging between 5 and 20, with each client entrusted with storing a large sub-graph. The central server and the clients collaborate to learn a GNN, with the clients ensuring the privacy of the raw sub-graphs by not disclosing them to the server during the learning process. This setup is particularly relevant when clients belong to different organizations, such as banks, drug discovery companies, or hospitals, or when clients are servers located across geopolitical boundaries, where privacy laws and regulations prohibit sharing of raw data.

Compared to the centralized setup, training in the federated setup incurs higher costs. This is because in every round of training, clients must communicate with each other to exchange intermediate representations and gradients through cross-client edges, as well as with the central server to share intermediate models. Despite this, the clients are still regarded as trusted "centralized" servers with large sub-graphs, and the communication costs, particularly those between clients, can be minimized by optimizing the use of their respective sub-graphs. Additionally, federated graph learning libraries may also disregard cross-client edges to further reduce costs [19, 8].

In our setup, we do not treat labels as private information. However, our proposed solution can be adjusted to accommodate a setup in which clients do not disclose their labels to the server with minor modifications. In this modified setup, clients would only share intermediate gradients and embeddings, not raw features, in line with the protocols of federated learning. We assume that all parties involved are trustworthy and will adhere to the established protocol, including both the server and the clients.

2.3 Node Classification on Fully-Distributed Graphs

Our proposed approach is distinct from prior setups in that it employs a fully-distributed setup. This setup pushes the federated setup to an extreme, where clients are only allowed to hold data for a single node. Specifically, a client holds the data for a node, including its attributes and edges, while a server is responsible for identifying the clients in the graph. We focus on the supervised node classification problem, where the server possesses labels for only a small fraction of the nodes, and the objective is to develop a model capable of classifying the remaining nodes into pre-determined categories. This classification can be applied to other nodes in the same graph (known as the *transductive setting*) or to an unseen graph (known as the *inductive setting*), which is particularly important for evolving graphs.

In our setup, we do not treat labels as private information. However, our proposed solution can be adjusted to accommodate a setup in which clients do not disclose their labels to the server with minor modifications. In this modified setup, clients would only share intermediate gradients and embeddings, not raw features, in line with the protocols of federated learning. We assume that all parties involved are trustworthy and will adhere to the established protocol, including both the server and the clients. Figure 1.1(c) illustrates this setup.

2.4 Concrete Application

The fully-distributed setup is widely used in various applications, such as mobile sensing-based proximity networks and distributed social networks. For instance, proximity networks established by users' mobile devices through sensors like Bluetooth are a common example. These networks capture users' short-term and recurrent physical interactions, making them useful for modeling social influence propagation, tracking location histories, and studying infectious diseases [16, 13, 9].

Supervised node classification on proximity networks has numerous potential applications, including health monitoring, personalized recommendations, and sentiment analysis. For example, using GNNs for supervised node classification on proximity networks has led to excellent results in predicting influenza symptoms [4]. However, users may be hesitant to disclose their sensitive attributes and edges in the proximity network to external centralized services. Therefore, enabling GNNs on fully-distributed proximity networks can help overcome this challenge and enable such applications to proceed. It's worth noting that clients in a fully-distributed network may have limited bandwidth, and the communication networks between clients may be asynchronous and unreliable. Given these conditions, there is definitely the need to train GNNs efficiently in such setups.

Chapter 3

GNNs for Fully-Distributed Setup

The proposed setup poses a challenge for existing GNNs because the message-passing mechanism is integrated into the end-to-end training process. To demonstrate this, we provide an example using a 2-layer GCN. Message-passing graph neural networks, such as the GCN, commonly use the AGGR operation as a mean function. In other words, a node's representation is computed as an average of its neighboring nodes' representations. Specifically, for a GCN:

$$\mathbf{Q}_v^m = \theta^{m-1} \cdot \text{ Concat } \left(\mathbf{Q}_v^{m-1}, \text{Mean}_{u \in \mathcal{N}(v)} \left(\mathbf{Q}_u^{m-1}\right)\right)$$

$\mathbf{Q}^0$ is the input node features with dimension I, and θ^{m-1} are the weight parameters with hidden size H.Therefore, a GCN can be parameterized by its set of weight layers $\left\{\theta^0, \cdots, \theta^{m-1}\right\}$. A two layer GCN can be simply represented as $\left\{\theta_{IH}^0, \theta_{HO}^1\right\}$.θ_{IH}^0 represents the first layer which transforms the input node features with dimension I to the hidden dimension of size H.θ_{HO}^1 represents the second layer which transforms the hidden features with dimension H to output embedding with dimension O.

3.1 Baseline: Fully Distribued GCNs

In a fully distributed setup, training a GCN without modifications leads to high communication costs. The training process consists of R rounds, where each round starts with the server sending an aggregated model to the train-clients. The train-clients then perform a forward and a backward message-passing round, using clients in their 2-hop neighborhood.

During the forward pass in a fully distributed setup, the train-client will initiate the message-passing by sending the first layer of the GCN to its neighbors and requesting their raw features . The 1-hop neighbors will then request raw features from their neighbors and propagate the

received raw features along with their own raw features to their neighbors . After receiving the raw features from their neighbors, the 1-hop neighbors will use their own raw features and the first layer of the GCN to compute hidden representations which they send back to the train-client. Finally, the train-client will use the hidden and raw representations of its neighbors to compute the output embedding. This process is necessary for every training round of the GCN and may result in high communication costs.

To perform the backward pass in a fully distributed setup for a GCN, another round of message-passing is required. This is because some weights are shared by multiple nodes. The train-client needs to request certain factors from its neighbors to compute the gradient, and the neighbors may require some factors from their neighbors as well. It is worth noting that in the process described above, clients need to send layer parameters to their neighbors. This is necessary because the server does not have information about the neighbors of any client.

3.2 Problems with Distributed GCN

The GCN involves multiple rounds of communication among the clients in K-hop neighborhoods during each training round. This results in *high data transfer volumes and latency.* For example, the clients in proximity networks may be mobile phones that operate on low-bandwidth networks.

Enabling the backward pass in a fully-distributed setup is comparable to redesigning the automatic gradient computation mechanism used by popular machine learning libraries. This is a challenging task that requires *high availability* from the clients. In proximity networks, it is often difficult to ensure that a node's neighbors are available at the same time for both forward and backward passes. This challenge is even more significant when 2-hop and higher order neighbors are required. Even if everyone is available for one round, ensuring availability for all training rounds is not practical.

In a 2-layer GCN, if the degree of a train-client is d, it needs to send the parameters of the first layer to its d neighbors. The size of the first layer will be $2IH$, and the number of parameters sent by the train-clients will be $2IH \cdot d$. The clients from the 1-hop neighborhood would receive the layer, and the total number of parameters received by such clients would be $2IH$. During R rounds of training, the train-clients transfer $2IHdR \times 32$ bytes of data just for sharing the models with their neighbors. Furthermore, each neighbor of the train-clients transfers $2IHR \times 32$ bytes of data. The hidden representations and factors required for gradients will be $O((I + H) \cdot d)$ and $O(I \cdot d)$ bytes, respectively.

Consider a 2-layer GCN with $I = 1000$ and $H = 256$, and assume there is a train-client with $d = 10$ neighbors. During one training round's forward pass, the train-client would transfer $2 \times 1000 \times 256 \times 10 \times 32 = 160$ Mb of data. If this process is repeated for $R = 400$ rounds,

the total data transferred by the train-client just for message-passing during the forward pass would be 64 Gb. In addition, the train-client's neighbors would transfer at least 6.4 Gb. These communication costs increase linearly with the degree of the train-client, the input size, the size of additional pooling layers, and the number of layers K for each train-client and their neighbors.

Chapter 4

The Retexo GNN Architecture

Our research introduces a new graph learning approach named RETEXO, which requires significantly fewer rounds of message-passing for training compared to traditional GNNs. RETEXO is essentially a modified version of GNNs that preserves the AGGR and COMBINE functions, ensuring accuracy in homophilous, heterophilous [23], and inductive settings [7]. In other words, RETEXO can be considered a generic transformation of traditional GNNs into their RETEXO equivalents.

In a traditional GNN architecture, the graph edge structure is encoded in a layer, which leads to simultaneous optimization of all layer parameters in each training round. However, in a fully-distributed setup, this can be costly due to the need to synchronize parameter state between neighbors in a training round. To address this, our framework RETEXO decouples the edge structure from layer training and postpones message-passing until after training one layer. For example, if a GCN has 2 layers, RETEXO will break down the layers and train them sequentially, with one layer's output logits sent to neighbors in one message-passing round after its training is complete. The nodes will then use the aggregated logits as features to train the next layer. The AGGR and COMBINE functions are preserved in RETEXO, ensuring accuracy in homophilous, heterophilous, and inductive settings. See Figure 4.1for an illustration.

RETEXO is a GNN architecture that requires much fewer message-passing rounds for training compared to traditional GNNs. In a traditional GNN, the edge structure of the graph is encoded as a layer, and all layer parameters are optimized simultaneously in each training round, which is expensive in a fully-distributed setup. However, RETEXO decouples the edge structure from layer training and delays message-passing until after the training of one layer is complete. For instance, in a GCN with 2 layers, RETEXO will train the layers sequentially, and after training each layer, its output logits are sent between neighbors in one message-passing round. The number of message-passing rounds in a RETEXO GNN equals the number of layers in the original GNN, but it is independent of the number of training rounds, drastically reducing

the total number of message-passing rounds. Furthermore, RETEXO naturally extends to the federated setup and remains compatible with training improvements developed for the federated learning setup. Examples of such improvements include federated optimization [17], handling for asynchronous real-world deployments [2, 15], defenses against data poisoning, and training for better robustness [12].

4.1 Architecture of RETEXOGNNs

RETEXOGNN is a modified version of a K-layer base GNN, which consists of $K+1$ independent trainable models. The first model in RETEXOGNN is an MLP that learns embeddings from the node features, and there is no AGGR before this MLP since it operates on raw features. After the first MLP, there are K additional MLPs that include the AGGR and COMBINE layers of the base GNN (Figure 4.1, right side). The intermediate embeddings in RETEXOGNN, which are the logits after each model, are expressed as:

$$
\begin{aligned}
\mathbf{Q}_v^1 &= \mathbf{f}_\theta^0\left(Q_v^0\right) \\
\mathbf{Q}_v^m &= \mathbf{f}_\theta^{m-1}\left(\text{COMBINE}\left(\mathbf{Q}_v^{m-1}, \text{AGGR}_{u\in\mathcal{N}(v)}^{m-1}\left(\mathbf{Q}_u^{m-1}\right)\right)\right) \\
\mathbf{f}_\theta^0 &= \text{MLP}_0 = \left(\theta_{IH}^0, \theta_{HO}^0\right) \\
\mathbf{f}_\theta^{m-1} &= \text{MLP}_{m-1} = \left(\theta_{OH}^{m-1}, \theta_{HO}^{m-1}\right), m \in \{2, \ldots, K+1\}
\end{aligned}
$$

In $(\theta_{OH}^{m-1}, \theta_{HO}^{m-1})$, the θ_{OH}^{m-1} converts the aggregated embeddings to a representation of size H and the θ_{HO}^{m-1} converts that into the output. In our running example, the 2-layer RETEXO-GCN can be represented by $\{MLP_0 : (\theta_{IH}^0, \theta_{HO}^0), MLP_1 : (\theta_{OH}^1, \theta_{HO}^1), MLP_2 : (\theta_{OH}^2, \theta_{HO}^2)\}$.

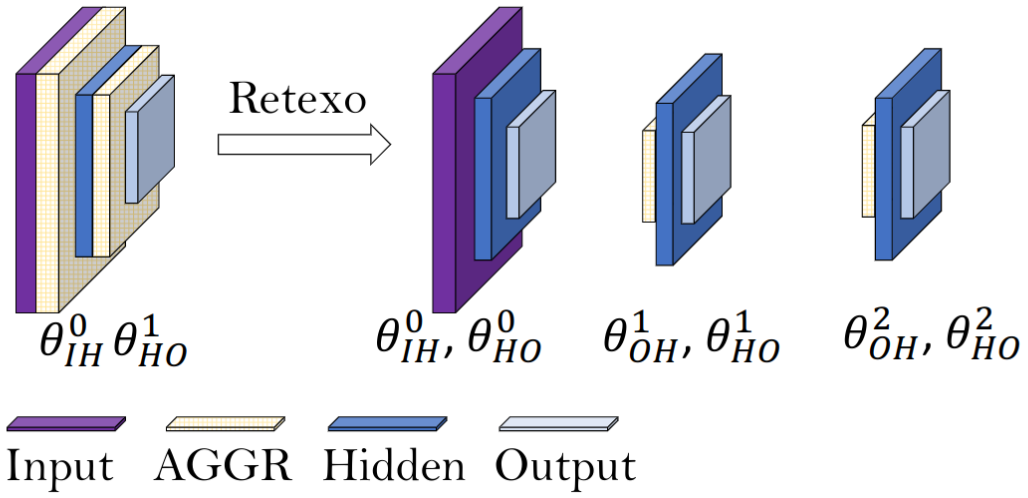

FIGURE 4.1: Base GNN to its RETEXOGNN

4.2 Training a RETEXOGNN

The training of each MLP in RETEXOGNN follows traditional federated learning approaches. Before training each MLP (except the first one), a message-passing round is conducted to obtain embeddings from the node's neighbors. The training protocol is briefly outlined in Algorithm 1, with detailed information on the federated learning and message passing protocols provided in Appendix A.

4.2.1 Federated Learning

The training process for the RETEXOGNN involves the server and clients working together to train K + 1 models sequentially. Training is done in rounds, with the server selecting a subset of train-clients for each round and assigning them training/validation tasks. During each training task, the server shares the model parameters with the selected clients who perform a forward and backward pass on the model using their local data. The clients then return the updated model to the server. In each validation task, the server only asks the clients to report their accuracy. The AGGR layer uses the embeddings collected from its neighbors in the previous message-passing round. Each client has only one input (feature vector and label) and performs a single forward and backward pass per training round, making it similar to the standard FedSGD training protocol. In the federated setup with multiple nodes per client, the FedAVG protocol can be used, which involves multiple forward and backward passes carried out locally on the client [14]. Full details of the federated learning and message passing protocols are provided in Appendix A.

4.2.2 Message-Passing

Before training the m^{th} MLP, the server initializes a message-passing round by sending all the clients the $(m-1)$ trained MLP parameters, i.e., it syncs the latest model with the clients. The clients then compute embeddings from it and share them with their neighbors. For example, consider a RETEXOGNN with three MLP and say the server is training its third MLP. At this point, the server sends the second model to all clients. So, now all clients will have the first MLP (MLP_0), their own first embeddings output by their first MLP (Q_v^1), the first embeddings obtained from their neighbors (Q_u^1, $u \in \mathcal{N}(v)$) from the previous message passing round, and the second MLP (MLP_1). The clients locally compute Q_v^2 using this information. The clients then share Q_v^2 with their neighbors to complete the current message-passing round.

In the context of communication networks, RETEXO facilitates asynchronous communication by allowing the server to initiate a message-passing round and allowing the computation to

Algorithm 1 Outline of training a RETEXOGNN

Input: $\mathcal{V}$, Train-val-test nodes: $\mathcal{V}^{tr}$-$\mathcal{V}^{val}$-$\mathcal{V}^{te}$, features: $\mathbf{F}$, labels: $\mathbf{L}$, model: $\mathbf{f}_\theta$, optimizer: Opt, max hops: K, # training rounds: R, message-passing timeout: $timeout$
// initialize server and clients
Server: $\{\mathcal{V}, \mathcal{V}^{tr}, \mathcal{V}^{val}, \mathcal{V}^{test}, \mathbf{f}_\theta, K, R\}$
Clients: $[\{v, nei{:}\mathcal{N}(v), feat{:}\mathbf{F}[v], label{:}\mathbf{L}[v], Opt\}$ $\forall v \in [|\mathcal{V}|]]$
for $m = 0$ **to** $m = K$ **do**
  if $m > 0$ **then**
    // server syncs previous trained model with all nodes and starts m^{th} message-passing round
    Server.initMessagePassing($\mathbf{f}_\theta^{m-1}$, m)
    // Clients share embeddings with their neighbors
    AsyncMessagePassing(Clients, $timeout$, m)
  end if
  $\mathbf{f}_\theta^m$ = FederatedLearning(Server, Clients, m, R)
end for
Output: Trained RETEXOGNN $\mathbf{f}_\theta$

proceed at its own pace. When it comes to proximity networks, clients have the option to share information with nearby devices using protocols such as Nearby Share or Airdrop [5, 1]. On the other hand, in distributed social networks, clients are not bound by physical proximity or the requirement of public IP addresses. Instead, WebRTC can be employed as a viable solution for communication [20].

In highly asynchronous networks, it can take anywhere from a few seconds to a few hours for a single successful message-passing round, including synchronization, before the training procedure can commence. This highlights the significance of minimizing the number of message-passing rounds. It is important to note that, unlike in a GNN (Graph Neural Network), training RETEXOGNN requires clients to share only the embeddings once per message-passing round, which is feasible even in low-quality mobile network environments. Moreover, limiting the number of message-passing rounds to K enables the server to synchronize the model with all clients after completing the training process.

In a message-passing round of RETEXOGNN, it is not necessary for all the neighbors of a training client to be present or accessible. The client can aggregate the received embeddings

from the nodes that are available.

Chapter 5

Evaluation

We evaluate RETEXOGNNs on 2 aspects.

Performance. What accuracy do RETEXOGNNs achieve in the transductive, inductive, and heterophilous settings?

Communication Efficiency. How do RETEXOGNNs communication costs compare to those of the baseline GNNs in the fully distributed setup?

In our evaluation of RETEXOGNNs using various real-world graph datasets, we assess their performance in transductive, heterophilous, and inductive scenarios. We observe that the classification accuracy of RETEXOGNNs is, on average, within a 1.2% margin compared to their corresponding GNNs in the configurations we evaluated. Additionally, when considering the implementation of GNNs and RETEXOGNNs for the fully-federated setup, we measure the amount of data transferred per client for model training.Remarkably, in RETEXOGNNs, the total data transferred per client through client-to-client communication channels is up to 6 orders of magnitude ($10^6\times$) lower compared to base GNNs. Furthermore, when considering both client-to-client and client-to-server channels, the total data transferred per client for training RETEXOGNNs is up to $20\times$ lower compared to base GNNs.

5.1 Experimental Setup

We examine three popular traditional GNN models, namely GCN, GraphSAGE, and GAT, which are widely recognized for their strong performance across various scenarios. In the case of GraphSAGE, we utilize the max-pooling layer for aggregation purposes.

5.1.1 Datasets

In the evaluation of the transductive setting, we assess the performance on seven datasets: Cora, Citeseer, PubMed, Facebook, LastFMAsia, Wiki-cooc, and Roman-empire. The first five datasets are commonly used for the homophilous setting, while the latter two have recently been introduced for the heterophilous setting. For training and validation, we utilize 5% to 10% of the nodes. In smaller datasets, we employ 10% to ensure an adequate number of nodes for training, while for larger datasets, we use 5%.

In the inductive setting, we construct evolving graphs following prior work, based on the homophilous-transductive datasets. For both the homophilous-transductive and inductive settings, we utilize 2-layer GNNs. On the other hand, for the heterophilous-transductive setting, we employ 5-layer GNNs, as recommended in the literature. Subsequently, we construct the corresponding RETEXOGNNs based on these GNN models. For more comprehensive information regarding the datasets, data splits, and network architectures, please refer to the Appendix B.

5.1.2 Implementation and Training

To begin with, we initially developed the baseline GNN models and their corresponding RETEXOGNN variants within a centralized server environment. Subsequently, we proceeded to implement RETEXOGNNs using the FederatedScope library to ensure that our centralized implementations seamlessly translated to the fully-distributed setup.

Our implementations were carefully designed to ensure that all operations performed in the centralized setup could be directly extended to the fully-distributed setup without any modifications. In both setups, we exclusively utilized the SGD optimizer with momentum and weight decay. This choice was made due to the unpredictable outcomes observed with other optimizers when there is only one node per client. Furthermore, dropout and batch normalization techniques were not employed for similar reasons.

By adhering to these considerations, we aimed to maintain consistency and ensure the reliability of our experiments across both the centralized and fully-distributed setups. To assess the communication costs, we did not execute baseline GNNs and RETEXOGNNs on an actual distributed network. Instead, we relied on simulations within the centralized setup to measure communication efficiency, specifically the data transferred between clients. This approach allowed us to emulate practical implementations of federated learning protocols. In our simulations, we diverged from the traditional epoch-based training and adopted a round-based [2] approach to better align with federated learning practices. Within each round, we randomly sampled a batch of training nodes and validation nodes, totaling up to 1024 nodes. This sampling was

TABLE 5.1: Micro-F1 scores for node classification in the transductive setting.

Method	Cora	Citeseer	Pubmed	Facebook	LastFMAsia	Wiki-cooc	Roman-empire
MLP	0.644 ± 0.011	0.652 ± 0.007	0.779 ± 0.006	0.710 ± 0.007	0.645 ± 0.007	0.895 ± 0.005	0.653 ± 0.004
GCN	0.814 ± 0.014	0.710 ± 0.007	0.825 ± 0.003	0.871 ± 0.002	0.804 ± 0.007	0.918 ± 0.005	0.752 ± 0.004
RETEXOGCN	0.763 ± 0.011	0.702 ± 0.011	0.801 ± 0.008	0.816 ± 0.010	0.756 ± 0.010	0.940 ± 0.004	0.804 ± 0.005
GraphSAGE	0.805 ± 0.010	0.699 ± 0.006	0.815 ± 0.004	0.894 ± 0.005	0.815 ± 0.005	0.936 ± 0.007	0.815 ± 0.005
RETEXOSAGE	0.785 ± 0.010	0.705 ± 0.010	0.806 ± 0.010	0.843 ± 0.009	0.788 ± 0.010	0.9616 ± 0.010	0.835 ± 0.004
GAT	0.813 ± 0.014	0.707 ± 0.008	0.815 ± 0.003	0.877 ± 0.006	0.808 ± 0.009	0.976 ± 0.009	0.800 ± 0.009
RETEXOGAT	0.802 ± 0.013	0.711 ± 0.006	0.804 ± 0.011	0.851 ± 0.006	0.807 ± 0.006	0.972 ± 0.004	0.816 ± 0.005

independent of previous rounds. In the transductive and inductive settings, we fixed the total number of rounds at 400, while in the heterophilous setting, it was set to 1000.

For the aggregation process, we limited the number of sampled neighbors considered to a maximum of 25 across each hop. This constraint ensured a manageable and consistent aggregation size throughout the experiments.

We provide accuracy measurements for all models based on their best hyperparameter settings, which were determined using grid search (refer to Appendix B for more details). Additionally, we present validation accuracies obtained through early-stopping techniques. This analysis demonstrates how employing early-stopping can effectively reduce communication costs, albeit with a slight decrease in accuracy.

5.2 Results

5.2.1 Node-Classification Performance

We present the micro-F1 scores in Tables 5.1 and 5.2, focusing on the transductive and inductive settings. Across all evaluated configurations, RETEXOGNNs exhibit a performance within a 1.20% margin compared to their respective GNNs on average. Among the RETEXO models, RETEXOGAT consistently achieves the best performance across most configurations, occasionally outperforming GAT in the inductive setting by up to 3.3%. In the heterophilous setting, RETEXOGNNs demonstrate superior performance compared to their corresponding GNNs, achieving up to a 5% increase in accuracy.

However, it is worth noting that RETEXOGCN is the least performing model, exhibiting 5% lower accuracy than GCN in three homophilous-transductive datasets. This discrepancy can be attributed to the fact that RETEXOGNNs aggregate over the embeddings of the first MLP, unlike GNNs, which aggregate directly over the raw features. Therefore, it is reasonable to expect instances where RETEXOGNNs may exhibit lower performance compared to GNNs due to this difference in aggregation approach.

TABLE 5.2: Micro-F1 scores for node classification in the inductive setting.

Method	Cora	Citeseer	Pubmed	Facebook	LastFMAsia
MLP	0.593 ± 0.012	0.623 ± 0.020	0.769 ± 0.031	0.708 ± 0.005	0.603 ± 0.010
GCN	0.747 ± 0.016	0.689 ± 0.018	0.828 ± 0.009	0.850 ± 0.007	0.757 ± 0.005
RETEXOGCN	0.670 ± 0.021	0.673 ± 0.015	0.807 ± 0.012	0.810 ± 0.008	0.712 ± 0.013
GraphSAGE	0.709 ± 0.024	0.684 ± 0.020	0.816 ± 0.007	0.876 ± 0.006	0.760 ± 0.011
RETEXOSAGE	0.722 ± 0.025	0.664 ± 0.027	0.808 ± 0.010	0.836 ± 0.007	0.741 ± 0.010
GAT	0.769 ± 0.015	0.685 ± 0.017	0.810 ± 0.008	0.827 ± 0.019	0.743 ± 0.014
RETEXOGAT	0.743 ± 0.030	0.683 ± 0.018	0.808 ± 0.011	0.860 ± 0.006	0.774 ± 0.012

5.2.2 Communication-efficiency

Our experiments focus on homophilous-transductive datasets, where we measure the data transferred between client-to-client and client-to-server communication channels in megabytes (MB). In the initial experiment, we use the same configurations that were evaluated for accuracy.

Across all evaluated configurations (see Figure 5.1), we find that RETEXOGNNs significantly reduce the total data transferred on client-to-client communication channels compared to vanilla GNNs, with reductions ranging from 4 to 7 orders of magnitude. To illustrate this, let's consider the Cora dataset and compare GraphSAGE with RETEXOSAGE. If RETEXOSAGE performs 2% worse than GraphSAGE, the total data transferred on client-to-client channels for training GraphSAGE is approximately 7.710^6 MB, while for RETEXOSAGE, it is only 1.13 MB, representing a reduction of $10^6\times$.

Furthermore, when training RETEXOGNNs, the data transferred per client is as low as 0.018 MB across 2 message-passing rounds. In contrast, for example, GCN requires a client to transfer up to $43,763$ MB in the worst-case scenario across 400 rounds, resulting in savings of several orders of magnitude ($> 10^6\times$). Similar reductions in data transfer are observed across all other evaluated configurations. Overall, these results highlight the substantial reduction in data transfer achieved by RETEXOGNNs compared to vanilla GNNs in various configurations.

The majority of data transfer costs for RETEXOGNNs primarily occur on the client-to-server channels, which are inevitable in the federated learning setup. These costs are common to both baseline GNNs and RETEXOGNNs, and they exhibit similar patterns. We present the total data transferred per client on both channels.

Comparing GraphSAGE with RETEXOSAGE, when considering the total data transferred by all clients on both channels, GraphSAGE transfers approximately 10^7 MB, whereas RETEXOSAGE transfers around 1.1×10^6 MB, resulting in a reduction of about $10\times$. In RETEXOSAGE, the data transferred is almost uniform for all train clients, averaging about $2,000$ MB, while for the remaining clients, it is close to 2 MB. On the other hand, GraphSAGE exhibits a skewed distribution of data transfer across clients, with the most affected ones transferring up to $80,000$ MB. Similar trends are observed across the remaining datasets.

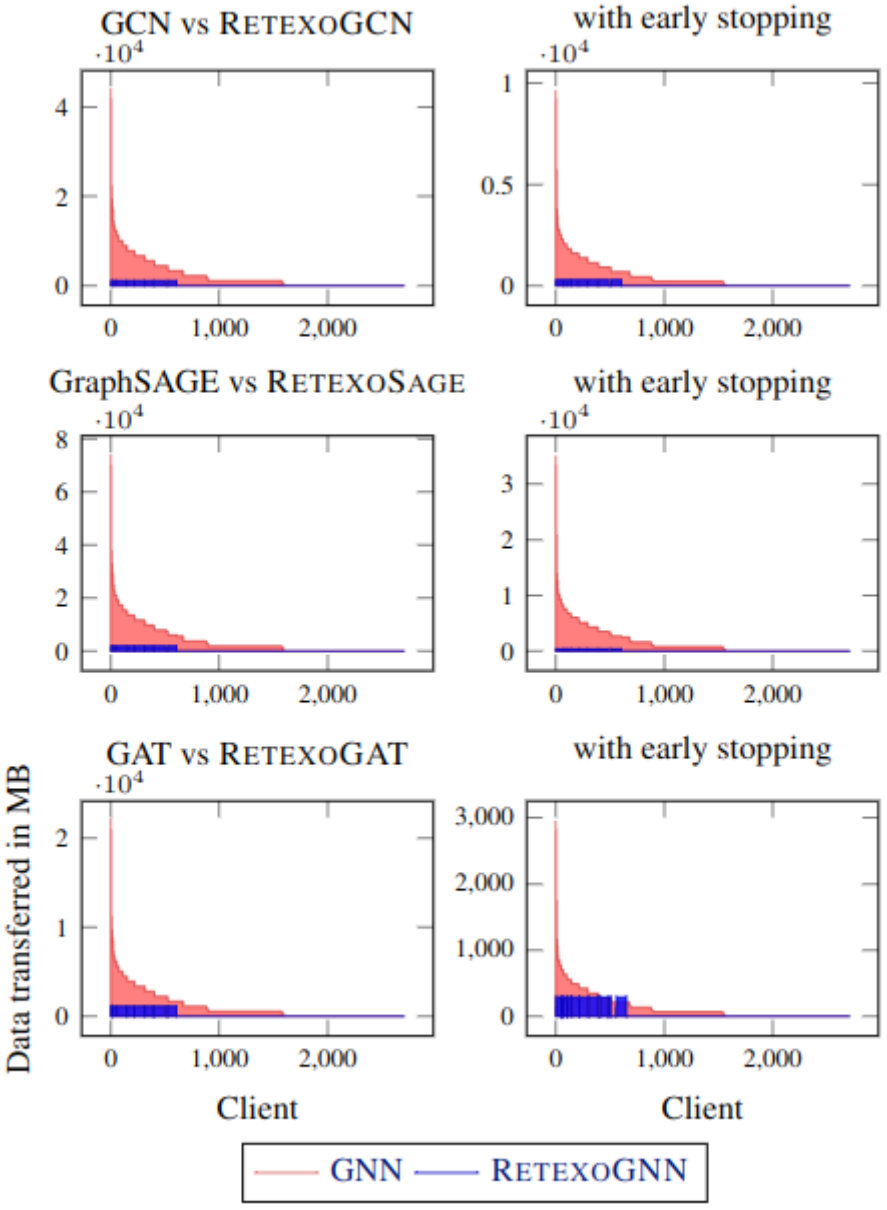

FIGURE 5.1: Comparison of data volume transferred for Cora, 400 rounds on the left and with early stopping on the right

These findings demonstrate that while the client-to-server data transfer costs are unavoidable and similar for both GNNs and RETEXOGNNs, RETEXOGNNs consistently achieve lower total data transfer per client on both channels compared to baseline GNN.

In order to assess the potential for further reducing communication costs in GNNs, we introduce early-stopping in the transductive setting while keeping the other hyperparameters unchanged. Since it is challenging to identify a common criterion for early-stopping across all evaluated configurations, we manually set the patience value to 30, which is significantly smaller than the total number of rounds (400) and yielded the best results for GNNs in most configurations.

Figure illustrates the results of these experiments. Let's consider the example of GraphSAGE versus RETEXOSAGE. On the client-to-client channels, GraphSAGE transfers a total of approximately 3.3×10^6 MB, while RETEXOSAGE only transfers 1.13 MB, resulting in a reduction of $10^6\times$. When considering both channels together, the total data transferred for GraphSAGE and RETEXOSAGE amounts to about 4.3×10^6 MB and 2.4×10^5 MB, respectively, representing a reduction of $20\times$.

These findings demonstrate that by implementing early-stopping, further reductions in data transfer can be achieved for GNNs. RETEXOGNNs consistently outperform their baseline counterparts, leading to significantly lower total data transfer on both client-to-client and client-to-server channels. Additional details on the remaining evaluated configurations can be found in the Appendix B.

To summarize, RETEXOGNNs by design have minute communication costs, up to $10^6\times$ lower, on client-to-client channels compared to GNNs even with early-stopping. Further, as GNNs become deeper, with higher K for instance in the heterophilous setting, the difference in communication costs between GNNs and RETEXOGNNs will only increase.

Chapter 6

Conclusion

Our work introduces RE-TEXOGNNs, novel neural network architectures designed specifically for training on fully-distributed graphs. These architectures offer a practical solution by significantly reducing communication costs during the training process, making them more efficient than existing GNN architectures.

To reproduce our experiments we have released our code here [18].

Appendix A

Training RETEXOGNN

We provide more detailed algorithms for federated learning and message-passing rounds to train RETEXOGNNs in Algorithms 2 and 3 respectively.

To train each MLP in RE-TEXOGNN, we employ the FedSGD protocol. The server possesses knowledge of all the clients and determines which clients will be utilized for training, validation, and testing. Each client independently stores its own feature vector, label, and information about its neighbors. Additionally, the clients maintain a local record of their neighbors' embeddings from previous message-passing rounds. The clients also have access to the optimizer employed for training. In practice, the server can send the optimizer, initialized with a fresh internal state, to the clients prior to training a model. Throughout the training process, the internal state of each client's optimizer undergoes local changes until the training is completed.

Before training an MLP (except for the first one), the server initiates a message-passing round by sending the most recently updated model to all clients. For example, if the server intends to train MLP_m, it first distributes MLP_{m-1} to all clients and instructs them to commence the message-passing round. The clients utilize their own features, embeddings, neighbors' embeddings from previous message-passing rounds, and previously trained models to compute the current embedding Q^m. Subsequently, the client shares this embedding with its neighbors. In proximity networks, sharing occurs whenever clients are physically close to each other. In distributed social networks, sharing takes place whenever the neighbors are online and a successful connection is established.

Algorithm 2 FederatedLearning: Outline of the federated learning protocol to train an MLP in RETEXOGNN

Input: Server, Clients, message-passing round number: m, # training rounds: R
// initialize server and clients
Server: $\{\mathcal{V}, \mathcal{V}^{tr}, \mathcal{V}^{val}, \mathcal{V}^{test}, \mathbf{f}_\theta, K, R\}$
Clients: $[\{v,\ nei{:}\mathcal{N}(v),\ feat{:}\mathbf{F}[v],\ label{:}\mathbf{L}[v],\ Opt\}$ $\forall v \in [|\mathcal{V}|]]$
Server executes:
    MLP_m = Server.$\mathbf{f}_\theta^{m-1}$
    Server.initModel(MLP_m)
    for $i = 0$ to $i = R - 1$ **do**
        $\mathcal{V}_s^{tr}, \mathcal{V}_s^{val} = \text{sampleNodes}(\mathcal{V}^{tr}, \mathcal{V}^{val})$
        sendMessage($\mathcal{V}_s^{tr}$, MLP_m, "train")
        sendMessage($\mathcal{V}_s^{val}$, MLP_m, "validation")

        $\text{MLP}_m^j, client_j$ = receiveMessage()
        // On receiving all updated local models
        $\text{MLP}_m^{agg} = \frac{\sum_{j=0}^{j=|\mathcal{V}_s^{tr}|} \text{MLP}_m^j}{|\mathcal{V}_s^{tr}|}$
            $\text{MLP}_m = \text{MLP}_m^{agg}$
    end for
    Output: Trained MLP_m

Each client executes:
    $\text{MLP}_m, task$ = $receiveMessage()$
    if $task ==$ "train" **then**
        $\text{MLP}_m^{tr} = \text{trainModel}(\text{MLP}_m, feat, label, Opt)$
        sendMessage($server$, MLP_m^{tr})
    end if
    // for validation compute accuracy and send it back

Algorithm 3 AsyncMessagePassing: Outline of message-passing round to train RETEXOGNN

Input: Clients, $timeout$, message-passing round number: m
Clients: [{v, nei:$\mathcal{N}(v)$, $feat$:$\mathbf{F}[v]$, $label$:$\mathbf{L}[v]$, Opt} $\forall v \in [|\mathcal{V}|]$}]
for $client$ in Clients **do**
  // Computes m^{th} embedding
  $client.Q^m$ = client.computeEmb(m)
end for
while not $timeout$ **do**
  for $client$ in Clients **do**
    for $client\_nei$ in $client.neighbors$ **do**
      // if client & client_nei are in proximity
      if Contact($client, client\_nei$) **then**
        send and receive embeddings
        $client$.sendEmbedding($client\_nei, client.Q^m$)
      end if
    end for
  end for
end while

Appendix B

Evaluation

Here we provide additional details of our evaluation.

Table B.1: All datasets' statistics.

Dataset	Nodes	Edges	Density	Features	Classes
Cora	2,708	10,566	0.0015	1,433	7
Citeseer	3,327	9,104	0.0008	3,703	6
Pubmed	19,717	88,648	0.0002	500	3
Facebook	22,470	342,004	0.0007	128	4
LastFMAsia	7,624	55,612	0.0009	128	16
Wiki-cooc	10,000	2,243,042	0.0448	100	5
Roman-empire	22,662	32,927	0.0001	300	18

Table B.2: Best hyperparameters for node classification in the transductive setting (learning rate-hidden size-number of layers/models).

Method	Cora	Citeseer	Pubmed	Facebook	LastFMAsia	Wiki-cooc	Roman-empire
MLP	$0.1-256-2$	$0.05-128-2$	$0.1-256-2$	$0.025-128-2$	$0.025-256-2$	$0.5-512-2$	$0.5-512-2$
GCN	$0.075-128-2$	$0.05-128-2$	$0.1-256-2$	$0.025-256-2$	$0.025-256-2$	$0.1-512-2$	$0.1-512-2$
RETEXOGCN	$0.005-256-2$	$0.01-256-2$	$0.01-256-2$	$0.01-64-2$	$0.005-256-2$	$0.01-512-5$	$0.01-512-5$
GraphSAGE	$0.025-256-2$	$0.025-256-2$	$0.075-256-2$	$0.01-128-2$	$0.01-256-2$	$0.01-512-5$	$0.05-512-5$
RETEXOSAGE	$0.005-256-2$	$0.005-256-2$	$0.05-64-2$	$0.01-256-2$	$0.005-256-2$	$0.01-512-2$	$0.01-512-2$
GAT	$0.1-256-2$	$0.1-256-2$	$0.1-256-2$	$0.05-64-2$	$0.05-128-2$	$0.01-512-5$	$0.01-512-5$
RETEXOSAGE	$0.005-256-2$	$0.005-256-2$	$0.1-64-2$	$0.1-128-2$	$0.005-256-2$	$0.01-512-5$	$0.01-512-5$

Table B.3: Best hyperparameters for node classification in the inductive setting.

Method	Cora	Citeseer	Pubmed	Facebook	LastFMAsia
MLP	$0.1-256-2$	$0.05-128-2$	$0.1-64-2$	$0.025-128-2$	$0.01-128-2$
GCN	$0.05-128-2$	$0.025-256-2$	$0.1-128-2$	$0.025-128-2$	$0.01-256-2$
RETEXOGCN	$0.005-128-2$	$0.005-128-2$	$0.1-64-2$	$0.005-64-2$	$0.005-256-2$
GraphSAGE	$0.025-256-2$	$0.025-256-2$	$0.1-256-2$	$0.01-256-2$	$0.01-256-2$
RETEXOSAGE	$0.005-256-2$	$0.005-128-2$	$0.025-128-2$	$0.01-64-2$	$0.005-256-2$
GAT	$0.1-64-2$	$0.1-128-2$	$0.1-256-2$	$0.01-256-2$	$0.025-128-2$
RETEXOGAT	$0.005-128-2$	$0.005-256-2$	$0.1-64-2$	$0.01-64-2$	$0.005-128-2$

Table B.4: Total data transferred by all the nodes over client-to-client channels (in MB) for 400 rounds

Method	Cora	Citeseer	Pubmed	Facebook	LastFMAsia
GCN	4627658.72	9659665.72	3300607.15	2087862.72	1259000.01
RETEXOGCN	1.13	0.83	4.06	20.87	15.27
GraphSAGE	7760932.15	15145751.66	6630759.18	7276392.72	4388046.26
RETEXOSAGE	1.13	0.83	4.06	20.87	15.27
GAT	2322407.15	4842582.91	1661856.44	1066370.87	642969.03
RETEXOGAT	1.13	0.83	4.06	20.87	15.27

Table B.5: Total data transferred by all the nodes over client-to-client channels (in MB) with early stopping

Method	Cora	Citeseer	Pubmed	Facebook	LastFMAsia
GCN	918371.17	1734677.93	1791573.12	785299.90	206842.69
RETEXOGCN	1.13	0.83	4.06	20.87	15.27
GraphSAGE	3324337.78	4749349.10	1341075.34	2554058.92	1034681.50
RETEXOSAGE	1.13	0.83	4.06	20.87	15.27
GAT	281081.41	532270.57	744665.34	366553.73	75584.53
RETEXOGAT	1.13	0.83	4.06	20.87	15.27

Table B.6: Total data transferred by all the nodes (in MB) for 400 rounds

Method	Cora	Citeseer	Pubmed	Facebook	LastFMAsia
GCN	5887658.72	13554115.72	4105407.15	2299062.72	1492600.01
RETEXOGCN	652239.79	1978234.94	426662.56	127980.72	204704.34
GraphSAGE	10098932.15	21525026.66	8657159.18	8412392.72	5557646.26
RETEXOSAGE	1107737.30	2523817.94	1260578.82	964903.49	1056650.35
GAT	2953306.08	6790882.52	2065875.19	1173595.87	761481.53
RETEXOGAT	647851.57	1974021.95	425043.30	124736.83	178985.21

Table B.7: Total data transferred by all the nodes (in MB) with early stopping

Method	Cora	Citeseer	Pubmed	Facebook	LastFMAsia
GCN	1176671.17	2386998.30	2206045.12	862915.90	243634.69
RETEXOGCN	168741.04	325437.82	97316.56	19764.72	31344.34
GraphSAGE	4358902.78	6615287.04	1731157.34	2943138.92	1297841.50
RETEXOSAGE	243849.80	409768.94	180618.82	152839.49	288106.35
GAT	359943.77	731971.28	916373.30	402474.10	88917.19
RETEXOGAT	166479.07	325453.70	100151.30	23052.83	33045.21

TABLE B.8: Micro-F1 score with a certain fraction of edges considered for each node for RETEXOGNNs.

Method	Fraction	Cora	Citeseer	Pubmed	Facebook	LastFMAsia
RETEXOGCN	0.5	0.753 ± 0.011	0.700 ± 0.011	0.800 ± 0.009	0.807 ± 0.010	0.751 ± 0.009
	0.7	0.761 ± 0.011	0.702 ± 0.011	0.800 ± 0.008	0.813 ± 0.010	0.754 ± 0.009
	1.0	0.763 ± 0.011	0.702 ± 0.011	0.801 ± 0.008	0.816 ± 0.010	0.756 ± 0.010
RETEXOSAGE	0.5	0.779 ± 0.011	0.699 ± 0.011	0.803 ± 0.010	0.834 ± 0.008	0.784 ± 0.011
	0.7	0.786 ± 0.007	0.701 ± 0.011	0.807 ± 0.010	0.842 ± 0.009	0.788 ± 0.009
	1.0	0.785 ± 0.010	0.705 ± 0.010	0.806 ± 0.010	0.843 ± 0.009	0.788 ± 0.010
RETEXOGAT	0.5	0.788 ± 0.013	0.708 ± 0.007	0.798 ± 0.011	0.842 ± 0.008	0.799 ± 0.004
	0.7	0.800 ± 0.013	0.709 ± 0.007	0.799 ± 0.010	0.849 ± 0.005	0.806 ± 0.007
	1.0	0.802 ± 0.013	0.711 ± 0.006	0.804 ± 0.011	0.851 ± 0.006	0.807 ± 0.006

Bibliography

[1] Apple. *AirDrop, Apple*. https://support.apple.com/en-sg/HT204144.

[2] Keith Bonawitz et al. "Towards federated learning at scale: System design". In: *Proceedings of Machine Learning and Systems* (2019).

[3] Carole Cadwalladr and Emma Graham-Harrison. "Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach". In: *The Guardian* (2018). https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election.

[4] Guimin Dong et al. "Influenza-like symptom recognition using mobile sensing and graph neural networks". In: *Proceedings of the Conference on Health, Inference, and Learning*. 2021.

[5] Google. *Nearby Share, Android*. https://support.google.com/files/answer/10514188?hl=en.

[6] Xinning Gui et al. "When fitness meets social networks: Investigating fitness tracking and social practices on werun". In: *Proceedings of the 2017 CHI conference on human factors in computing systems*. 2017.

[7] Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs". In: *Advances in neural information processing systems* 30 (2017).

[8] Chaoyang He et al. "Spreadgnn: Serverless multi-task federated learning for graph neural networks". In: *arXiv preprint arXiv:2106.02743* (2021).

[9] Enrique Hernández-Orallo et al. "Evaluating how smartphone contact tracing technology can reduce the spread of infectious diseases: The case of COVID-19". In: *IEEE Access* (2020).

[10] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[11] Aashish Kolluri et al. *RETEXO: Scalable Neural Network Training over Distributed Graphs*. 2023. arXiv: 2302.13053 [cs.LG].

[12] Rui Liu and Han Yu. "Federated Graph Neural Networks: Overview, Techniques and Challenges". In: *arXiv preprint arXiv:2202.07256* (2022).

[13] Anmol Madan et al. "Social sensing: obesity, unhealthy eating and exercise in face-to-face networks". In: *Wireless Health 2010*. 2010, pp. 104–110.

[14] Brendan McMahan et al. "Communication-efficient learning of deep networks from decentralized data". In: *Artificial intelligence and statistics*. 2017.

[15] John Nguyen et al. "Federated learning with buffered asynchronous aggregation". In: *International Conference on Artificial Intelligence and Statistics*. 2022.

[16] Wei Pan, Nadav Aharony, and Alex Pentland. "Composite social network for predicting mobile apps installation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2011.

[17] Sashank Reddi et al. "Adaptive federated optimization". In: *arXiv preprint arXiv:2003.00295* (2020).

[18] *Retexognn-Prototype*. URL: `https://github.com/aashishkolluri/retexognn-prototype`.

[19] Zhen Wang et al. "FederatedScope-GNN: Towards a Unified, Comprehensive and Efficient Package for Federated Graph Learning". In: *KDD* (2022).

[20] WebRTC. *WebRTC: Real-time communication for the web*. `https://webrtc.org/`.

[21] Zonghan Wu et al. "A comprehensive survey on graph neural networks". In: *IEEE transactions on neural networks and learning systems* (2020).

[22] Longfei Zheng et al. "ASFGNN: Automated separated-federated graph neural network". In: *Peer-to-Peer Networking and Applications* (2021).

[23] Xin Zheng et al. "Graph neural networks for graphs with heterophily: A survey". In: *arXiv preprint arXiv:2202.07082* (2022).

[24] Jie Zhou et al. "Graph neural networks: A review of methods and applications". In: *AI Open* (2020).