A DISSERTATION
SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN COMPUTER SCIENCE AND ENGINEERING

Efficient Machine Learning Systems by Exploiting Data Sparsity

by

Tao Liu

March 2024

The thesis titled

Efficient Machine Learning Systems by Exploiting Data Sparsity

by

Tao Liu

is reviewed and approved by:

Chief referee

Senior Associate Professor Date

Peng Li Jan. 23, 2024

Professor Date

Anh Tuan Pham 2024/02/19

Professor Date

Mohamed Hamada 19 Feb 2024

Senior Associate Professor Date

Truong Cong Thang Feb 16, 2024

THE UNIVERSITY OF AIZU

March 2024

Contents

List of Figures

List of Tables

List of Abbreviations

CNN	Convolution Neural Network
DGL	Deep Graph Library
FGL	Federated Graph Learning
FL	Federated Learning
GCN	Graph Convolutional Network
HE	Homomorphic Encryption
LSH	Local Sensitive Hash
MAE	Masked Autoencoder
NLP	Natural Language Processing
PS	Parameter Server
ViT	Vision Transformer
WAN	Wide-area Network

Acknowledgment

During my doctoral studies, I received a great deal of assistance and support, and I would like to express my gratitude to all those who helped me during this time.

First and foremost, I would like to extend my sincere appreciation to my supervisor, Professor Peng Li, for his invaluable guidance, encouragement, and unwavering support throughout my Ph.D. journey. He has been an exceptional mentor and a source of inspiration to me. My sincere appreciation also goes to Prof. PHAM Tuan Anh, Prof. HAMADA Mohamed, and Prof. TRUONG Cong Thang for their reviews and support.

I would also like to thank the entire university staff for creating a stimulating and friendly environment that has greatly contributed to my success. I am grateful for the assistance and cooperation of the staff and students who have supported my research in various ways. Special thanks go to the SAD and ALO staff for their help with my personal and academic affairs.

I would like to acknowledge my lab members and friends at the university for their constant support and valuable advice on my research work. We shared some unforgettable experiences, such as snowboarding together, which helped us bond and learn from each other.

Finally, I owe a debt of gratitude to my family for their unconditional love, patience, and understanding during this challenging period. They have been my source of strength and motivation, and I dedicate this thesis to them.

Abstract

As machine learning technology becomes increasingly integrated into our daily lives through various applications, building efficient machine learning systems is becoming an important goal for both academia and industry for communication or computation resource savings. One promising approach is exploiting data sparsity, which refers to the fact that many real-world datasets are inherently sparse and used redundantly in machine learning processes. By exploiting this property, significant overhead savings can be achieved.

Our work focuses explicitly on exploiting data sparsity for both Euclidean data, such as images, and non-Euclidean data, such as graphs. We make three distinct contributions, which contains both training and inference processes.

The first part of our work focuses on the scene of federated graph convolutional network (GCN) training. The training process of GCN requires information exchange between connected nodes in a graph. However, when graph data is distributed among different owners, there are cross-device connections among nodes that belong to different owners, resulting in cross-device communication. Frequent communication may become a bottleneck for the whole system, and direct data interaction may lead to privacy leaks. To address this, we propose a secure federated graph learning system that encrypts transmitted data and explores redundant connections among data owners. We evaluate the contributions of neighbors to each data owner and then eliminate unnecessary transmission. We further schedule transmission flows in the network to make the training more efficient.

The second part of our work focuses on the inference process of the GCN. When we conduct inference on a large graph, the GPU may not have enough memory to load the whole data. We usually need to generate multiple mini-batch to load into the GPU one-by-one. We observe that many nodes in a graph are loaded repeatedly into the GPU due to the non-sampling strategy in the inference. This repeat loading significantly reduces inference speed. To address the issue, we propose a more efficient system that uses an adaptive sampling design to sample nodes' neighbors according to their degree. And we reuse the loaded data from the previous mini-batch. To get more data that can be used again, we reorder these inference batches based on how similar they are using a local sensitive hash (LSH)-based clustering scheme.

In addition to graph data, we also explore the sparsity of image data. Many edge devices with weak computing power collect image data that needs to be identified, but these devices may not have enough resources to conduct complex neural network identification. Therefore, the data often needs to be uploaded to a server for processing. We propose an offloading system that does not require computation on the edge device and only needs to transmit part of the image data to the server. The server can then recover the image and perform inference using a feedback-driven strategy designed to achieve content-aware transmission.

概要

機械学習技術がさまざまなアプリケーションを通じて私たちの日常生活にますます統合されるにつれて、通信や計算リソースの節約のために、効率的な機械学習システムを構築することが学術界と産業界の両方にとって重要な目標となっています。有望なアプローチの一つは、データの疎性を利用することです。これは、多くの実世界のデータセットが本質的に疎であり、機械学習プロセスで冗長に使用されているという事実を指します。この特性を利用することで、大幅なオーバーヘッド節約が実現できます。

私たちの研究は、ユークリッドデータ（例えば画像）と非ユークリッドデータ（例えばグラフ）の両方におけるデータの希薄性を活用することに明確に焦点を当てています。私たちは訓練と推論のプロセスを含む 3 つの独自の貢献をしています。

私たちの研究の最初の部分は、連合グラフ畳み込みネットワーク（GCN）トレーニングのシーンに焦点を当てています。GCN のトレーニングプロセスでは、グラフ内の接続されたノード間で情報交換が必要です。しかし、グラフデータが異なる所有者の間で分散されている場合、異なる所有者に属するノード間にデバイス間接続があり、デバイス間通信が生じます。頻繁な通信はシステム全体のボトルネックになる可能性があり、直接的なデータのやり取りはプライバシーの漏洩につながる可能性があります。これに対処するために、私たちは送信データを暗号化し、データ所有者間の冗長な接続を探求する安全な連合グラフ学習システムを提案します。私たちは、各データ所有者に対する隣接ノードの貢献度を評価し、不必要な送信を排除します。さらに、トレーニングをより効率的にするためにネットワーク内の送信フローをスケ

ジュールします。

私たちの研究の第二部では、GCN の推論プロセスに焦点を当てています。大規模なグラフに対して推論を行う際、GPU のメモリが全データをロードするには不十分な場合があります。通常、複数のミニバッチを生成し、それらを GPU に一つずつロードする必要があります。非サンプリング戦略により、グラフ内の多くのノードが推論のために GPU に繰り返しロードされることが観察されます。この繰り返しのロードは、推論速度を著しく低下させます。この問題に対処するために、私たちはノードの隣接ノードをそれらの度数に応じてサンプルする適応的サンプリング設計を使用するより効率的なシステムを提案します。そして、前のミニバッチからロードされたデータを再利用します。さらに多くのデータを再度使用できるようにするために、ローカルセンシティブハッシュ（LSH）ベースのクラスタリングスキームを使用して、これらの推論バッチをどれだけ似ているかに基づいて並べ替えます。

グラフデータに加えて、私たちは画像データのスパース性も探求しています。計算能力が弱いエッジデバイスは、識別が必要な画像データを収集しますが、これらのデバイスは複雑なニューラルネットワーク識別を行うのに十分なリソースを持っていないかもしれません。そのため、しばしばデータはサーバーへアップロードされて処理される必要があります。私たちは、エッジデバイス上での計算を必要とせず、画像データの一部だけをサーバーに送信するだけで良いオフロードシステムを提案します。その後、サーバーは画像を復元し、内容認識型トランスミッションを達成するために設計されたフィードバック駆動戦略を使用して推論を行うことができます。

Chapter 1

Introduction

Machine learning (ML) technology has integrated into our daily lives through diverse applications such as intelligent email, facial recognition, content recommendation, and more. The field of machine learning encompasses various distinct categories, and the structure of these research areas is depicted in Figure 1. The lower-level research is centered around hardware components, such as different high-speed processing chips, while the higher-level research is focused on software aspects, such as various frameworks and programming interfaces. The scope of our research is centered around the system level of machine learning. This entails studying the precise methods used to process data for computation, as well as the training and inference processes of models.

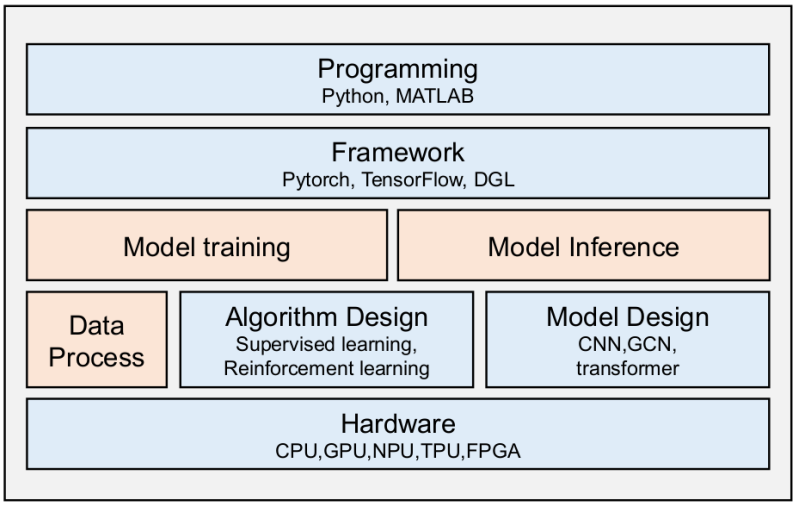


Figure 1.1: The composition of research fields in machine learning.

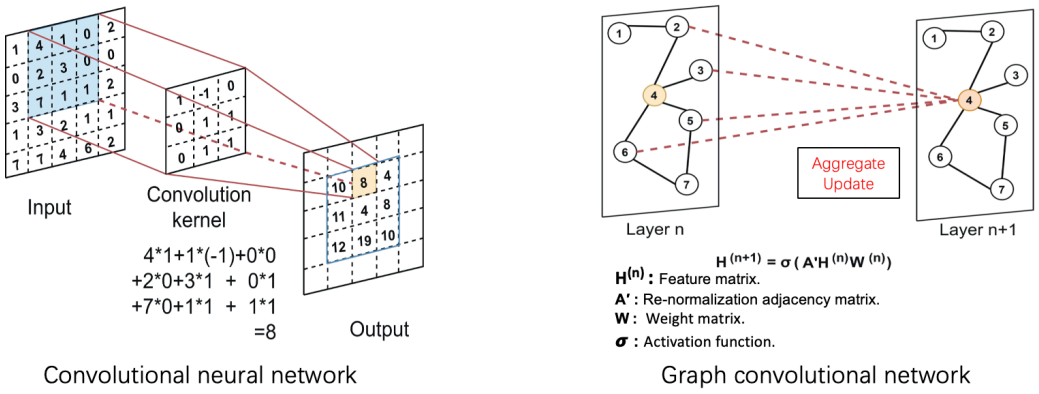


Figure 1.2: CNN vs. GCN.

Machine learning encompasses various forms of data that are utilized for distinct tasks. The data can be categorized into two main types: non-Euclidean data, which includes graphs, and Euclidean data, such as images. Figure 1.2 illustrates various models designed to handle diverse types of data. Typically, the image data is analyzed using a convolutional neural network, which employs a convolution kernel to extract the fundamental characteristics and create the abstract embeddings. The graph convolutional network (GCN) [1] aims to transform vertex feature vectors into compressed embeddings by leveraging both the graph structure and vertex features. Each vertex in the graph is associated with a feature vector. The GCN employs a stacking mechanism, where multiple layers are combined, and each layer maintains the identical structure as the original graph.

Throughout the process of inference and training, we observe that the presence of redundant content in the data can impede the overall efficiency of the system. One can investigate the data's sparsity to minimize unnecessary costs and speed up the system. Figure 1.3 illustrates the occurrence of data sparsity in both image data and graph data. The background of the image data is irrelevant in the process of image recognition; in fact, only the target area is typically necessary to achieve accurate identification results. While it is possible to incorporate additional nodes in a graph to depict the human skeleton, the existing information regarding crucial anatomical landmarks on the human body is already sufficient for certain applications. These encourage us to leverage the scarcity of data to minimize the

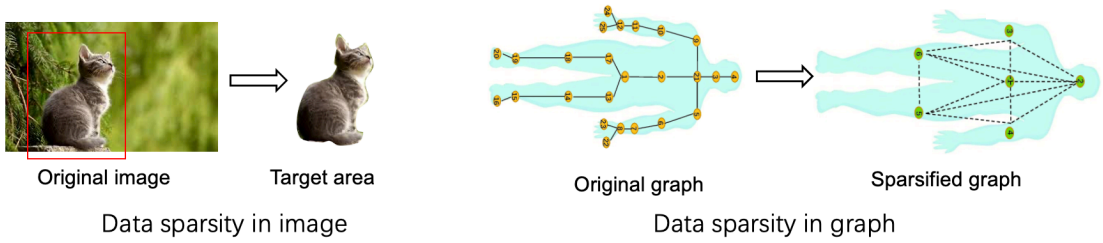


Figure 1.3: Sparsity of data.

communication and computational burden in machine learning systems.

Directly harnessing the sparsity property of data is a nontrivial task. For example, when the background of an image is intricate, it becomes challenging to accurately locate the essential target area within the image. Employing an intricate model for area extraction will result in additional overhead. In our work, we explore the data sparsity in specific training and inference scenarios to combine the unique characteristics of computation and communication.

Initially, we examine the scenario in which the graph data could be dispersed among multiple proprietors. Consider a scenario where there exist multiple hospitals alongside a central medical administration center. The global graph captures the data of patients in the city, including their attributes and interactions, over a specific time frame. More precisely, we possess two types of links: the first type is explicitly recorded within each hospital, while the second type comprises cross-hospital links that may exist but are not stored within any specific hospital. The objective of the medical administration center may be to acquire a globally influential graph mining model while avoiding the sharing of data. Federated learning facilitates collaborative training of a comprehensive model while preserving the privacy of raw data [2]. Nevertheless, federated learning on graphs encounters two primary obstacles. GCN training entails the exchange of features among adjacent graph nodes across FL servers, resulting in the compromise of privacy. Furthermore, previous studies have demonstrated that communication emerges as the primary constraint in distributed machine learning, significantly impeding the efficiency of learning tasks (Konečnỳ et al., 2016). While several techniques [3, 4]

have been suggested to enhance communication efficiency, they do not address the specific issue examined in our study. They typically focus solely on analyzing the trade-off between learning speed and energy efficiency [3, 4]. Privacy is not a matter of concern.

Besides the training of graphs, the inference process of the graph also faces a communication burden when loading graph data from the main memory to the GPU. The data loading time is even longer than the inference time itself. We further find that different inference batches contain many common nodes, and their features are repeatedly loaded into the GPU by the current systems, which leads to redundant energy consumption and time delay. In fact, this redundant loading issue is even more severe in the inference than in the training since there is no sampling operation in the inference for accuracy consideration [5]. This observation motivates us to improve the efficiency of GNN inference by reusing graph data already loaded into the GPU to avoid redundant data loading. And an accuracy-guaranteed sampling strategy is needed for the inference. Note that a similar reuse idea has been proposed for GNN training by [6]. However, the order of batches in the training is random and non-controllable, while the inference operation can be conducted periodically in an offline scenario to reorder batches. For instance, PinSage [7] utilizes Map Reduce to generate embedding in an offline process. GEM [8] detects malicious accounts with GNN daily. As a result of its lack of flexibility, the static solution in [6] cannot scale to large graphs in the inference.

Unlike process graph data, where the redundant information may come from the structure, image data usually contains the content itself. There are lots of edge devices with weak computing power that collect image data; the data may need to be identified. However, the weak device does not have enough resources to construct a complex neural network for identification. A straightforward offloading strategy sends raw data to the cloud [9, 10], which does not require computation at edge devices and achieves high inference accuracy by using powerful hardware

	Data	
	Graph	Image
Training	Explore graph sparsity in training -- with traffic throttling	
Inference	Explore graph sparsity in inference -- with adaptive sampling	Explore image sparsity in offloading -- with mask-recover scheme

Figure 1.4: Our contributions.

in the cloud. However, since raw data has a large size, this strategy has high communication costs. Some recent works propose data preprocessing techniques at edge devices to reduce communication costs. Such preprocessing techniques include DNN model splitting [11, 12], input data compression [13], and input data filtering [14]. These methods still need to be calculated on the edge device and may incur an accuracy decrease.

The main idea of this dissertation is to exploit the data sparsity for both Euclid data and non-Euclid data to propose efficient machine-learning systems to handle the above issues. We summarize our works in Fig. 1.4.Specifically, we propose a federated graph learning system with enhanced security insurance and a series of novel designs to address the communication challenge. We found that some neighbors were not necessary for the training, so we designed a traffic throttling strategy to eliminate the transmission. In order to protect vertex embeddings, we adopt a homomorphic encryption (HE)-based protocol that only transmits encrypted data.To accelerate the inference of the graph, we adopt an adaptive sampling strategy that aims to reduce the number of loaded nodes while guaranteeing accuracy. We also provide a Local Sensitive Hash (LSH) [15] based hierarchical batch clustering scheme to reorder and reuse batch nodes.We finally propose a new approach to achieve efficient offloading for image recognition on weak edge devices. Its basic idea is to let edge devices randomly sample a small portion of image patches and send them to the server, which then uses a masked autoencoder (MAE) [16] to recover the image and conduct inference.MAE was originally de-

signed for pre-training, and we exploit its powerful capabilities in image recovery for DNN offloading. Therefore, it is promising to achieve high inference accuracy with limited sampled data. Our contributions are listed as follows:

1. We explore graph sparsity in federated graph learning by proposing an efficient distributed training system. It lets each data owner evaluate others contributions and eliminates unnecessary transmission. The data flows are properly scheduled.

2. We further explore graph sparsity in the inference. We propose an efficient inference system that uses adaptive sampling neighbors to reduce the number of loaded nodes. And we reuse the previously loaded data for time savings. The LSH-based minibatch clustering method is also included to reorder the batches and reuse more data.

3. We finally explore image sparsity in the offloading of weak edge devices. We propose a novel offloading system for image classification that does not require computation on the device and only needs to transmit part of the image data to the server. The server recovers the image and then conducts inference.

4. We conduct extensive experiments to compare our methods with various baselines and verify the effectiveness of the proposed systems.

For the rest of the dissertation, the background is given in Chapter 2. We present our efficient federated graph learning systems in Chapter 3, followed by the graph inference system in Chapter 4. Chapter 5 describes our novel image offloading system. The conclusion and future work is in Chapter 6.

Chapter 2

Background

There are various machine learning models and computational paradigms involved in our work. Specifically, for our first work, we use the **federated learning** scheme to distribute train the **graph convolution network**. The **homomorphic encryption** is used to protect privacy in data exchanges among different data owners. For the second work, the **local sensitive hash** is used to quickly cluster the batches. For the third work, we recover a masked image with a **ViT**-based **Masked Autoencoder** before the inference. In this chapter, we describe the background of these machine learning models and computational paradigms to serve as the fundamental principles for the subsequent work.

2.1 Federated Learning

Federated learning (FL) has been proposed to enable collaborative training among multiple devices (data owners) without leaking any raw data. Its basic idea is to let device share model parameters instead of training data. A typical parameter synchronization scheme widely adopted by federated learning is the Parameter Server (PS) as shown in Fig. 2.1 [17]. Specifically, a federated learning system consists of a parameter server and a set of computing device. The training process contains multiple rounds. Every device uses its local dataset to train a model in each round. After local training, device send their local models to the

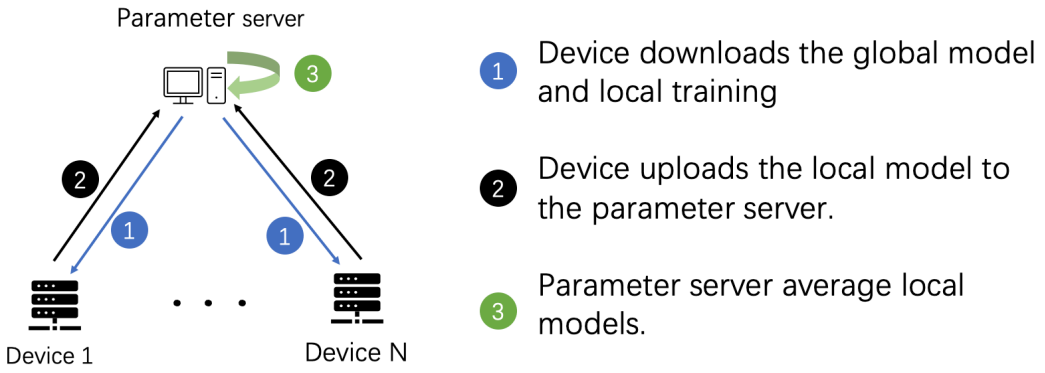


Figure 2.1: Federated learning process.

parameter server, and the parameter server then creates a new global model and distributes it for the next-round training. Despite the simplicity, PS-based federated learning suffers from the global synchronization bottleneck at the parameter server.

2.2 Graph Convolutional Network

Given a graph where each vertex is associated with a feature vector, a graph convolutional network (GCN) aims to transfer vertex feature vectors into compressed ones, also called embeddings, by exploiting graph structure and vertex features. GCN stacks multiple layers, and each layer has the same structure as the original graph. The graph convolution operation is defined as aggregating node embeddings of neighboring nodes as shown in Fig. 2.2. We use A to denote the graph's adjacent matrix and H^l to denotes the matrix of node embeddings in the l-th layer. The propagation rule of the GCN can be formally expressed by

$$z^{l+1} = \overline{A}H^l W^l, \qquad H^{l+1} = \sigma(z^{l+1}), \tag{2.1}$$

where $\overline{A} = D^{-\frac{1}{2}}(A+I)D^{-\frac{1}{2}}$, W^l denotes trainable feature weights, D is the degree matrix and σ is the activation function.

Given a set V of nodes with labels, the GCN training goal is to minimize the

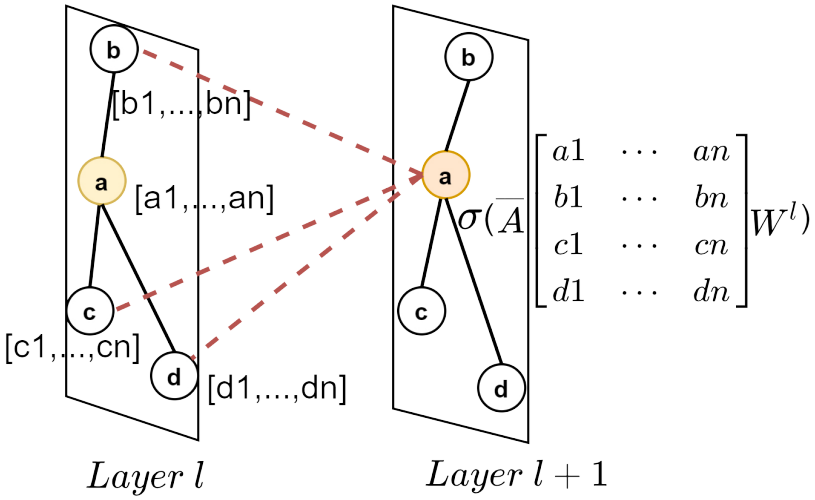


Figure 2.2: Convolution operation on the graph..

loss function:

$$\mathcal{L} = \frac{1}{|\mathcal{V}|} \sum_{j \in \mathcal{V}} \mathcal{F}(y_j, \hat{y}_j), \tag{2.2}$$

where y_j and $\hat{y}_j$ denote the true label and the predicted one, respectively. The loss function $\mathcal{F}$ usually uses cross-entropy.

2.3 Homomorphic encryption

Homomorphic encryption (HE) is designed for carrying out function computation on ciphertexts while preserving the functional characteristics of the plaintext [18]. A typical homomorphic encryption scheme can be expressed as $HE = (KeyGen, Enc, Dec, Eval)$, where the function $KeyGen$ generates a public key pk and a private key sk, Enc and Dec are the encryption function and the decryption function, respectively. The computation operations supported by homomorphic encryption are denoted by $Eval$. The characteristic of homomorphic encryption can be described as $D_{sk}(Eval(E_{pk}(a), E_{pk}(b))) = Eval(a, b)$, which indicates that by decrypting the results of operations on ciphertext, we can obtain the same results with those of original operations on the plaintext. Due to this unique characteristic, homomorphic encryption is usually used for outsourcing computation to the third-parties.

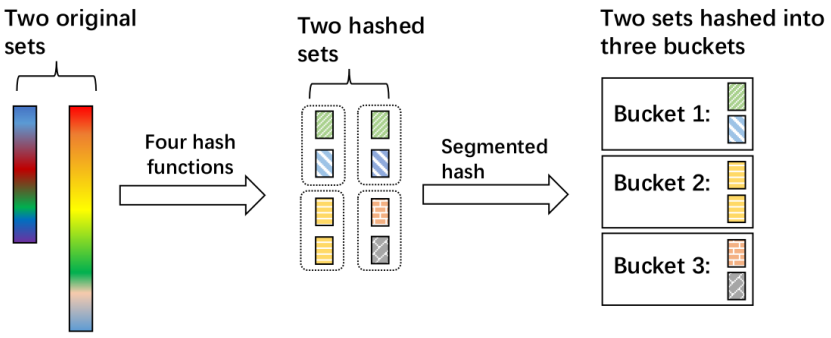


Figure 2.3: An example of the process of LSH.

For example, suppose two servers, holding a and b respectively, are untrusted with each other but still want to conduct some computation involving a and b. They can send encrypted data, i.e., $Eval(E_{pk}(a)$ and $Eval(E_{pk}(b)$, to the cloud that conducts operations and then returns results in the form of $Eval(E_{pk}(a), E_{pk}(b))$. Then, both computers can obtain the final result after decryption.

2.4 Local Sensitive Hash (LSH)

Local sensitive hash (LSH) has been proposed to deal with quick similarity queries [15]. In our case, we adopt the commonly used min-hash-based [19] LSH implementation, which measures the Jaccard similarity among sets. The Jaccard similarity of two sets w and v can be calculated as $(w \cap v)/(w \cup v)$, equal to the union of two sets that divide their intersection. This similarity calculation is suitable for our scenario since inference batches can be treated as multiple sets with various lengths (each element is the index of the node), making the Euclid distance unworkable. Instead of calculating the Jaccard similarity among pairs one by one, LSH utilizes hash functions to reduce the data size while keeping the similarity among the original data first and then maps similar sets to the same key value of the dictionary (also named bucket).

We give a toy example to express the process of min-hash-based LSH as shown in Fig. 2.3. Suppose there are two sets (w and v) with various lengths. The LSH process contains two hash periods. The first hash period is min-hash, each hash

function $f_i()$ goes through the two sets to obtain $f_i(w)$ and $f_i(v)$. Suppose there is a full set Ω that contains every element in the sets, $f_i(w)$ can be considered as the index of the first element of w in Ω after randomly disrupting w. We have

$$P(f_i(w) = f_i(v)) = Sim(w, v), \tag{2.3}$$

where $Sim(w, v)$ means the Jaccard similarity among two sets. The first hash period replaces the original set with a small number of hash values while keeping the similarities among sets. However, calculating their similarities is still time-consuming since the number of sets is still large. We need another hash process. For the second hash period, the previously obtained output will be further segmented and hashed. Specifically, every r values in every set will be formed into a band and hashed into a bucket. The first bands from the two sets are the same in the example, thus being hashed into the same bucket and becoming a similar pair. Suppose there are n sets with the average length as m, and the number of hash functions in the first period is F. Thus, the computation complexity is $O(n \cdot m \cdot F + n \cdot m/r)$, which equals $O(n)$ when ignoring the constants.

2.5 Transformer and ViT

Transformer has been proposed as a self-attention-based model that can effectively handle various learning tasks related to natural language processing (NLP) [20–23]. Given a sentence whose words can be expressed as feature embeddings $X = [x_0, x_1, ..., x_n] \in \mathbb{R}^d$, the core self-attention operation can be described as follows:

$$Q_i = x_i W^Q,\ K_i = x_i W^K,\ V_i = x_i W^V. \qquad \forall x_i \in X \tag{2.4}$$

$$A_{i,j} = \frac{Q^i K^j}{\sqrt{d}},\ x_i^{'} = (\sum_{j=0}^{n} softmax(A_{i,j}) V_j). \qquad \forall x_j \in X \tag{2.5}$$

The W^Q, W^K, $W^V \in \mathbb{R}^d$ in Equation 2.4 are trainable weights that transform X into query, key, and value matrices, respectively. $A_{i,j}$ is the normalized weight

of value for x_j. The newly generated hidden embedding $x_i^{'}$ is a weighted sum based on all values. A complete attention layer contains multiple self-attention operations (multi-heads), and a Transformer model contains multiple attention layers.

Inspired by Transformer's successes in NLP, Vision Transformer (ViT) has been proposed to deal with image data [24]. Concretely, an image is divided into different patches, where each patch's size is fixed at 16×16 since it brings good performance. All patches go through a convolutional layer to generate their initial feature embeddings. After that, they are formatted as a sequence with position embedding. An additional classification patch is added to the beginning of a recognition task's sequence. The self-attention mechanism described above is applied to the sequence of patches to generate a high-level representation for each patch. The classification patch can therefore aggregate embedding data from all other patches based on the attention and output of the final recognition result.

2.6 Masked Autoencoders (MAE)

Training ViT requires a vast quantity of labeled data and computational resources. However, many images have no labels. Masked Autoencoders (MAE) have been proposed for pre-training ViT model with unlabeled images [16]. Similar to the NLP pre-training that masks words in a sentence and utilizes the Transformer to infer [20], MAE also constructs a self-supervised task that masks image patches and then infers them using the ViT-based autoencoders.

As shown in Fig. 2.4, MAE is comprised of two steps. First, it masks some patches of an image and sends the rest to an encoder to generate high-level embeddings. Second, we combine these embeddings and the masked patches, which are then sent to a decoder to recover the original images. The masked patches are learnable vectors here since we do not know their content. The combined patch sequence goes through a decoder to infer the original pixel values of the masked

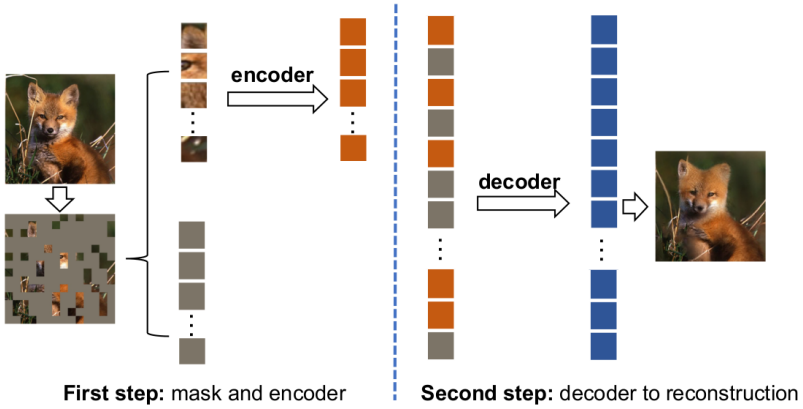


Figure 2.4: The process of MAE.

patches. The mean squared errors of the inferred values and the real values are used as the loss for the backward propagation. The decoder is a tiny ViT with a limited number of attention layers. As contrasted to the average pixel values of its neighbors, a large mask ratio can assist MAE in learning genuine visual semantics for reconstruction. The self-supervised trained encoder can be further fine-tuned based on the labeled images to obtain the final parameter for the classification task.

Chapter 3

Federated Graph Learning with Traffic Throttling and Flow Scheduling

This chapter focuses on the training of federated graph convolutional networks (GCN). GCN requires each node to aggregate its neighbors in the graph. However, when graph data is distributed among different data owners, the aggregation operation may cause cross-device communication. Direct data interaction may lead to privacy leaks, and frequent communication is a bottleneck for the whole system. To address this, we propose a secure federated graph learning system that encrypts transmitted data and explores redundant connections among data owners. We eliminates unnecessary transmission and schedules transmission flows more efficiently.

3.1 Introduction

Federated learning (FL) has shown great promise in enabling collaborative machine learning among distributed devices while preserving their data privacy [2]. When federated learning meets 6G networks, they can benefit each other while new challenges also emerge, which motivates us to study the communication efficiency

of federated learning in this Chapter.

Existing federated learning mainly focuses on convolution neural network (CNN) models that show superior learning accuracy in recognizing images and voices. However, many applications generate graph data (e.g., social graphs and protein structures) consisting of vertices and edges, which cannot be efficiently handled by CNN. Graph convolutional network (GCN) model [1] has been proposed to deal with graph learning by filtering the features of neighboring vertices.

In this Chapter, we study federated graph learning (FGL) under a cross-silo setting, including a set of FL servers maintained by different institutions (e.g., banks, hospitals, and universities). These servers may reside in public or private clouds, and they are connected by a wide-area network (WAN). Each server maintains a graph with edge connections to the others, and they cooperate in training a GCN model.

The FGL studied in this Chapter faces two main challenges. First, GCN training involves sharing features of neighboring graph nodes between FL servers, which leads to privacy leakage. Our previous work [25, 26] protects privacy by eliminating node feature sharing in the first layer of GCN. However, abandoning these node features may reduce learning accuracy. Second, there are massive devices in 6G, and existing works have shown that communication becomes the main bottleneck of distributed machine learning, which seriously affects the efficiency of learning tasks [2]. The communication challenge faced by FGL is more severe than traditional CNN-oriented federated learning. In addition to trainable parameter synchronization, servers in FGL need to exchange vertex embeddings in every graph convolutional layer during each training round due to its unique characteristics. Moreover, FGL relies on a wide-area network, which is shared by many applications, and bandwidth allocated to FGL is limited and dynamic.

Although various methods [3, 4] have been proposed to optimize communication, they differ from the problem studied in this Chapter. The most significant difference is that they usually only study the trade-off between learning speed

and energy efficiency [3, 4]. Privacy is not a concern. And the previous cross-silo related work [27] does not exploit GCN's characteristics about frequency embedding exchange. We also notice that some distributed systems [28, 29] conquer the communication issue by jointing data movement and task allocation schemes. In contrast, due to privacy protection, FGL does not allow such graph data movement. There also exist some general flow scheduling approaches like conventional shortest-flow-first [30], but they bring limited improvements. Some other complex scheduling approaches claim a better performance [31–33]. However, it should be quite a challenge to fully exploit the unique characteristics of FGL.

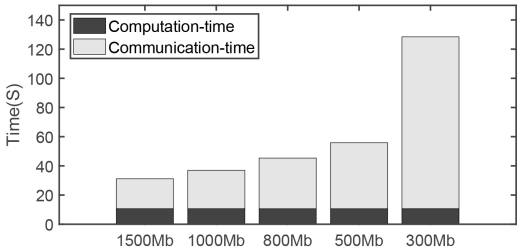
We propose S-Glint, a federated graph learning system with enhanced security insurance and a series of novel designs to address the communication challenge. In order to protect vertex embeddings, S-Glint adopts a homomorphic encryption (HE) based protocol that only transmits encrypted data. We adopt pre-aggregation and batching strategies with multiple HE servers to accelerate the encryption process. S-Glint further tackles the communication challenge with two novel designs. First, it reduces network traffic by eliminating the transmission of unimportant embeddings, also referred to as traffic throttling. Specifically, S-Glint lets each node quickly evaluate its neighbors' contributions based on marginal loss and only collects data from a subset of essential neighbors. The second novel design is a priority-based dynamic flow scheduling strategy. S-Glint monitors the training speed of FL servers and dynamically assigns different priority levels to network flows. Besides, traffic throttling and flow scheduling are jointly considered for further time savings. The main contributions of this Chapter are summarized as follows:

1. We propose a secure federated graph learning system called S-Glint to enable collaborative training of GCN models among distributed servers without leakage of their local graph data. We have identified that communication is the main bottleneck of the decentralized, federated graph learning system.

2. We analyze the graph training characteristics and propose a homomorphic

		Server 1	Server 2	Server 3
Server 1	Embedding	0	443 MB	440 MB
	Local model	0	0.25 MB	0.25 MB
Server 2	Embedding	416 MB	0	431 MB
	Local model	0.25 MB	0	0.25 MB
Server 3	Embedding	408 MB	427 MB	0
	Local model	0.25 MB	0.25 MB	0

(a) The amount of transmitted data.



(b) Time cost with various network bandwidths.

Figure 3.1: Example of cooperate training among three servers.

encryption-based embedding-sharing strategy for safety and efficient embedding interaction. We design the traffic throttling module to eliminate unimportant transmission. Besides, the flow scheduling problem is formulated and analyzed. We further combine traffic throttling and heuristic flow scheduling for joint optimization.

3. We simulate S-Glint based on trace data and evaluate its performance using a 20-server setting. Extensive experimental results show that S-Glint can significantly outperform existing works.

The rest of this Chapter is organized as follows. In Section 3.2, we first give some motivation. Then we present the system design in Section 3.3. The experimental results are presented in section 3.4, followed by some discussions in Section 3.5. The related work is in Section 3.6. We conclude our work in Section 3.7.

3.2 Communication Bottleneck in FGL

It has been well recognized that communication is the main bottleneck of distributed machine learning, especially in the WAN environment [34]. To better understand how communication affects FGL, we have conducted some preliminary empirical studies on a 3-server cluster, where each server is equipped with Inter i7-10700 CPU, 16GB memory, and Geforce RTX 2080 GPU. A switch connects three servers, and the network bandwidths of all links are set to 1500 Mbps. We divide the widely used Reddit graph dataset (232965 vertices with 602 features

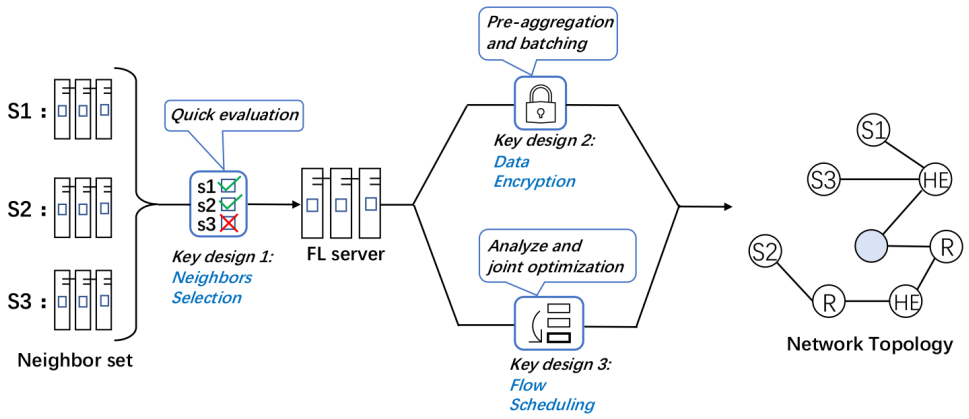


Figure 3.2: The architecture of S-Glint.

for each vertex) into three parts stored by these servers, respectively. Each server creates a two-layer GCN based on its local graph (including connected vertexes held by others) and exchanges trainable parameters after each training round. The amount of data exchange averaged over training rounds are shown in Figure 3.1. The size of trainable parameters is about only 0.25Mb, while the exchanged embedding size is over 400 Mb.

We then change the network bandwidth and study how it affects training time. As shown in Fig. 3.1, we can see that the communication time increases dramatically as bandwidth decreases, while the computation time is almost the same. Other graph data in practice may involve even billions of nodes and edges [7], which makes the FGL more challenging.

3.3 System Design

3.3.1 Overview

We consider a typical federated setting consisting of a set C of distributed servers connected by a WAN with limited bandwidth. Each FL server $c_i \in C$ holds a local training dataset expressed as a graph $\mathcal{G}_i = (V_i, E_i)$, where V_i and E_i denote the vertex set and edge set, respectively. Every vertex $v \in V_i$ is associated with a feature vector x_v that cannot be exposed to other servers. Note that there

exist some graph edges across servers. Each server is aware of the existence of connected vertices at other servers but cannot access their vertex feature vectors. For example, suppose there are some hospitals and a medical administration center. The global graph records, for a certain period, the city' s patients (nodes), their information (attributes), and interactions (links). Specifically, we have two kinds of links, the first one is explicitly stored in each hospital, and the second one is the cross-hospital links that may exist but are not stored in any hospital. The medical administration center may have the goal that the system obtaining a globally powerful graph mining model while not sharing data.

Based on the above scenario, we propose a secure federated graph learning system, S-Glint. S-Glint adopts a decentralized design to coordinate model parameter sharing and embedding sharing. At the beginning of each training round, each FL server collects model parameters from others and then averages them to get the initial weights used in the current-round training, eliminating the global barrier of the centralized parameter server in traditional federated learning. The schematic diagram of S-Glint is shown in Fig. 3.2. There are three critical designs in S-Glint. The first is neighbor selection. For each FL server, its neighbors may have various contributions; S-Glint aims to evaluate each FL server's neighbors and select essential neighbors. The second is data encryption. S-Glint adopts the HE mechanism with pre-aggregate and batching operations to efficiently encrypt transmitted data, thus protecting users' privacy. In the conference version [25], the features of nodes are multiplied by the weight at the second GNN layer. The decreased dimension thus protects privacy. We claim that this method mainly contains two weaknesses. First, although the above method is workable in most situations, it has a high risk of being exposed when the feature matrix is sparse since it lacks rigorous safety theory proof. Second, although the conference version considers the neighbors' information from the second layer, ignoring the first GNN layer harms the accuracy. In our experiments, the dimension decrease method causes a little accuracy decrease. The gap needs to be made up. The third key design

of S-Glint is flow scheduling. S-Glint analyzes the flow scheduling problem in our scenario and proposes a heuristics scheduling method with co-design to neighbor selection design. The details of the three designs are described in the following subsections.

3.3.2 Secure embedding sharing

For each FL server c_i, the set of nodes with edge connections to graphs held by other servers is denoted by V_i^n, whose embeddings are encrypted as $\{En_{pk}[h(v)], \forall v \in V_i^n\}$. These encrypted embeddings are sent to a HE server which aggregates them according to the requirements of FL servers. For example, FL server $c4$ requests embeddings of nodes v_3 from server $c1$ and v_5 from server $c2$, the HE server aggregates their encrypted embeddings as $En_{pk}[h(v_3)] + En_{pk}[h(v_5)]$ and sends the result to the $c4$.

Pre-aggregate

We find that the above process has high communication overhead due to the transmission of a large number of encrypted node embeddings from the FL servers to the HE server. By carefully examining this process, we find that the communication overhead can be reduced if FL servers can aggregate some embeddings before encryption. For example, if two shared node sets of different parties have repeated nodes, the repeated nodes can be aggregated first to avoid being encrypted and transmitted separately.

Motivated by the above example, we express embedding requests as a matrix and derive its basis. Then, each FL server encrypts its embeddings according to this basis and sends them to the HE server. Any linear combination of these encrypted embeddings at the HE server can be computed and shared with other FL servers.

Batching

We find that the ciphertexts generated by HE are significantly larger than plaintexts. Large ciphertexts incur high communication overhead. That is because ciphertexts have roughly the same number of bits with the key size rather than plaintext size. For example, graph embedding values are typically 32-bit, but the key size of Paillier, a popular HE lib, is 2048. To reduce ciphertext size, we adopt the batching technique [35] that concatenates several embedding values to encrypt them together so that they share the same plaintext while preserving the additive property.

Multiple HE Servers

The straw-man design has a single HE server, which would be the communication bottleneck since all shared embeddings, in the encrypted form, need to go through it. We deploy multiple HE servers to distribute the communication burden and eliminate this bottleneck. Specifically, we set one HE server and several FL servers to become a group. The HE server is responsible for the incoming data flow for all FL servers in the group (including the data interaction among the FL servers in the same group). The whole system then contains multiple groups and still works in a decentralized way.

3.3.3 Traffic Throttling

In S-Glint, each server collects feature embeddings from its neighbors. These neighbors have different contributions to the training process. We divide the commonly used Cora dataset (containing 2708 nodes and 10556 edges) into four parts stored by four servers. Server 1 and server 2 can train a model with 50% accuracy, while server 1 and server 3 achieve 60%. The basic idea of traffic throttling is to evaluate neighbors' contributions and eliminate unimportant data transmissions, which have a linear complexity. Traffic throttling mainly contains two phases: contribution evaluation and neighbor selection, whose pseudo-codes are shown in

Algorithm 1 Experience-based Traffic Throttling.

INPUT: C is the set of all servers, N_i contains the neighbors of server c_i, Epo is the total training epochs, and The previous k epochs are utilized for evaluation.
OUTPUT: Selected neighbors of every c_i.
 # The contribution evaluation phase :
for server $c_i \in C$ in parallel **do**
  for $r = 1, 2, \ldots, k$ **do**
    c_i calculates neighbors' embedding contributions scores with (3.1).
    c_i calculates neighbors' weight contributions scores with (3.1).
  end for
end for
 # The neighbors selection phase :
for each server $c_i \in C$ in parallel **do**
  Selects the neighbors according to the joint optimization with the threshold ψ.
  for $r = k+1, k+2, \ldots, Epo$ **do**
    Collects information from the selected neighbors.
    Estimates the unselected neighbors' embedding and model weight using (3.2) and (3.3), respectively.
    Conduct the local GCN training.
  end for
end for

Algorithm 1.

Contribution Evaluation

We follow the idea in [36] to quantify the contribution of shared embeddings. Note that the network flows associated with model sharing can be throttled in a similar way. Specifically, in each epoch r, server c_i first receives embeddings from all neighbors in set N_i and conducts training to generate a local model $M^r_{N_i}$. Then, for each neighbor $c_j \in N_i$, sever c_i also trains a model $M^r_{(N_i-j)}$, where $N_i - j$ means the neighbors set excluding the server c_j. The embedding contributions of c_j thus can be calculated as:

$$S_{j,i} = \frac{\sum_{r=1}^{k}[L(M^r_{N_i}) - L(M^r_{N_i-j})]}{\sum_{j\in N(i)}\sum_{r=1}^{k}[L(M^r_{N_i}) - L(M^r_{N_i-j})]}, \tag{3.1}$$

where $L(M_{N_i})$ denotes the training loss of model M_{N_i}. Both models $M^r_{N_i-j}$ and $M^r_{N_i}$ are trained based on the $M^{r-1}_{N_i}$ in the previous epoch. Note that k is the

number of epochs used for contribution evaluation.

The contribution evaluation process can be completed quickly with several reasons and strategies. Firstly, we claim that a few epochs (threshold k) are enough to evaluate the contributions, and it will be verified in Section 3.4. Secondly, the formulation (3.1) does not involve the process of complete backpropagation and verification, which will save time. Thirdly, compared to CNN, the GCN model is lightweight enough that the overhead in computation time is limited even in the case of multiple executions. Finally and most importantly, we propose a caching strategy to accelerate the process. There are many repeated nodes in the evaluation, and we can save the repeated nodes in the GPU to avoid reloading and reduce communication and computation overhead.

Neighbor Selection

The neighbor selection phase selects neighbors with high contributions and estimates the training input accordingly based on contribution evaluation results. In our conference version [25], we let each FL server sort its neighbors according to their contributions scores and select those with larger contributions until their total scores exceed the threshold ψ. However, this design may demand data transmission from a slow neighbor, while two faster neighbors may replace the contribution of this slow neighbor and save time. Thus we only expect the sum of the contribution to exceed the given threshold here. The specific neighbor selection needs to jointly consider the communication conditions and arriving speed of incoming flows. We will talk about the details in the following section.

As for ψ, our experimental results in Section 3.4 show that an appropriate threshold can speed up the training process with minor accuracy reduction. After neighbor selection, we estimate the embedding used for training input according to the collected ones from selected neighbors. For each server c_i with α_i flows, it extracts the embedding information contained in these flows and updates the local adjacent matrix $\overline{A}_{uv}$ as follows.

$$\overline{A}_{uv}^{(l-1)} = \begin{cases} \frac{|N(u)|}{|N'(u)|}\overline{A}_{uv}, & \text{if } v \in N(u), \\ 0, & \text{otherwise}, \end{cases} \tag{3.2}$$

where $N(u)$ denotes the set of neighbors of vertex u in the original graph, and $N'(u)$ denotes the set of neighbors obtained with α_i flows. Note that both $N(u)$ and $N'(u)$ contain internal neighbors and external neighbors maintained by other servers. For weight-sharing stages, we estimate the initial weights with β_i flows, whose senders are included in set $\mathbb{C}_i$:

$$\omega_i = \sum_{c_j \in \mathbb{C}_i} \frac{|V_j| w_j}{\sum_{c_j \in \mathbb{C}_i} |V_j|}, \tag{3.3}$$

where V_j is the set of labeled vertices held by server c_j.

3.3.4 Flow Scheduling

It has been well recognized that flow scheduling is significant for communication efficiency [3]. The intuitive idea is to formulate the scheduling process as an optimization problem. Thus we first formulate and analyze the problem.

Formulation

Suppose the data amount of flow f_{ij}^s is D_{ij}^s, which means the server c_i needs transmit D_{ij}^s data to the server c_j at training stage s. We suppose the server's index that takes the longest time to complete the total S training stage is 0. We let T_j^s denotes the completion time of the server c_j in the stage s. To minimize the total training time T, i.e., the completion time of the final stage, we formulate the flow scheduling problem as follows:

$$minT_0^S; \tag{3.4}$$

$$T_j^s = \max_t\{T_{ij}^s\} + t_{j,cpt}^s, \forall j, s; \tag{3.5}$$

$$T_{ij}^s = \max_t\{t \cdot x_{ij}^s[t]\}, \forall f_{ij}^s; \tag{3.6}$$

$$x_{ij}^s[t] \geq \frac{y_{ij}^s[t]}{D_{ij}^s}, \forall i, j, s, t; \tag{3.7}$$

$$\sum_t y_{ij}^s[t] \geq D_{ij}^s, \forall f_{ij}^s; \tag{3.8}$$

$$\sum_{f_{ij}^s \in F(p)} y_{ij}^s[t] \leq B_p \forall t, p; \tag{3.9}$$

$$x_{ij}^s[t]\hat{T}_{ij}^s \leq t \cdot x_{ij}^s[t], \forall f_{ij}^s, t; \tag{3.10}$$

$$T_i^{s-1} \leq \hat{T}_{ij}^s, \forall f_{ij}^s, t; \tag{3.11}$$

Our objective is to minimize the completion time of the final stage, which is expressed in constraint (3.4). For each stage s, the completion time of the server c_j is calculated by (3.5), where T_{ij}^s denotes the completion time of flow f_{ij}^s and $t_{j,cpt}^s$ means the computation time of the server c_j for the stage s. We define a binary variable $x_{ij}^s[t]$ to denotes whether we transmit the flow f_{ij}^s in the time slot t. Then the flow completion time T_{ij}^s can be expressed by (3.6). Another integer variable $y_{ij}^s[t]$ is defined to denote the amount of transmitted data of flow f_{ij}^s in the time slot t, whose relationship with $x_{ij}^s[t]$ is shown in (3.7). If f_{ij}^t is transmitted in time slot t, i.e., $y_{ij}^s[t] > 0$, the binary variable $x_{ij}^s[t] = 1$. Otherwise, we have $x_{ij}^s[t] = 0$ due to the minimization objective. Constraint (3.8) presents that the total amount of transmitted data should be no less than the flow size. Due to the bandwidth constraint of each network link, we have constraint (3.9), where $F(p)$ denotes the set of flows going through the network link p. We let $\hat{T}_{ij}^s$ denotes the start time of flow f_{ij}^s and it should be no less than the completion time T_i^{s-1} of the previous stage, which is represented by constraints (3.10) and (3.11).

The above formulation is hard to solve directly. It is a mixed-integer non-

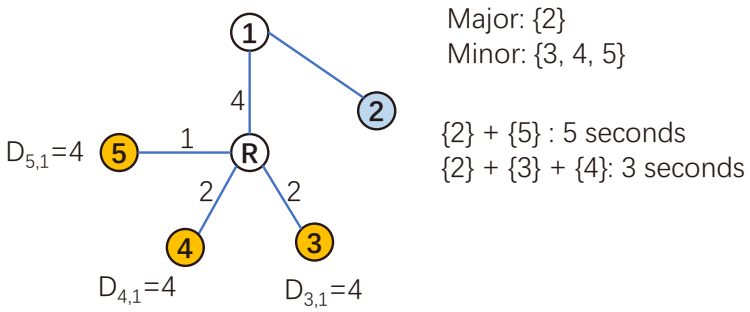


Figure 3.3: An example of joint optimization.

linear programming problem, which is generally NP-hard. Although it is easy to transfer it into a linear form by applying some internalization techniques, it is still challenging to solve this large-scale optimization problem since federated graph learning contains many stages. Besides, the network bandwidth B_p could be dynamic because many applications share the network.

Insights

Although it is challenging to find the optimal scheduling solution, there are some heuristic insights when revisiting the federated graph learning details. First, the flows belonging to the earlier stages should be transmitted preferred because they can enable the received server to finish the current stage computation faster. Second, we need to consider the computation time of the servers after receiving flows. If a server needs a longer time for training, its incoming flows should have a higher priority to avoid being the bottleneck.

Since the neighbor selection decision in the previous traffic throttling module has multiple possibilities, which will seriously affect the flow scheduling module, S-Glint adapts a joint optimization scheme to combine the neighbor selection and flow scheduling. The details are described as follows.

Joint Optimization

Our conference version [25] only considers the above two insights into a dynamic flow scheduling strategy while ignoring the selection of neighbors. Here we

Algorithm 2 Joint Optimization.

INPUT: There are P priority levels. Each server c_i divides its output flows into major flow set $\{F_{i_1}\}$ and minor flow set $\{F_{i_2}\}$. c_{j_1} is the major neighbor set of c_i, c_{j_2} is the minor neighbor set of c_i. C is the FL server set.
OUTPUT: The final transmission time cost.
for server $c_i \in C$ in parallel **do**
  # The first period
  When c_i starts its $r-th$ training epoch, c_i collects stage set St_i, time set T_i and records their size Len_i.
  c_i distributes priorities of flows in $\{F_{i_1}\}$ according to (14) and (15).
  c_i distributes lowest priorities of flows in $\{F_{i_2}\}$.
  if a server c_i receive all major neighbors' data **then**
    # The second period
    for server $c_j \in c_{j_2}$ **do**
      if c_j transmit data to c_i before **then**
        c_j will be selected
      end if
      select servers in c_{j_2} according to estimated arrival time.
    end for
    Turn the selected neighbor in minor to major and distribute priorities according to (14) and (15).
  end if
end for

propose a joint heuristic scheme with linear complexity. The process is shown in Algorithm 2. For a FL server c_i, we classify its neighbors as two sets, $\{c_{j_1}\}, j_1 \in N_i$ and $\{c_{j_2}\}, j_2 \in N_i$. $\{c_{j_1}\}$ contains the major neighbors whose embedding information is indispensable. c_{j_2} contains the minor neighbors part of them may be selected to match the contribution threshold.

Fig. 3.3 gives an example of neighbor selection. Suppose server 1 has the major neighbor 2 and minor neighbor set $\{3, 4, 5\}$. Suppose the contribution of neighbor 5 is larger than neighbor 3 and neighbor 4, and the server has two choices to match the contribution threshold, $\{2\} + \{5\}$ or $\{2\} + \{3, 4\}$. Obviously, if we still select neighbors according to the contribution order, neighbor 5 will be selected, and it will take 5 seconds for minor neighbors. Otherwise, if we choose $\{3, 4\}$, it only takes 3 seconds.

Since the selection of neighbors is related to both their contributions and arrival time, we separate our joint optimization into two periods. Specifically, we let each

FL server receive all neighbors' data for the first period, and a dynamic priority-based flow scheduling method has been adopted. The main idea of the scheduling is that all the major neighbors should prioritize over the minor neighbors, and we should arrange proper priorities among the major neighbors. We suppose the system provides P priority levels that range from 0 to $P-1$, where 0 means the highest priority level. In training epoch r, server c_i maintains a set of flows F_i to be transmitted to other servers, which can be divided as major flows set F_{i_1} and minor flows set F_{i_2} according to the major or minor neighbor set it belongs. It also collects the information of corresponding receivers' current training stage and previous training stage time cost of the flows in F_{i_1} and maintains them in set St_i and T_i, respectively. We let Len_i denotes the size of St_i. We first sort the flows in F_{i_1} according to their receivers' training stages. The flows are ordered from the smallest training stage value to the largest one, and the smaller one means a larger priority. For the flows whose receivers are in the same training stage, we describe them as a sub-flow set and further sort the flows in each sub-flow set according to the time cost information. The sorted F_{i_1} is denoted as F'_{i_1}. For a flow f_j in F'_{i_1}, we calculate its priority P_j as follows:

$$ind = \lceil Len_i / P \rceil, \tag{3.12}$$

$$P_j = \lceil j / ind \rceil. \tag{3.13}$$

Then, the flow f_j is put into the P_j priority level queue. Note that all flows belonging to F_{i_2} are also sorted and only put into the $P-1$ priority.

Once a server obtains all flows from its major neighbors, it starts the second period. The server needs to choose neighbors in the minor set in the second period. If a neighbor in the minor set has already transmitted some data to the server previously, it will be chosen, like neighbor servers $\{3, 4\}$ in the previous example, and the rest minor neighbors whose estimated arrival time is the shortest will also be selected until the sum of the contributions exceeds the threshold. If all

Table 3.1: Graph Datasets

Dataset	Nodes	Edges	Features	Classes
Cora	2,708	10,556	1,433	7
PubMed	19,717	88,651	500	3
Coauthor physics	34,493	991,848	3,703	6
Reddit	232,965	114,848,857	602	41

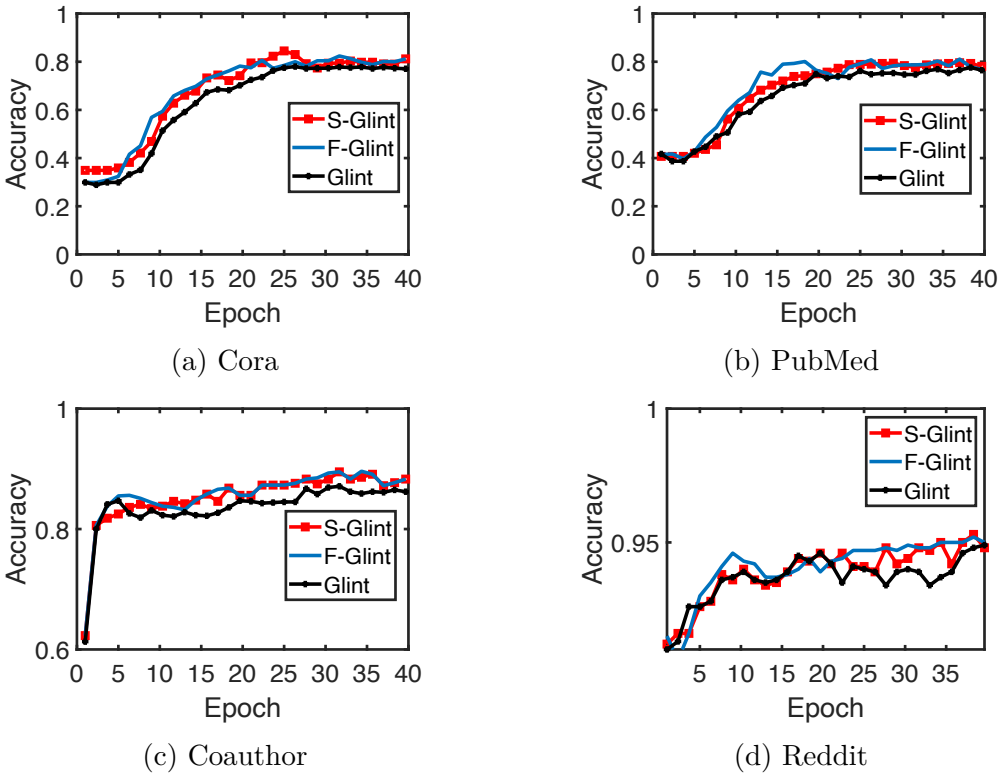
the neighbors in the minor set have no data transmission before, all selections are based on estimated arrival time. The selected neighbors in the minor set then turn to the major neighbor, and their priority level will also be distributed from the above principle. The transmission from unselected neighbors will be eliminated.

The above joint optimization process considers dynamic training speed and time cost to avoid the slower party becoming the bottleneck. It also ensures that the major neighbors are selected, and the minor neighbors are selected according to their coming speed to accelerate the process.

3.4 Experiments and Evaluation

3.4.1 Experimental Settings

We simulate S-Glint based on PyTorch, and a python graph learning package, named Deep graph Library (DGL) [37]. The hardware includes Inter i7-10700 CPU, 16GB memory, and Geforce RTX 2080 GPU. We choose four widely used graph datasets: Cora, PubMed, Coauthor, and Reddit, whose details are summarized in Table 3.1. We extract a network topology from a real peer-to-peer network [38], which contains 20 FL servers, 10 HE servers, and 76 routers. Each HE server is responsible for the incoming data streams of two FL servers. The trace about data transmission among each part thus can be obtained through their real data interaction demand when we divide each dataset into 20 parties. Different links do not have identical capacities, and we set the bandwidth of links randomly from 5 Mbps to 500 Mbps. Note that network bandwidth may fluctuate over time, and network congestion can also happen. Thus we randomly set



(a) Cora (b) PubMed

(c) Coauthor (d) Reddit

Figure 3.4: Time-accuracy performance comparison.

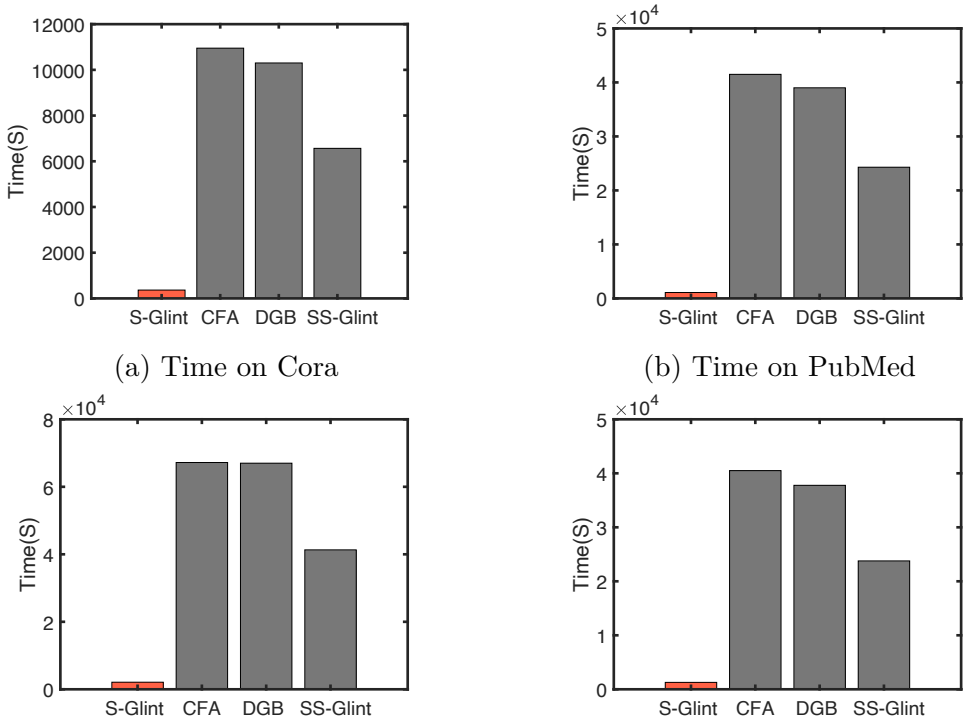
the bandwidth values of all links according to a Gaussian distribution, where the standard deviation is one-tenth of the mean. The threshold k, which represents the number of epochs for contribution evaluation, is set to 10. The threshold ψ, which denotes the sum of transmitted flows' contribution score, is set to 0.9. Each server randomly chooses a subgraph from a given dataset and trains a two-layer GCN model with the ADAM optimizer.

3.4.2 Results

We evaluate S-Glint in different dimensions with various baselines.

Accuracy Results

To evaluate the accuracy results of S-Glint, we choose two baselines. The first is F-Glint, where each FL server has no traffic throttling part. The second is our conference version, Glint [25]. Glint lets each FL server transmit the embedding information after the dimensional reduction in the second GCN layer to protect



(a) Time on Cora

(b) Time on PubMed

(c) Time on Coauthor

(d) Time on Reddit

Figure 3.5: Overall time cost comparison.

privacy and contains independent traffic throttling and flow scheduling strategies. The accuracy convergence of them over different datasets is shown in Fig. 3.4. We can see that the accuracy of S-Glint is similar to the F-Glint, while Glint has a slight gap between S-Glint and F-Glint. For example, for the Cora dataset, S-Glint and F-Glint both have an accuracy of about 81%, while Glint is bout 80%. The reason is that Glint still ignores some information interaction since it only transmits data in the second GCN layer. F-Glint guarantees accuracy while does not consider privacy issues. S-Glint achieves similar accuracy to F-Glint and further efficiently encrypts the data to protect privacy and utilizes joint optimization to release the communication burden.

Overall time cost

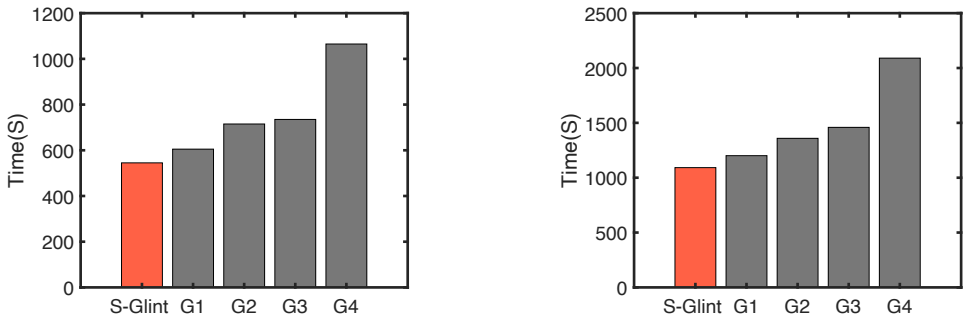
We also compare S-Glint with three baselines to evaluate the overall time cost. The first is centralized federated average graph learning (CFA). We extend FdeAvg [17], which has been widely used by many federated learning schemes,

to implement a centralized federated average graph learning (CFA) scheme by adapting the single priority-based flow scheduling strategy like in Glint for the embedding exchanging part. It has a secure embedding sharing strategy without pre-aggregation and batching. The second is decentralized gossip-based federated graph learning (DGB). Combo [39] uses a segment operation and a gossip protocol to deal with local models' aggregation under a decentralized-federated learning scenario. The basic idea is to let the models take random walks in the network topology and get updated when they arrive at a server, which is different from ours. We extend Combo to implement a decentralized gossip-based federated graph learning (DGB) baseline with Glint's same single-flow scheduling strategy. It also has a secure embedding sharing strategy without pre-aggregation and batching. The third is simple S-Glint (SS-Glint) with joint optimization, while the secure embedding sharing strategy has no pre-aggregation and batching.

We measure the completion time of different systems after 40 training epochs when they all have converged and show the results in Fig. 3.5. We can see that S-Glint achieves about 30x-40x speedup than other CFA and DGB. The improvement comes from the efficient, secure embedding sharing strategy and joint optimization. The performance of DGB is a little better than CFA because its local model aggregation needs less communication time. However, the improvement is limited because local models are a small percentage of the whole network traffic, as shown in Fig. 3.1. The effectiveness of the encryption acceleration in our secure embedding sharing is just the gap between S-Glint and SS-Glint, which is about 10x-25x speedup. The traffic throttling and flow scheduling bring about 40% to 50% improvement when comparing CFA and DGB with SS-Glint.

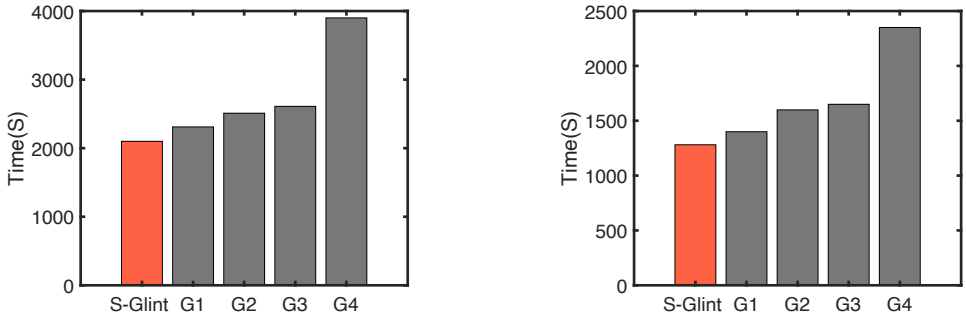
The Effectiveness of the Joint Optimization

To further evaluate the joint optimization, we also choose several related baselines. Firstly, we modify S-Glint to separate the joint optimization part as independent traffic throttling and flow scheduling as in Glint, named G1. Secondly,



(a) Time on Cora

(b) Time on PubMed



(c) Time on Coauthor

(d) Time on Reddit

Figure 3.6: The effectiveness of the joint optimization.

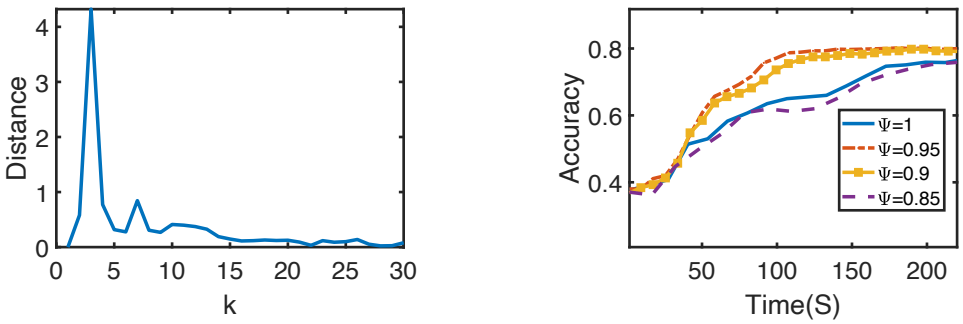
we modify S-Glint to adopt traffic throttling and shortest-first scheduling, named G2. Thirdly, we modify S-Glint to adopt traffic throttling and fair-share scheduling, named G3. Finally, we modify S-Glint to abandon the traffic throttling part, named G4.

The results are shown in Fig. 3.6. We can find that the joint optimization brings about 10% improvement compared with the separate one. Even though we separate the joint optimization, there is also about 10% improvement compared with other scheduling methods. The specific traffic throttling strategy brings about 40% benefits when comparing S-Glint with G4.

3.4.3 The Influence of System Parameters

S-Glint's performance is also affected by system parameters, k represents the number of training rounds for contribution evaluation, and ψ denotes the threshold for neighbor selection in traffic throttling.

We use the Cora dataset as an example to show the influence of parameters



(a) Various k (b) Various ψ

Figure 3.7: The influence of system parameters.

in Fig. 3.7. We conduct experiments to calculate the embedding contribution scores of server 1's neighbors. We first calculate the scores based on the complete training epochs. The scores of all neighbors thus can be formed as a vector. Then we compare the Euclidean distance between the above vector and the new vector calculated based on various k. When the value of k is larger than 10, their distance is stabilized and approaches 0. Similar phenomena occur in other datasets. Thus we set k to 10 in previous experiments.

For the threshold ψ, we set it as 1, 0.95, 0.9, 0.85, respectively. The choice of neighbors is based on the joint optimization until the sum score of neighbors exceeds the threshold. We can find that a smaller ψ may cause a large input error and further result in no coverage or slow down the training. We set ψ as 0.9, which has attractive performance on all datasets in practice.

3.5 Discussions

The Efficiency of the Caching Strategy. S-Glint introduces a caching mechanism in the neighbor contribution evaluation part to reuse the common nodes to accelerate the process. We claim that the caching strategy only brings very limited overhead. We use the tensor mask operation on GPU in the cache process to avoid complex index operation, thus having a minor increase in the computation overhead. In fact, the data loading from CPU to GPU takes the main time in the training of GNN [6], thus the limited computation overhead of our

caching strategy is favorably offset by reducing the data loading communication.

The Overhead of the Encryption. Homoerotic encryption usually is a heavy operation with time-consuming. In S-Glint, we have two core strategies to reduce encryption time. The first one is pre-aggregation. We do not encrypt every original node's features, instead, we express the request as a matrix and derive the basis. The encryption is according to the basis, thus can reduce the amount of encrypted data. In our experiments, The reduced ratio can be achieved from about 10% (Cora dataset) to about 30% (PubMed dataset) when the dataset is divided into 20 parties.

The second strategy is batching. In HE, the size of ciphertexts is only related to the key rather than plaintext. The batching technique can concatenate plaintext and reduce the computation and communication overhead. We use the Cora dataset as an example to compare the S-Glint (with pre-aggregation and batching), SS-Glint (direct apply HE without pre-aggregation and batching), and P-Glint (plaintext with no encryption operation). Their overall time costs for the 40 epochs are about 560 seconds, 6280 seconds, and 230 seconds. We find that direct encryption brings a very high overhead on communication and computation, while the proposed strategies in S-Glint successfully reduce the overhead by an order of magnitude.

The Synchronization in S-Glint. There are two kinds of transmitted data flows in the network. The first is the shared encrypted embedding, and the second is the encrypted weight. In a decentralized scheme, a naive method to synchronize the weight lets every participant transmit its model to all of the others, making the synchronization the basic bulk synchronous parallel (BSP). However, in S-Glint, we do not force this kind of fully-connected transmission. Instead, the traffic throttling strategy has been adopted to eliminate unnecessary transmission. Since we reduce the transmission, the communication overhead will be released. As the results are shown in the previous section, there are about 40% brought by the traffic throttling strategy. The benefit comes from eliminating some unnecessary

and slow neighbors for each FL server.

The Overhead of Neighbors' Evaluation. In the contribution evaluation process, the FL server needs to perform forward propagation many times based on the marginal loss. We claim that this brings limited computation overhead due to the characteristic of GCN (small model) and our strategy, caching (quickly achieved by the tensor mask operation).

We compare the time cost of various components with a single training epoch of the Cora dataset, including forward-backward propagation (original training), our evaluation process, and the whole communication time. Their time costs are about 0.48 seconds, 1.25 seconds, and 10.57 seconds. Although our evaluation takes several times on time cost when compared with the original one, it only occupies a tiny part of the overall time (communication takes the most). Thus the evaluation process is far from being the bottleneck of the system. Besides, the evaluation process was only conducted in the several previous epochs in training, further releasing its computation burden.

3.6 Related work

3.6.1 Federated Learning

Federated learning has been proposed to enable joint learning among distributed data owners without privacy leaking concerns [17]. Due to its great promise, substantial growth has occurred in this research field. For example, to address data non-IID issues, Zhao et al. [40] explain the impact with mathematical and try to release the problems by sanding a set of uniform distribution data among servers. Mehryar et al. [41] proposed an agnostic federated learning scheme to avoid distribution bias. Recently, the DRL algorithm has been adapted to dynamically select a subset of training participants by Wang et al. [42], which can accelerate the model convergence and alleviate the non-IID issue.

3.6.2 Graph Convolutional Networks

After being proposed, GCN shows its effectiveness in various fields like recommendation systems [7], temporal link prediction [43], and spam review detection [44]. A realistic situation is that the graph's size is usually too large to load in the memory. Thus various sampling strategies have been proposed to make the training process more efficient. GraphSAGE [5] mandates each node to sample a fixed number of neighbors in each layer, which may lead to data exploding when layers get deeper. To release the issue, VR-GCN [45] utilizes history neighbors' embedding to control variance and reduce the sampled neighbors to two nodes. Cluster-GCN [46] clusters the large graph into subgraphs and then completes convolution separately. FastGCN [47] fixes the number of nodes in each convolutional layer to avoid data exploding.

3.7 Conclusion

This Chapter proposes a decentralized and secure federated graph learning system, named S-Glint, to jointly train a global GCN model among distributed graph data owners. We adopt homomorphic encryption (HE) based security protocol with pre-aggregation and batching strategies to preserve privacy efficiently. The traffic throttling and flow scheduling strategies are jointly optimized to alleviate the communication burden further. The former evaluates the embedding contribution of neighbors and selects partial of them while estimating others. The latter dynamic tuns the priority of flows according to the training stage and training completion time. We conducted multi-dimensional comparison experiments with various baselines, including a centralized solution and a decentralized solution. The experimental results have shown the superiority of S-Glint over the baselines.

Chapter 4

Graph Inference with Adaptive Sampling and Local Sensitive Hash

This chapter focuses on the inference process of the GCN. We observe that many nodes in a graph are loaded repeatedly into the GPU during the inference process, significantly reducing inference speed. To address this, we propose a more efficient system that adaptive sampling neighbors in the graph and reuses loaded data to accelerate the inference process. Our system also reorders inference batches according to their similarities with a local sensitive hash (LSH)-based clustering scheme to reuse as many nodes as possible.

4.1 Introduction

Recently, the emerging graph neural networks (GNNs) have received lots of attention because of their impressive capability in dealing with graph data for various network-based tasks like network traffic forecast, network traffic scheduling, network node classification, etc. [1, 5, 7, 44, 48–50]. The basic idea of GNNs is that each node aggregates the features of its neighbors and generates a new representation via a linear or non-linear transformation. This operation can lead

to a significant waste of energy because of the extensive, repeated aggregation.

We have seen many research efforts aimed at improving the training efficiency of GNNs. For example, instead of aggregating all neighbors, GraphSAGE [5] samples a subset of neighbors and aggregates their features, thus saving lots of computational resources. Chen et al. [51] find that this sampling method cannot well guarantee the training convergence, and it uses historical node activations to reduce the variance of aggregation results. Cluster-GCN [46] splits a large graph into subgraphs and then conducts graph convolution operations separately to avoid memory overflow. GraphSAINT [52] generates subgraphs according to node importance to improve training efficiency.

A sustainable GNN should be efficient in both training and inference. However, all of the above works focus on GNN training, while the inference has been seldom studied. The GNN inference in social networks and product networks is the core operation of many graph-related businesses [7, 8], which aims to provide efficient inference services with low-latency and high-throughput. Unfortunately, the efficiency of existing GNN inference methods is still far from meeting these requirements. The main bottleneck stems from the overhead incurred when loading graph data from the main memory to the GPU, which has been confirmed by our experimental results in Section 4.2. The data loading time is even longer than the inference time itself. We further find that different inference batches contain many common nodes, and their features are repeatedly loaded into the GPU by the current systems, which leads to redundant energy consumption and time delay. In fact, this redundant loading issue is even more severe in the inference than in the training since there is no sampling operation in the inference for accuracy consideration [5]. This observation motivates us to improve the efficiency of GNN inference by reusing graph data already loaded into the GPU to avoid redundant data loading. And an accuracy-guaranteed sampling strategy is needed for the inference. Note that a similar reuse idea has been proposed for GNN training by [6]. However, the order of batches in the training is random

and non-controllable, while the inference operation can be conducted periodically in an offline scenario to reorder batches. For instance, PinSage [7] utilizes Map Reduce to generate embedding in an offline process. GEM [8] detect malicious account with GNN daily. As a result of its lack of flexibility, the static solution in [6] cannot scale to large graphs in the inference.

In this Chapter, we propose **RAIN** [1], a sustainable and efficient inference system for graph learning. When **RAIN** finishes the inference computation of the current batch, it maintains the graph data that the next batch can reuse in the GPU instead of cleaning GPU memory. In such a way, we can reduce the amount of data loaded from the main memory to the GPU. Motivated by this data reuse idea, given a number of inference batches, we can reorder their computation sequence so that adjacent batches have more common nodes. A straightforward way of reordering inference batches is to compare the similarity between pairs of batches and then conduct inference for similar batches sequentially, which unfortunately leads to high overhead, especially when there are a large number of batches and each batch contains massive nodes.

To address this challenge, **RAIN** abandons pair-wise comparison and adopts a Local Sensitive Hash (LSH) [15] based hierarchical batch clustering scheme. The scheme conducts coarse-grained clustering first and then turns to fine-grained clustering, and the batches in the same fine-grained cluster will be arranged adjacently. Specifically, we first leverage different sizes among batches to naturally divide them into different coarse-grained clusters. Every batch in GNN has the same number of target nodes but varying total sizes because different target nodes have different amounts of neighbors. The insight of the above operation is that there are more repeated nodes among large batches in terms of probability. After coarse-grained clustering, each cluster still contains lots of batches that need to be clustered further. Instead of using complex similarity computation over every pair, we aim to quickly map the similar batches into the same feature space and

[1]Source code: https://github.com/xiaobing0/RAIN .

become the same fine-grained cluster. The LSH method is utilized here. The main idea behind LSH is to use multiple hash functions and double hash processes to obtain similarities among batches of different lengths. We obtain the clustering results based on the output of LSH. We further adopt a simple index-sampling strategy for all batches in the same coarse-grained cluster to accelerate the LSH.

To further alleviate the communication burden of data loading, **RAIN** also provides an adaptive sampling strategy that aims to reduce the loaded nodes while guaranteeing accuracy. It separates batches into different parts according to their average degrees. The batch with a low average degree saves all neighbors, while those with high degrees adaptive sampling target nodes' neighbors. We set the number of the sampled nodes proportional to the size of the degree. The adaptive strategy's insight comes from the intuitive idea that there are many redundant neighbors with high-degree nodes, and we need to sample more neighbors to control the bias of the high-degree nodes. Our experiments show that a proper sampling strategy can significantly save time while having a negligible accuracy decrease. Our main contributions are summarized as follows:

- We analyze the data loading overhead issue in the inference of graph learning and claim that the redundant data loading is even more severe than the training process.

- We propose our clustering-based efficient inference system, **RAIN**. **RAIN** utilizes LSH to cluster the batches quickly with a two-level scheme. The same nodes from two adjacent batches will be reused to reduce repeated data loading.

- **RAIN** further contains the adaptive sampling strategy to save lots of inference time while having a negligible accuracy decrease.

- We conducted extensive experiments to verify the effectiveness of the proposed system.

Table 4.1: The original nodes and loaded nodes of four datasets.

Dataset	Nodes	Loaded nodes
Reddit	232,965	32,177,528
Yelp	716,847	23,795,116
Amazon	1,598,960	274,445,528
OGB-products	2,449,029	237,185,914

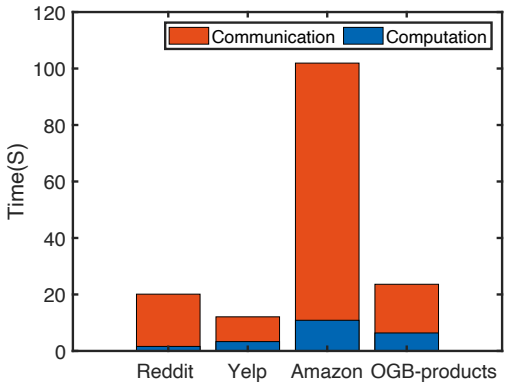


Figure 4.1: Time cost for the inference on four datasets.

The rest of this Chapter is organized as follows: The motivation is given in Section 4.2. The LSH-based graph workload clustering is presented in Section 4.3 and the adaptive sampling strategy is proposed in Section 4.4. In Section 4.5, we conduct extensive experiments to verify the effort of our system, and the related work is summarized in Section 4.6. We finally conclude our Chapter in Section 4.7.

4.2 Motivation

The inference of graph learning usually needs to load the data into the GPU (communication) and then calculate the forward process (computation). It is unrealistic to load the entire graph into GPU when facing large graph data due to memory limitations. Instead, we divide the graph into multiple batches to process in sequence [6]. There is a lot of data redundancy among batches.

Here we conduct some preliminary experiments to study the time cost of communication and computation in inference. The experiments are based on PyTorch and Deep Graph Library (DGL) [37]. The hardware includes Intel i7-10700 CPU,

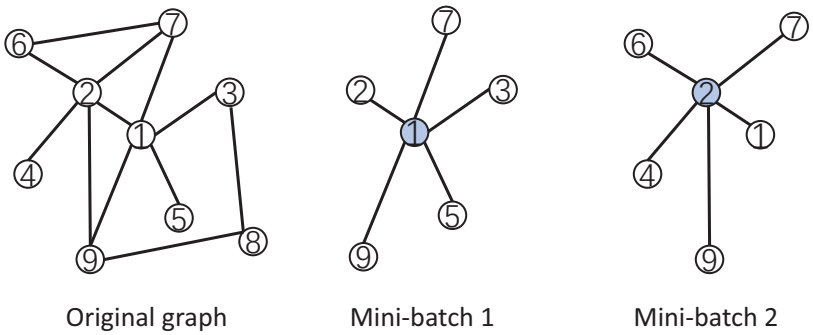


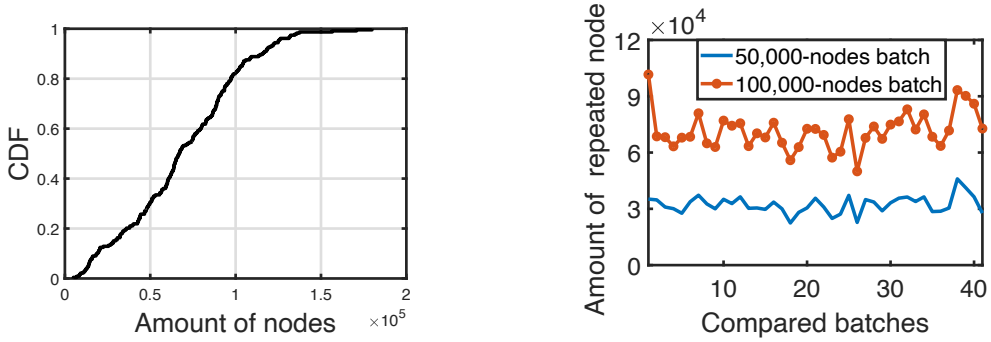
Figure 4.2: The process of inference in different mini-batches.

8GB memory, and Geforce RTX 1080 GPU. We conduct experiments based on four commonly used network-based datasets, Reddit (social network) [5], the OGB-products (product co-purchasing network) [53], the Yelp (social network) [52] and the Amazon [52] (product co-purchasing network). The number of their nodes is summarized in Table 4.1.

The results of inference time are shown in Fig. 4.1. We can see that the communication time is longer than the computation time on all datasets. The gap mainly comes from repeated data loading. As the example shown in Fig. 4.2, suppose there are two mini-batches with node 1 and node 2 as their target node, separately. The two target nodes and their neighbors will be formed as a subgraph and loaded into the GPU for aggregation. We can see that there are four repeated nodes in two subgraphs, which means these nodes need to be loaded twice and cause severe overhead.

Table 4.1 also gives the number of loaded nodes among four datasets. We can see that the number of loaded nodes is tens or even hundreds of times greater than the original nodes, where the ratio is about four in training [6]. The exact ratios of time costs for communication and computation among different datasets are related to their average degree and features' dimensions.

Note that there is no neighbor-sampling operation in the inference, which will cause the scalability problem. For example, the original process in GraphSAGE will generate a subgraph for each inference batch that includes all the target nodes and all of their 2-hop neighbors (which means two layers). In our experiment,



(a) The CDF curve of the number of nodes among inference batches.

(b) Amount of repeated nodes among different batches.

Figure 4.3: Several observations.

the size of this single-batch subgraph may include almost all of the nodes in the graph, which could cause memory to overflow. And repeated loading of a large number of redundant nodes between these subgraphs will make the whole inference process extremely slow. To reduce the overhead, DGL first computes the embedding of all nodes for only one layer at a time to reduce the size of the subgraph. After conducting the process twice, all nodes complete the inference of two GNN layers. Although DGL reduces some overhead, there are still a lot of repeat communications between batches, as shown in Table 4.1. We claim that communication is the bottleneck in the inference process, even with the strategy in DGL.

4.3 LSH-based Graph Workload Clustering

The basic idea of **RAIN** is to cluster the batches according to their similarity and arrange the batches in the same cluster adjacently to reuse the repeat data. To achieve efficient clustering, we start with two observations.

4.3.1 Observations

First, we observe that batches show a great diversity of sizes, even across several orders of magnitude. We count the number of nodes in batches in the Reddit dataset and show the CDF curve in Fig. 4.3a. We can see that the smallest batch

Algorithm 3 Coarse-grained clustering.

Require:

$\{Q_i\}$ is the set of batches with the original order.

λ is the size of the interval.

Ensure:

K coarse-grained clusters, $\{Q_{k'}\}$.

Define K intervals (clusters) according to the λ.
for $i = 1, 2, 3, ...len(\{Q_i\})$ **do**
  Cluster the batch Q_i to different intervals according to its size.
end for
for $k = 1, 2, 3, ...K$ **do**
  $\{Q_k\}$ is the batch set of the $k-th$ cluster.
  for $j = 1, 2, 3, ...len(\{Q_k\})$ **do**
    Sampling the index on the batch Q_{kj} to generate $Q_{k'j}$.
  end for
  $\{Q_{k'}\}$ is the sampled batch set of the $k-th$ cluster.
end for

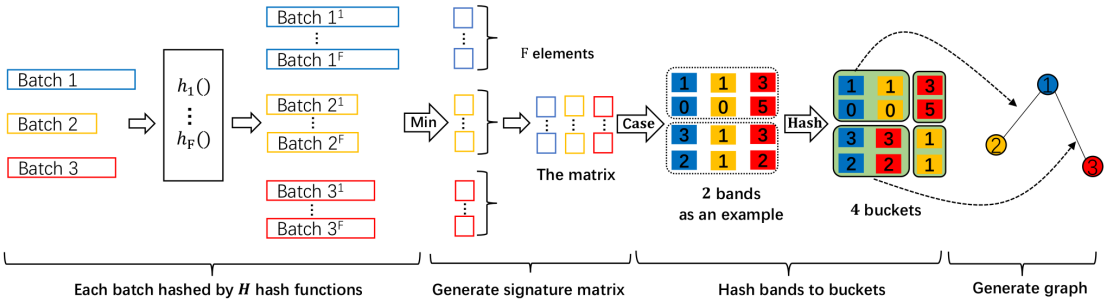


Figure 4.4: The process of generating a similarity graph.

contains thousands of nodes, while the biggest one has over a hundred thousand. This phenomenon is reasonable since the neighbors of nodes are distributed from several to hundreds. Our second observation is that the number of common nodes among larger batches is higher than that among smaller batches. We randomly select a 50,000-node batch and a 100,000-node batch and compare their common nodes with all batches whose size is more than 100,000. Figure 4.3b shows the result. In a large batch, target nodes usually have high degrees, so they would have a high possibility of being reused in other batches. We have similar observations on other datasets.

Algorithm 4 Fine-grained clustering.

Require:
$\{Q_{k'}\}$ is the set of batches of a coarse-grained cluster.
H is the number of hash functions.
Ensure:
The final set of batches, $\{Q_{new}\}$.
#Generate signature matrix.
1: **for** $i = 1, 2, 3, ...len(\{Q_{k'}\})$ **do**
2: **for** $n = 1, 2, 3, ...F$ **do**
3: Calculate hash value for each element in the batch $Q_{k'i}$ by $f_n()$;
4: m_{in} is the min-hash value of the batch $Q_{k'i}$, generated by the $n-th$ hash function.
5: **end for**
6: M_i consist of $\{m_{in}, n = 1, 2, 3, ...F\}$.
7: **end for**
8: Each batch is a column, the signature matrix M consist of $\{M_i, i = 1, 2, 3, ...len(\{Q_{k'}\})\}$.
#Divide M into b bands by rows and hash separately.
9: Each band contains r rows.
10: A graph $G = (V)$, V is the set of index of batches.
11: **for** $j = 1, 2, 3, ...b$ **do**
12: **for** $i = 1, 2, 3, ...len(\{Q_{k'}\})$ **do**
13: Hash r min-hash values $M_i[j * (r-1), j * r]$ of the $j-$th band of batch i to the bucket.
14: **if** The bucket already exists another batch i' **then**
15: Nodes i and i' in the G will have a connection and the weight of the connection will be plus one.
16: **end if**
17: **end for**
18: **end for**
Arrangement
19: Cluster the G into ψ parts and generate new order of $\{Q_{k'}\}$ as $\{Q_{new}\}$, where the batches belong to the same cluster will adjacent front and rear.

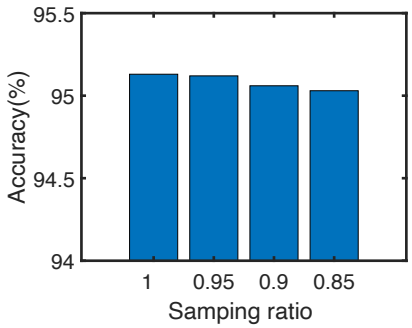
4.3.2 LSH-based Hierarchical Clustering

RAIN adopts a two-level scheme to accelerate the clustering inspired by the above observations. The coarse-grained clustering process is based on batch size information. Algorithm 3 shows the process of coarse-grained clustering. We sort batches according to their size and then divided them into K clusters. For each cluster $\{Q_k\}$, we further sample the indexes in every batch to generate a new batch (line 8). The algorithm has a linear complexity. The sampling can save on computation overhead when we further conduct fine-grained clustering

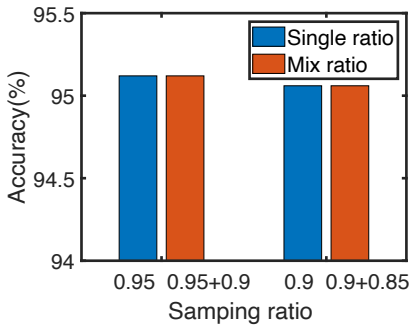
since fewer elements are included. The coarse-grained clustering design can significantly reduce the overhead compared with full-set clustering since there is no need to compare across coarse-grained clusters. Meanwhile, according to the second observation, it has a negligible negative impact on the final result since there are more repeated nodes in large-scale batches by nature.

The algorithm 4 shows the process of fine-grained clustering, which contains several steps. First, we need to generate a signature matrix based on all batches. In each batch, every element is the node index of the original graph and is hashed by a hash function $f_n()$ (line 3). The m_{in} represents one mini-hash value of the current batch (line 4). We use several hash functions to generate multiple min-hash values for each batch (line 6). These values can be expressed as a signature matrix (line 8). An illustration of this process is shown in Fig. 4.4. Note that we sample the indexes in each batch in the coarse-grained clustering phase so that we do not need to hash every index value in the batch and the generation of the matrix can be efficient even with multiple functions. After generating the signature matrix, we need to hash the signature matrix further. As shown in Fig. 4.4, each batch is split into multiple bands (two bands in the example), and the content in each band is mapped to a bucket by the general hash function (line 13 in Alg. 4). If two batches are mapped into the same bucket for a band, their min-hash values are the same in this band and thus become a similar pair.

Next, we go through all the bands and generate a similarity graph where edges contain weight (line 15 in Alg. 4). Specifically, each inference batch is a node in the graph; if two batches are a similar pair in one band, they have a connection, and the weight of the connection increases correspondingly. Then the weighted graph is clustered into ψ parts using Metis [54]. Note that ψ is a pre-defined threshold. After clustering, the batches in the same cluster are arranged adjacently.



(a) Accuracy based on various sampling ratios.



(b) Comparison between single ratio and mixed ratio.

Figure 4.5: Inference accuracy with different sampling ratios.

4.3.3 Parameters of LSH

The LSH-based algorithm contains several important parameters, including the number of hash functions, the number of bands b, and the number of rows each band contains r. We set the number of hash functions at 128. Suppose the similarity of two sets is s, then the possibility that they are the same pair at least one band is $1 - (1 - s^r)^b$. We calculate the b and r by minimizing the sum of false positives (sets should have the same bands, but do not) and false negatives (sets should not have the same bands but do) even with a small s, similar to the settings in [55]. Another parameter is ψ, which represents the number of clusters. Basically, a larger ψ generates more clusters, which may lead to high computational complexity. We set ψ as one-twentieth of the number of batches by balancing inference efficiency and clustering overhead.

4.4 Adaptive Sampling

Based on LSH-based clustering, we propose to further reduce graph data loading time by adaptive sampling.

4.4.1 Observation

We use the Reddit dataset as an example. We randomly sample a fixed portion of edges for each batch and show inference results in Fig. 4.5a. We find that some edges are redundant since non-sampling and sampling 95% edges have almost the same accuracy. The accuracy gap is observed when the ratio is 0.9 and becomes more obvious when the ratio is 0.85. A similar observation has been also reported by DyGNN [56]. The authors claim that some neighbors' edge information is useless in the neighborhood aggregation phase. The authors further use experiments to verify that some similar neighbors do not contribute extra information to the nodes, and a similarity threshold-based method is proposed to filter out similar neighbors. However, this method of reducing edge redundancy needs to calculate and compare each neighbor's similarity for every node, which is time-consuming.

We conduct additional mix ratio experiments to reduce the sampling ratio of nodes with a degree greater than 120, which is set to 0.9 when the basic ratio is 0.95, and 0.85 when the basic ratio is 0.9. Figure 4.5b shows the accuracy results based on single and mixed ratios. We can see that the lower ratio on high-degree nodes does not reduce the accuracy compared with these single-basic ratio settings. That means the single ratio-based simple sampling method can not reduce the redundancy well since there is still much redundancy with the high-degree nodes. Besides, it may eliminate some valuable connections during the sampling, thus causing the accuracy to decrease even with a quite large sampling ratio (see the results in Fig. 4.5a). After all, the single ratio sampling strategy is on the batch level, which supposes all neighbors have the same importance to the target node. However, compared with the nodes with a large degree, some nodes have limited neighbors, and each of them could be important and can not be eliminated.

Table 4.2: The information in datasets, the letter "(m)" stands for multiple class classification.

Dataset	Nodes	Edges	Average degree	Features	Classes	Train/Val/Test
Reddit	232,965	114,848,857	50	602	41	0.66/0.1/0.24
Yelp	716,847	6,977,410	10	300	100 (m)	0.75/0.1 /0.15
Amazon	1,598,960	132,169,734	83	200	107 (m)	0.85/0.05/0.10
OGB-products	2,449,029	61,859,140	25	100	47	0.08/0.02/0.9

4.4.2 Adaptive Sampling

Based on the above observations, we propose a simple but efficient node-level adaptive sampling strategy to reduce redundant edges. It contains several steps. First, we re-index all the nodes according to their degrees; a small degree node will have a small index. This operation will help make the nodes in the same batch have as similar degrees as possible when we generate batches. Since deciding the sampled neighbors for each node is time-consuming, we turn to make the nodes in each batch have similar degrees and set them with the same sampling strategy. Second, we divide the batches into two parts according to their nodes' average degree with a pre-designed threshold. If the degree is below the threshold, the batch will belong to the "full-sample" part. Otherwise, it belongs to the "tune-sample" part. The batches in the "full-sample" part keep all their nodes, while batches in the "tune-sample" part conduct neighbor sampling. The number of neighbors sampled is tuned by the average degree of the batch. Specifically, the number grows as the increasing of batches' average degree. This is reasonable since nodes with large degrees should sample more neighbors to control the sampling bias.

4.5 Evaluation

4.5.1 Experiment Settings

We prototype our system based on the Ubuntu system with Intel i7-10700 CPU and Nevada Geforce RTX 1080 GPU. We use PyTorch and DGL [37] graph

process package. Four widely used graph datasets are chosen as summarized in Table 4.2. We run graph inference in a batch manner with the GraphSAGE model [5, 6]. We set the batch size to 1000, as the default setting. The threshold λ, which represents the internal size in coarse-grained clustering, is set as 20,000 empirically. We compare our system with three baselines:

1. The original inference process (OIW): This is the basic inference operation with mini-batch while having no optimized strategies, as the default setting in DGL [37].

2. Caching according to node degree (CAD): This is the caching strategy proposed in PaGraph [6]. The nodes with higher degrees have higher priorities to be cached in GPU memory. Since the cache size usually is small, we suppose the cache-based solution can only cache 200 MB for all datasets. The other cache size conditions will be further discussed in the following subsection.

3. Simple reusing of the repeated nodes without reordering (SRW): We also consider reusing the repeated nodes among batches according to the order of the default inference setting.

4.5.2 Overall Results

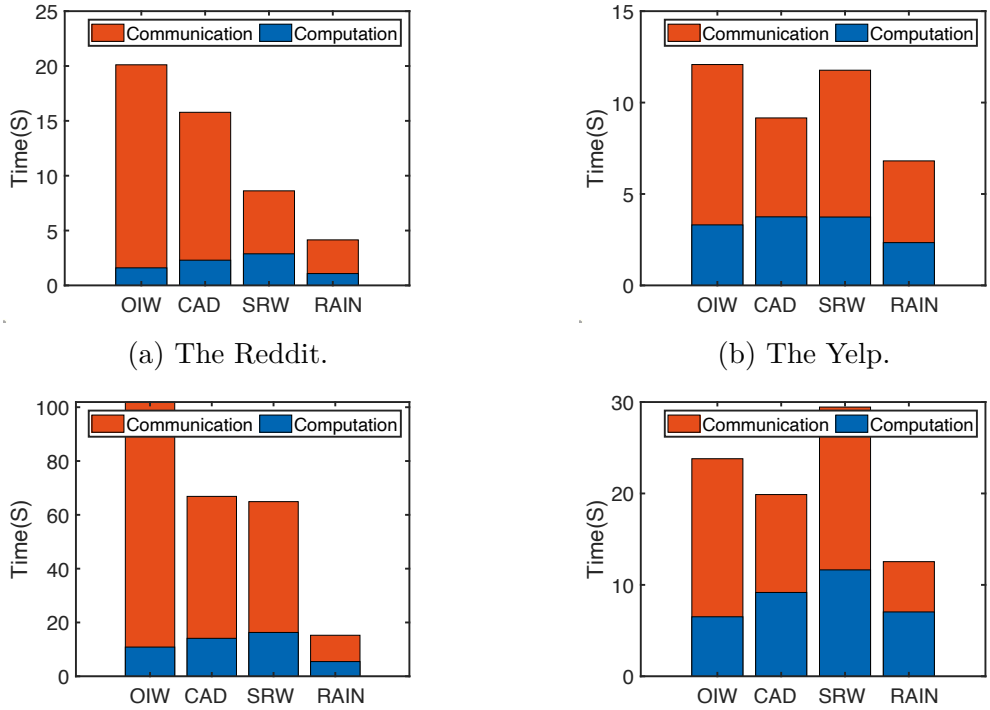
The overall results of the time cost of inference are shown in Fig. 4.6. Note that the pre-processing time is not contained since the inference process is conducted periodic [7,8] and thus can be considered as an off-line scenario. We can see that our system has a lower time cost compared with all other baselines. Specifically, we are faster by about 4.5X, 1.8X, 6.8X, and 2.0X over the Reddit, Yelp, Amazon, and OGB-products datasets compared with the original solution. The improvement is related to the average degree of the dataset. For example, the improvement on the Yelp dataset is the smallest, and the dataset has the smallest average degree. The Amazon dataset has the largest improvement with the

largest average degree. The results are reasonable since a large average degree means more redundant neighbors and more severe communication overhead, and our system can significantly reduce this kind of overhead.

As for other baselines, our system is faster by about 2.1X compared with SRW on the Reddit dataset, 1.72X compared with CAD on the Yelp dataset, 4.2X compared with SRW on the Amazon dataset, and 1.6X compared with CAD on the OGB-products dataset. The SRW takes more time than CAD on the OGB-products and Yelp datasets because these two datasets have a limited average degree, and the feature size of nodes is also tiny. Hence, the benefit of data reuse is less than the overhead from data replacement and the logical process of these two solutions. Note that we not only reduce the communication time but also reduce the computation time. CAD and SRW's computation times are larger than OIW's because of the extra logical process, while **RAIN** has a lower computation time, and the reduction comes from the benefit of adaptive sampling.

The results of accuracy are shown in Table. 4.3. We can see that **RAIN** has a negligible accuracy decrease, from 0.0003 to 0.0007. The minor decrease verifies the effort of the proposed adaptive sampling method. Specific sampling is a trade-off between accuracy and efficiency. We will discuss the details of the trade-off in the following subsection.

Although our system focuses on the scene that allows us to pre-process the data in idle time for the reuse of as much data as possible, we claim that **RAIN** still achieves fast pre-processing rather than a long preliminary preparation time. We record the total time cost (including pre-processing time and accuracy statistics time) over four datasets among different solutions, and the results are shown in Table 4.4. We can see that **RAIN** achieves time reduction on all datasets, even considering the pre-processing time, and the improvement can be across the order of magnitude. For example, we reduce the time cost from 116.26 seconds to 34.89 seconds on the Amazon dataset.



(a) The Reddit.

(b) The Yelp.

(c) The Amazon.

(d) The OGB-products.

Figure 4.6: Overall time costs for different datasets.

Table 4.3: The comparison of accuracy.

Dataset	Reddit	Yelp	Amazon	OGB-products
Original accuracy	0.9513	0.6161	0.7510	0.7018
Accuracy in RAIN	0.9507	0.6158	0.7504	0.7011
Decrease	0.0006	0.0003	0.0006	0.0007

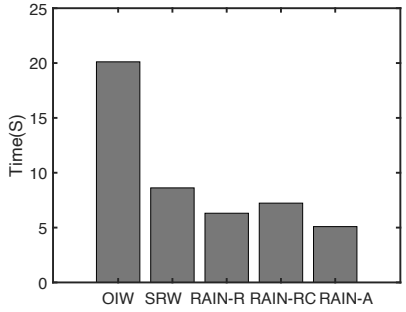
4.5.3 The Effectiveness of Two strategies

To fully evaluate the effectiveness of the LSH-based workload clustering and the adaptive sampling strategies, we also conduct some ablation experiments for comparison. The first two baselines are still OIW and SRW. The third solution is our system with only the clustering part, RAIN-R. The fourth is our system with only a clustering part, and that part only contains the coarse-grained classification, named RAIN-RC. The fifth is the RAIN with only an adaptive sampling part, RAIN-A.

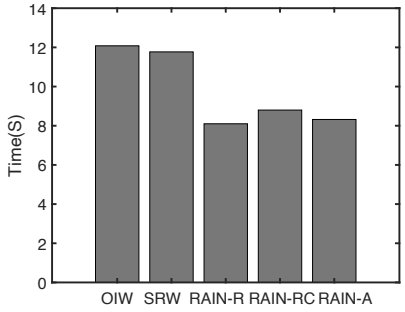
The results of the inference time on four datasets are shown in Fig. 4.7. We can see RAIN-R achieves about 30% to 70% improvement compared with OIW on the Reddit, Yelp, and Amazon datasets. For the OGB-products dataset, RAIN-R

Table 4.4: Time cost in on-line scene.

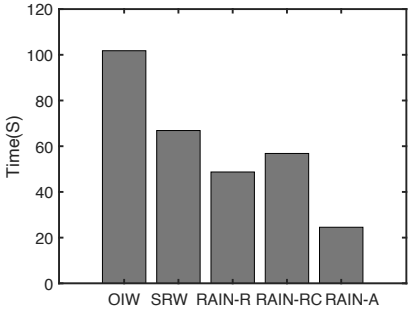
Dataset	Reddit	Yelp	Amazon	OGB-products
OIW (S)	25.73	18.20	116.26	41.74
CAD (S)	21.39	14.72	82.65	38.24
SRW (S)	13.68	15.94	86.79	40.31
RAIN (S)	**10.02**	**13.78**	**34.89**	**35.20**



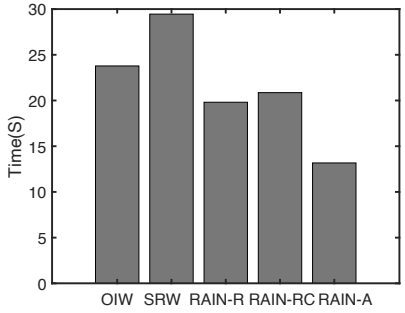
(a) The Reddit.



(b) The Yelp.



(c) The Amazon.

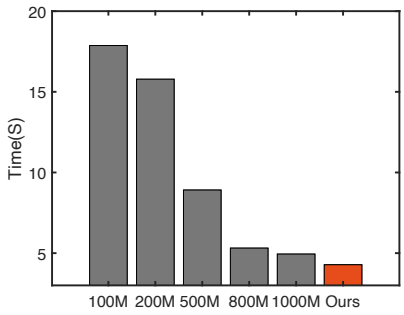


(d) The OGB-products.

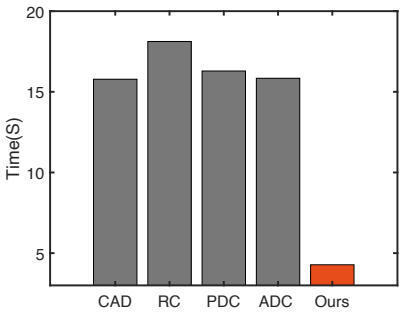
Figure 4.7: Time costs with various modified solutions for different datasets.

achieves about 11% improvement compared with OIW because the dataset has a limited average degree and feature dimension. The significant improvement verifies the effort of the LSH-based clustering strategy even without the proposed adaptive sampling strategy.

When we compared with SRW, which simply reuses data without clustering, RAIN-R still achieved about 10% (Yelp dataset) to 30% (Amazon dataset) improvements, which means the clustering operation is important and our LSH-clustering achieves both speed and effectiveness. Note that RAIN-RC always takes more time than RAIN-R, which means the coarse-grained clustering is insufficient. The comparison between OIW and RAIN-A represents the effectiveness



(a) Time costs of various cache sizes in CAD and ours.



(b) Time costs of various cache-based solutions and ours.

Figure 4.8: Time cost of baselines with various parameters.

of the adaptive sampling strategy, which can reduce about 30% to 70% time cost among four datasets.

4.5.4 Variants of the CAD Baseline

There are some parameters or variants in the caching-based baseline that may affect their performance when compared with **RAIN**. With the Reddit dataset, we conduct more experiments to investigate the time cost under different parameters in CAD and different variants of CAD.

Size of the cache in CAD

The size of the cache is the critical point in the CAD. We conduct more experiments to record the time cost of CAD under different cache sizes, as shown in Fig. 4.8a. One pronounced tendency is that the time cost decreases as the cache size increases. However, our solution is still in the lead even with a 1000M cache. The total graph is almost all cached in such a large cache size. However, when we process a vast graph with batching realistically, the graph cannot be completely loaded into the GPU memory, and the cache size should be limited. Otherwise, we can directly increase the size of the batch to accelerate the process. The experiments verify the effort of our proposal in this realistic setting.

Variants of CAD

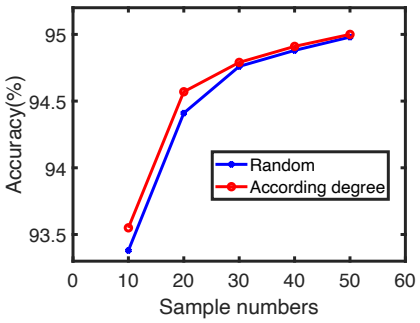
Besides the degree-based cache in CAD, we also conduct experiments to compare some other caching strategies, including random caching (RC, random choosing some nodes for caching), period dynamic caching (PDC, change cached content every fifty batches), and all-hit dynamic caching (ADC, dynamic change the cached content to ensure that all needed nodes are in the cache). The time costs of these caching solutions are shown in Fig. 4.8b. We can see that all these variants have similar time costs to CAD. RC takes more time than CAD since the random strategy has a lower hit ratio. PDC should have a higher hit ratio, but the overhead of cache replacement is greater than the reduction in data loading. A similar overhead is also severe in ADC.

4.5.5 Design details of RAIN

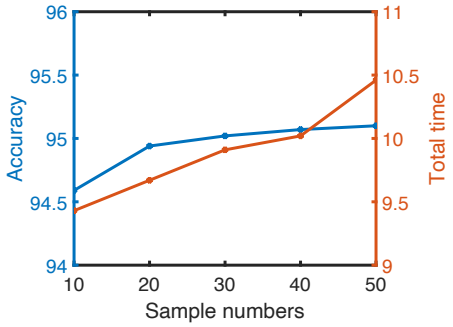
There are some parameters in the **RAIN** that may affect the performance; here, we use the Reddit dataset as an example to discuss the relationship between the performance and some parameters.

About re-index with degrees in adaptive sampling

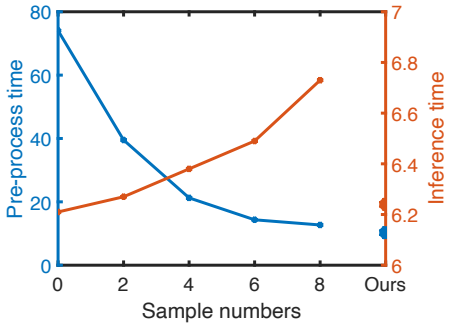
In the proposed adaptive sampling part, we make the target nodes in the same batch have a similar degree. This operation can be simply achieved by re-indexing the target nodes according to their degree, from small to large. It can help us eliminate the influence on accuracy when we conduct the same sampling strategy for the nodes in the same batch. We designed a simple experiment to verify the benefit of this operation. We specify that batches with less than 50,000 points are not sampled, and for batches with more than 50000 points, each target node sample N neighbors. N is set as 10, 20, 30, 40, 50, and the results of accuracy are shown in Fig. 4.9a. The accuracy increases as the increased of sampled neighbors. Compared with the random index, indexing according to degrees always has higher accuracy. The two lines tend to be the same when all neighbors are sampled.



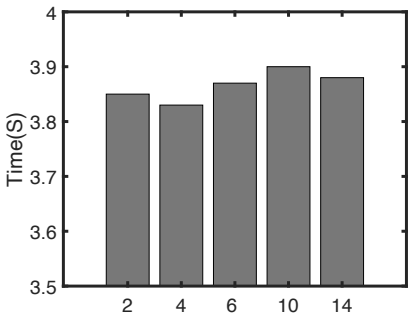
(a) Accuracy of random and reordered indexes.



(b) Accuracy and total time cost when sampling various neighbors.



(c) Time cost for pre-processing and inference when sampling various indexes for LSH.



(d) Time cost for inference when clustering various classes with LSH.

Figure 4.9: The influence of some designs in RAIN.

About sampled neighbors in adaptive sampling

The above experiments are just used to verify the effort of re-index. We still need to decide on the specific sample strategy for adaptive sampling. In the **RAIN**, we split batches with fewer than 10000 nodes into the "full-sample" set and others into the "tune-sample" set. The latter can be further divided into four sets, including $S1$ whose nodes are not larger than 50000, $S2$ whose nodes are not larger than 10000, $S3$ whose nodes are not larger than 150000, and $S4$ whose nodes are larger than 150000. We set the target nodes in $S1$ sample 10 neighbors and the others sample 10 more on top of the previous one, i.e., 20 for $S2$, 30 for $S3$. We also set $S1$ start at 20, 30, 40, and 50, and other sets still sample 10 more on top of the previous one for comparison. The sampled neighbors decide the total time cost and final accuracy, and the results are shown in Fig. 4.9b. We can see from the two lines that a higher number of neighbors means higher accuracy while needing more time. A small number of neighbors can complete the inference process quickly while causing an accuracy decrease. In the **RAIN**, we set $S1$ to start at 40, which has a good trade-off between time cost and accuracy.

About sample index in LSH

After coarse-grained clustering in **RAIN**, we conduct the LSH process for each batch. This process is time-consuming since we need to go through every node. We propose to sample the indexes for the batches in the same coarse cluster for time-saving. We also give some experimental results to verify the effort of this index sampling design. Since this sampling is not related to the adaptive sampling part, we use all neighbors here. We experiment with various solutions, including no sampling (use all indexes to do LSH), sampling every two indexes (choose one index to do LSH for every two indexes), sampling every four indexes, sampling every six indexes, and sampling every eight indexes. Their trade-off is shown in Fig. 4.9c. A low sampling ratio can significantly reduce the pre-processing time cost on LSH since only a small part of the indexes are involved; however, it has

a negative effect on the results of LSH-based clustering and increases the final inference time.

As a comparison, with the same amount of nodes, our solution conducts coarse-grained clustering first to split batches into three clusters according to their size. And then sample their indexes with different strategies in each cluster separately. For the cluster whose batches' sizes are small, we sample one index for every 10 indexes for all batches in the cluster. Otherwise, we sample one index for every 100 indexes. As bold nodes are shown in the figure, we achieve a better trade-off between pre-processing and inference time.

About cluster graph in LSH

In our LSH-based clustering, we need to generate a similarity graph and use the Metis [54] method to cluster the graph into ψ parts. We do some experiments to see the time cost under different settings of ψ. The results are shown in 4.9d. We ignore the coarse-grained clustering process to emphasize fine-grained clustering. We can see from the figure that a large amount of clustered classes do not mean a small time cost because when we have lots of clusters, the order among clusters may also become important. In our experiments, set ψ as 4 (about one-twentieth of the amount of the batches) has a good result on the Reddit dataset. For other datasets, we also set ψ as the one-twentieth of the amount of the batches.

4.6 Related Work

4.6.1 Various GNNs

Extensive studies show the effectiveness of GNNs in various applications like recommendation systems [7], temporal link prediction [43], and spam review detection [44]. A significant challenge in these business applications is handling the huge graph. Various strategies have been proposed to make the training process more efficient. GraphSAGE [5] lets each node sample a fixed number of neighbors

for each layer. VR-GCN [51] utilizes history neighbors' embedding to control variance and reduce the sampled neighbors to two nodes. Cluster-GCN [46] cluster the large graph into subgraphs and then complete convolution separately. Some works consider the cache aspect. For instance, PaGraph [6] has been proposed to cache the nodes with large degrees in the GPU memory. BGL [57] is a co-design of cache policy and neighbor selection to increase the cache hit rate.

Some works also focus on the inference process of GNNs. Zhou et al. [58] utilize channel pruning technology to reduce the input feature dimensions and cache some visited nodes' hidden features for reuse. However, the model needs to be retained after each pruning since the dimensions of features changed, and it does not introduce a specific caching strategy. This work is orthogonal to ours because we do not modify the model. Zeng et al. [59] propose to extract a subgraph for the target nodes in inference to decouple the depth and scope of GNNs. The benefit of the decoupling operation is the lower computation complexity in inference when conducting a deeper GNN layer on the limited subgraph. However, it still involves many nodes since the subgraph is extracted with hundreds of neighbors or all 2-hop neighbors.

4.6.2 Caching Strategies

It is not a new topic in data caching and reusing. However, the previous works have high complexity and are not the optimal solution in our case because of the unique character of the graph learning's inference. We will analyze the existing solutions in our scenario in this subsection. For the general offline caching problem, the Farthest-in-Future (FF) algorithm [60] can achieve the optimal result. The basic idea is to discard the information in the cache that will not be needed for the longest time in the future when the cache is full. FF is not optimal in our scenario because when queering each batch's nodes, the nodes can be obtained from the cache or directly from the main memory. We also call this characteristic as *bypassing* as in [61]. Besides the *bypassing*, the reordering operation is

also allowed in our case. In [62], the author also allows *bypassing* and considers query j may be served before i if $j - i < r$. Under the constraint of this kind of $r-$reordering, the author batches the queries first and gives the corresponding bound compared to the standard $r-$reordering. After batching operations, the author also gives the optimal offline algorithm, BMIN. The main idea of BMIN is described as follows. Suppose node d is queried and d is not in the cache. Let $In(d)$ be the index of the batch where the next unserved query to d occurs. Determine $In_{max} = MAX_{d' \in S} b(d')$, where S is the set of nodes in the cache currently. if $In(d) < In_{max}$, then load d into cache and evict any node d' with $In(d') = In_{max}$. BMIN has a similar principle with FF while considering the *bypassing* simultaneously. Although considering *bypassing*, BMIN is not optimal in our scenario since an inference batch in our setting already contains multiple elements.

4.7 Conclusion

In this Chapter, we propose an efficient inference system for graph learning, named **RAIN**. We consider reusing the repeated nodes among inference batches to reduce the redundant data loading, which takes a significant toll on the time cost of the complete inference process. The inference batches can be reordered to reuse as many nodes as possible. However, directly reordering batches is time-consuming. **RAIN** adopts an LSH-based two-level clustering scheme to quickly cluster the unequal-length batches and arrange the batches in the same cluster adjacently. We also propose an adaptive sampling strategy to sample the target nodes' neighbors according to their degree while having a minor influence on the final accuracy. We compare **RAIN** with various baselines, and the results verify the effort of the LSH-based reordering and adaptive sampling strategies.

Chapter 5

Efficient Transformer Inference using Masked Autoencoders

In addition to graph data, we also explore the sparsity of image data. Many edge devices with weak computing power collect image data that needs to be identified, but these devices may not have enough resources to conduct complex neural network identification. Therefore, the data often needs to be uploaded to a server for processing. We propose an offloading system that does not require computation on the edge device and only needs to transmit part of the image data to the server. The server can then recover the image and perform inference using a feedback-driven strategy designed to achieve content-aware transmission.

5.1 Introduction

There is a strong demand to deploy intelligent applications, e.g., object detection [63], data augmentation [64], and image recognition [65, 66], on mobile/IoT devices with various sensors. These applications are based on the inference operations of complex deep neural networks (DNN), which can hardly run on devices with limited hardware resources. This dilemma motivates broad-spectrum research on offloading DNN inference operations to edge servers or clouds. An ideal offloading policy should satisfy three requirements: (1) high inference accuracy;

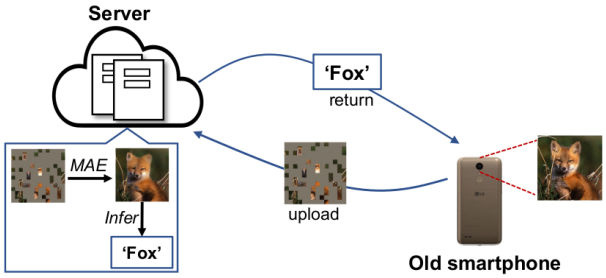


Figure 5.1: The basic MAE-based offloading scheme.

(2) low communication overhead, as mobile/IoT devices using wireless networks usually have limited network bandwidth; and (3) low computation overhead on the device side [67, 68].

Unfortunately, none of the existing work can achieve all three requirements at the same time. A straightforward offloading strategy sends raw data to the cloud [9], which eliminates computation at edge devices and achieves high inference accuracy by using powerful hardware in the cloud. However, since raw data have a large size, this simple strategy would incur a high communication cost. Some recent works have proposed data preprocessing techniques at edge devices to reduce communication costs. Such preprocessing techniques include DNN model splitting [11], input data compression [13], and input data filtering [14]. DNN model splitting is based on the observation that the output of some intermediate layers is smaller than the original input. Thus, we can trade running a few DNN layers at edge devices for a significant reduction in communication cost. However, running just a few DNN layers could be also a heavy burden for weak-edge devices. Furthermore, not all DNN models exhibit the feature of smaller intermediate data. Some DNN-based data compression methods also have a heavy computation overhead for edge devices [13]. A higher compression ratio can reduce the amount of data transmitted over networks, however, the inference accuracy could also be decreased if the data is over-compressed. Data filtering techniques select and send image regions, including target objects, instead of whole images. The commonly used MobileNet-SSD [69] filter demands about 1200 Million Floating Point Operations (MFLOPs) for 300×300 images [70], while some edge devices, like raspberry

pi-zero-w and raspberry pi-aplus, only support about 200 Million Floating-point Operations per Second (MFLOPS) [71]. Du et al. [67] have proposed a server-driven offloading method for video analysis. Edge devices transmit low-quality frames with reduced size to the server first, and then the server identifies target regions and requests a re-transmission of high-quality content within these regions.

There are additional works that formulate and resolve various offloading optimization problems. The issue of privacy is considered during model splitting in [72] and the energy consumption constraint is added in [73]. The deep Q-network-based offloading strategies are also proposed with consideration of channel conditions [74, 75]. and reinforcement learning is used to let each device make its own offloading decision [76]. However, these methods are tailored to specific tasks and lack a holistic perspective that simultaneously considers computation, communication, and accuracy.

In this Chapter, we propose a new approach to breaking the myth of the impossible trinity of DNN offloading. The basic idea is shown in Fig. 5.1. Edge device (e.g., an old smartphone) collects image data and randomly samples a small portion of image patches, and sends them to the server, which then uses a masked autoencoder (MAE) [16] to recover the image and conduct inference. Sampling is a simple operation with negligible computation overhead for edge devices. Since the sampling ratio could be very low, usually less than 30%, only a small amount of data need to be transmitted over networks, leading to low communication cost. MAE was originally designed for pre-training, and we exploit its powerful capability in image recovery for DNN offloading. Therefore, it is promising to achieve high inference accuracy with limited sampled data. Note that our method is orthogonal to conventional compression methods that encode image data using various transform strategies. The sampled data can also be further compressed by these methods.

Although the MAE-based scheme shown in Fig. 5.1 is promising, we are facing several critical challenges to make it work efficiently in practice. The first is to de-

termine how many patches should be sampled to guarantee a good recovery with high inference accuracy. More patches could be helpful for better image recovery while leading to higher communication costs. Especially, weak edge devices have insufficient hardware resources to run complex algorithms for content recognition to make decisions. We address this challenge by designing a two-round offloading scheme for inference, named A-MOT (Adaptive MAE-based Offloading for Transformer inference). A-MOT contains an image selection process. Specifically, in the first round, edge devices randomly sample a small number of patches and send them to the server. If these patches are sufficient for recovery and obtaining inference results with high confidence, the server returns results and completes the inference service. Otherwise, several "important" patches are selected and requested by the server in the second round of offloading.

Second, we find that different images require different numbers of patches for correct inference. Some images with simple contents can be well recovered by MAE even with a few patches, but more patches are needed for complex images. Offloading efficiency could be further improved if this feature is well exploited. Since both edge devices and the server are unaware of image contents before the first round of offloading, we let all devices offload the same amount of patches. In the second round of offloading, the server requests different amounts of patches for images, by using the information obtained by MAE and inference operation. However, this method is agnostic to SLO (service level objective), i.e., it determines the number of patches without considering network bandwidth. Thus, A-MOT has an SLO-adaptive design that can decide how many patches are transmitted in the second round of offloading for different images, given a traffic budget.

The final challenge is the high computational burden on the server. Although the two-round offloading scheme is promising in terms of reducing communication costs and increasing inference accuracy, the server has high computational overhead because it needs to run two inference operations for some images that need the second round of offloading. The commonly used inference models are Vision

Transformer (ViT)-based, which divides an image into multiple small patches to form a patch sequence. The most significant component of the models is the attention layer, where they calculate the attention value among each patch pair to generate new embeddings. The overhead of inference is thus positively related to the length of the patch sequence. The optimization method for language inference with the Transformer model in [77] does not work here since it takes advantage of the fact that language sentence lengths are naturally different, while images from the same device have the same size. To reduce the overhead, A-MOT has a lightweight inference operation for the second round of offloading. Instead of running a full inference with the complete patch sequence, the server lets newly received patches go through an encoder, which has the same attention layer as the inference model. This operation has low overhead because of the short input. Then, the embeddings generated by this encoder are combined with the ones from the first round. The combined results are sent to a decoder to generate the final output.

The main contributions of this Chapter are summarized as follows:

- We propose a two-round inference offloading scheme based on MAE so that weak edge devices can also achieve high inference accuracy with low communication costs.

- We design an SLO-adaptive strategy to maximize the inference accuracy with the constraint of limited network bandwidth.

- We reduce the computational cost at the server by proposing a lightweight inference operation for the second-round offloading.

The rest of this Chapter is organized as follows. We introduce the motivation in section 5.2. The system design is described in section 5.3. We evaluate the system in section 5.4 with various baselines. Section 5.5 discusses some related works, and Section 5.6 is the conclusion.

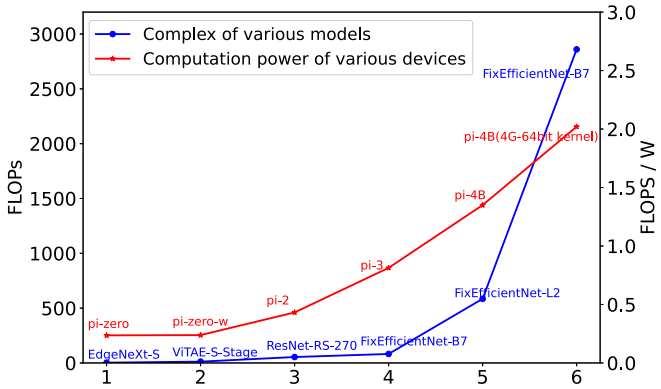


Figure 5.2: The development of models' complexity and devices computation power per energy unit.

5.2 Motivation

5.2.1 Limited Resources of Edge Devices

The popular deep neural networks are vital in dealing with data analysis [78]. However, edge devices could be rather resource-limited [67, 68], making it hard to run inference tasks with complex models. Fig. 5.2 shows the FLOPs of several commonly used models for image recognition and the FLOPS/W of the various raspberry pi devices [71]. The growth of computing capability on edge devices lags behind the increase in model complexity. For example, the computing power of *pi-4B(4G-64bit)* is about 2.0 FLOPS/W, which is around four times that of *pi-zero*, while the FLOPs of the *FixEfficientNet-B7* are about 2700, which is thousands of times that of the *edgeNetx-S*. This suggests that edge devices have struggled to efficiently run growing models. Offloading has therefore been widely exploited for inference tasks. However, bandwidth is a scarce and even volatile resource [11, 79, 80]. The direct transmission of raw data may incur a significant delay. To reduce offloading traffic, some works use small-size selectors to choose and transmit critical regions in the image. However, these selectors are still resource-intensive for resource-limited edge devices. There are also efforts to run a part of the inference model on edge devices and upload the intermediate output to the server for the rest of the inference [11]. These DNN partition-oriented works hypothesize

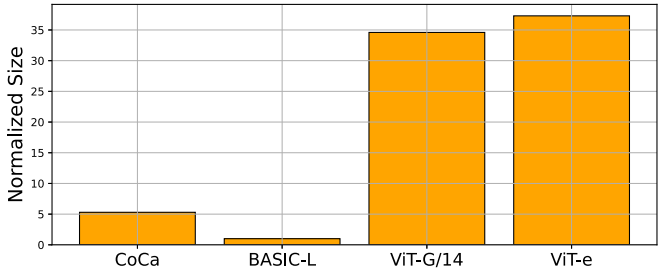


Figure 5.3: The output size of various state-of-the-art inference models.

that some intermediate layers in the neural network may have a smaller output size than the raw data, thus saving bandwidth. However, lots of models do not show this characteristic. We list four models with the highest accuracy on the commonly used ImageNet dataset, including CoCa [81], BASIC-L [82], ViT-G/14 [82], and ViT-e [83]. These models are all attention-based with an isotropic architecture [84], which means all main layers contain heavy attention operations and have the same output size. Figure 5.3 shows the normalized output sizes (with the size of the input image set to one) of intermediate layers for each model. Coca, ViT-G/14, and ViT-e have large intermediate data. For BASIC-L, the size is similar to the input image. However, this output size is obtained after running more than 40 layers, and such a computational burden cannot be afforded by weak-edge devices.

5.2.2 Possibility of MAE-based Bandwidth-saving

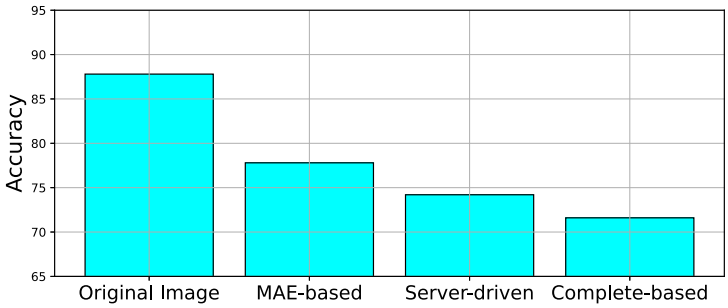
Due to MAE's potent reconstruction capabilities, it is reasonable for the device to randomly mask images before transmitting the preserved data to the server for MAE reconstruction and inference. Two related competitors exist. The first option is to replace the MAE with other ways to reconstruct the image, such as the complete-based method in [85]. The other server-driven method [67] initially reduces image resolution on the device and then utilizes server-side computing to identify the target object in the image before retransmitting high-level pixels.

We conducted some preliminary experiments to compare the efforts of different solutions. We employ a subset of the ImageNet dataset for accuracy testing with 10 classes and 50 images for each. We apply the Large-ViT-based encoder in

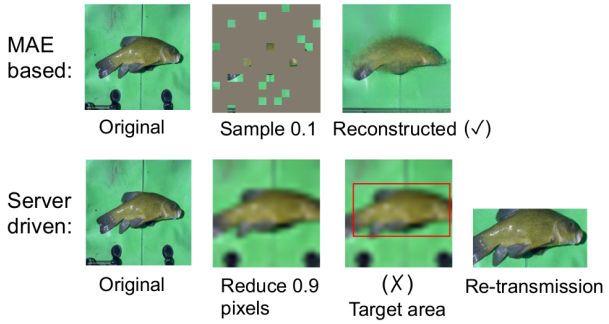
MAE and the DeiT-Small inference model (a variant of ViT) [86]. To conserve bandwidth, all three systems transport only 30% of the original data volume in total. For the server-driven method, we only retransmit part of the target area and prioritize high-attention patches to match the budget constraint. Figure. 5.4a illustrates their accuracy. The MAE-based method has the best performance among all bandwidth-saving solutions. For the complete-based works, they target filling reasonable content with photorealistic appearance into the missing regions [85, 87]. The target is different from our reconstruction process and will generate unrelated content when we have a large mask ratio. For the server-driven work, we give an example as shown in Fig.5.4b. With a sample ratio of 0.1, detecting the fish in the masked image is difficult, but reconstruction makes it simpler, and the reconstructed image can be accurately identified. The server-driven method, on the other hand, must retransmit the target region, which is greater than the 0.1 transmission ratio in the MAE-based method. The comparison verifies that MAE may study the deep semantics of a masked image and be utilized for bandwidth savings in offloading. Note that there is a tradeoff between communication and accuracy in our scenario. However, it is difficult to mathematically formulate the relationship between increased communication and improved accuracy since we have an image recovery process.

5.2.3 Different Images Require Various Mask Ratios.

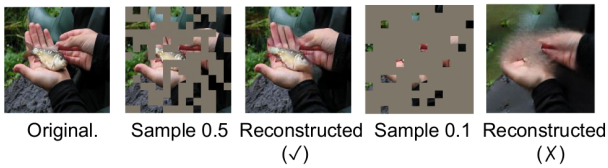
Although the MAE-based method has the best performance among all bandwidth-saving solutions, the challenge, however, is that the MAE-based recovery method still has a significant accuracy gap in comparison to the raw image in Fig. 5.4c. Allowing each image to have its optimal sampling ratio for a given transmission budget is a potential method for further enhancing accuracy. We investigated the MAE-based method using additional image instances. As depicted in Fig. 5.4c, the image has a complex background, and the object fish in the image is quite small, which cannot be identified with a sample ratio of 0.1 like in Fig. 5.4b.



(a) The accuracy of different solutions



(b) An image example between two solutions



(c) Complex image demands more data for reconstruction

Figure 5.4: MAE can be used for bandwidth-saving in offloading.

When we send more data to bring the mask ratio up to 0.5, the reconstructed image is much clearer, and the fish can be correctly identified. The results show that since the contents of different images are different, their mask ratios should be varied to achieve content-aware transmission and bandwidth savings. However, due to limited processing resources on edge devices, we cannot know the best mask ratio for each image, and giving all images the same sample ratio results in duplicate transmission and reduced accuracy.

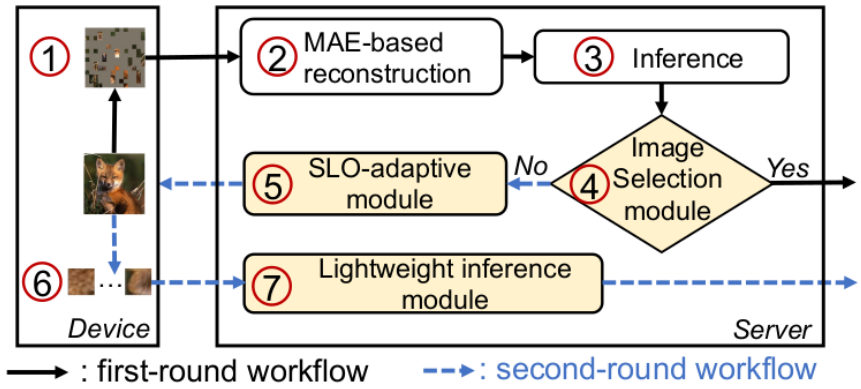
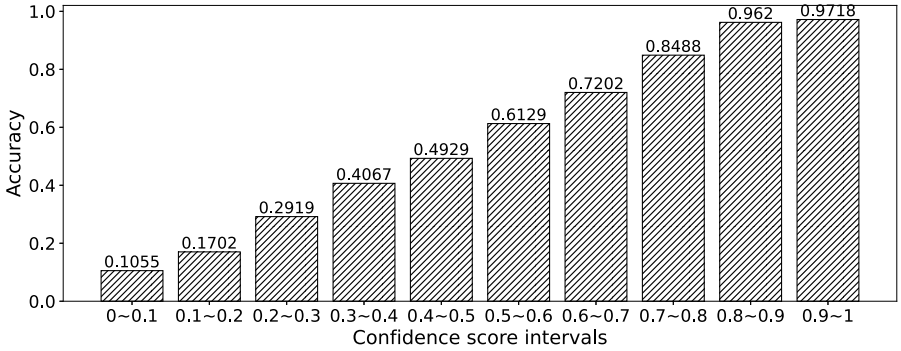


Figure 5.5: System overview.

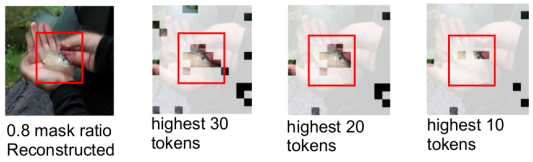
5.3 System Design

5.3.1 Overview

Fig. 5.5 illustrates the process of the proposed two-round offloading system, A-MOT. There are three key designs in the A-MOT, including the image selection module, the SLO-adaptive module, and the lightweight inference module. Specifically, the device randomly samples the data on each image (①) and transmits it to the server. The server then reconstructs the images using the MAE model (②) and conducts inference (③). The confidence score is compared with a threshold in the image selection module. If the score exceeds the threshold, the inference result is direct output; otherwise, high-attention tokens are chosen to prepare for the second round of offloading (④). The SLO-adaptive module (⑤) then determines and requests additional data volume for each image within the given traffic budget adaptively. The short extra-transmitted data sequence (⑥) will be encoded (⑦) independently in the lightweight inference module, and we combine the output of the encoder and the embeddings of the first inferences to send to an attention-based decoder and obtain the final results. We present the details in the following.



(a) The relationship between confidence scores and accuracy



(b) High-attention regions in the image are focused on the target object

Figure 5.6: Some crucial characteristics in the feedback process.

5.3.2 Two-round Offloading with Image Selection

The MAE model can help to recover the sampled data transmitted by the device in order to increase accuracy. It is critical to decide the sample ratios on the device side for each image since the ratio determines the communication costs and the recovery effect. However, weak edge devices have no sufficient hardware resources to run complex algorithms for content recognition to make decisions, and a one-time transmission with a content-random sample operation is not enough to obtain the desired accuracy. We choose to present a two-round offloading scheme. The scheme transmits a low ratio of data for all images first. If these patches are sufficient for recovery and inference, the server returns results and completes the inference service. Otherwise, a second round of offloading is needed. However, it is not a trivial issue to decide whether the second offloading is needed since we cannot know if the image is correctly recognized.

The image selection module is proposed to determine whether the second offloading is needed by exploring the potential of confidence scores. To study the correlation between confidence score and precision, we use the same settings in-

troduced in the previous section. We divide all image samples into ten groups based on their inference confidence scores. The accuracy of the samples at each interval is recorded as in Fig. 5.6a. We find that the confidence score correlates highly with accuracy. A similar observation is also made in [88]. Consequently, we can design a threshold-based strategy to focus on the additional data-required images based on the confidence score. The threshold is previously determined by the expected level of precision since it has a direct relationship with accuracy. If the confidence score exceeds the threshold, we finish the inference by generating the result and sending it back to the device; otherwise, the server will demand a second transfer to run the inference process with supplemented data to improve accuracy.

The selection module also selects the contents of the image for the second transmission. Instead of random content chosen, we find that the attention results of the initial inference are related to the image's content. An example is shown in Fig. 5.6b. After inferring the reconstructed image with a 0.8 mask ratio, we choose the 30/20/10 tokens that have the highest attention values on the classification token. Note that there are multiple attention heads in the ViT; we average the attention values of all heads in the last attention layer as the final value. We can see that these tokens are most focused on the target object, which is in the red square. Because of this, we can send the high-attention tokens during the second transmission to improve accuracy. Note that two-round offloading is usually enough to obtain good performance with the given communication budget. If we increase the rounds without limitation, the performance may even decrease since it will be complex to decide the bandwidth budget allocation among rounds, and the increased inference operation will bring an extra computation burden to the server.

Figure 5.7: Different images demand various r_2 for the second transmission.

5.3.3 SLO-adaptive Module

Different images require different numbers of patches for correct inference. Assume r_1 and r_2 are the transmission ratios compared to the original image size for the two offloading rounds mentioned above. We set all images to have the same r_1 since we were unaware of their contents at first. However, r_2 should be varied since all images have their own unique content. Fig. 5.7 shows an example where two reconstructed images have the same r_2 as 0.1. The upper image has a simple background, and the re-transmitted content can catch the central part of the target bird, whereas the other image with a complex background requires a larger r_2 of 0.5 to cover the target. In fact, the traffic budget can be limited and constant, so we need to require different images with various amounts of data and achieve the SLO (service level objective) by taking the given traffic budget into account.

We propose the SLO-adaptive module to overcome the challenge by exploring the relationship between the confidence score and the needed r_2. We first obtained the smallest r_2 values for each image that could be correctly identified by running the inference multiple times. The distributions of r_2 in various confidence score intervals are shown in Fig. 5.8 when the first-time transmission ratio r_1 is set as 0.6. We can see that a higher confidence score means a small and more concentrated distribution of r_2. This distribution is consistent among all images according to our experiments, which can be used as a priori knowledge.

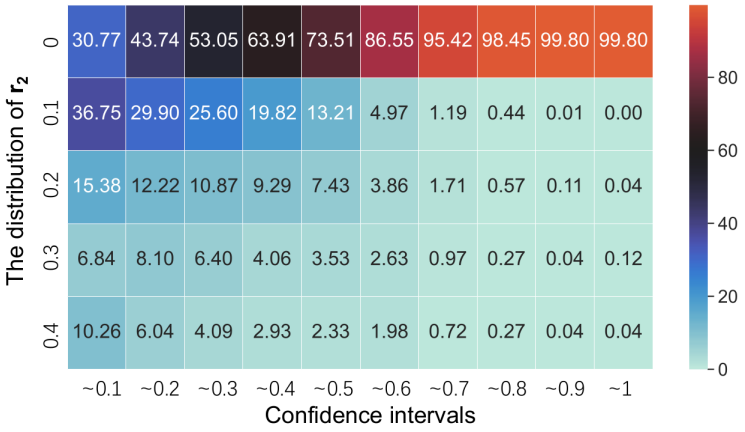


Figure 5.8: The distributions of r_2 in each confidence score interval.

The above distribution can be represented as $p_{i,j}$, which means the percentage in the $i-th$ raw and $j-th$ interval. Note that $\sum_{i=0}^{R} P_{i,j} = 1$, where R is the number of selections of r_2. Since the only information we know is the confidence score of the images, we give the images in the same confidence interval with the same r_2. Then we need to decide various r_2 for each interval. With the distribution information, we decide r_2 by solving a resource allocation problem. We formulate the problem first. Suppose the total transmission budget is B and we have a given first-time transmission ratio r_1. The N images in a batch will be naturally distributed in various confidence intervals after the first-time inference. For the $j-th$ interval, the amount of samples is denoted as d_j. Suppose x_j is the second-time transmission ratio (r_2) for the $j-th$ intervals. We have the following formulations:

$$maximize \sum_{j=0}^{K} \sum_{i=0}^{x_j} p_{i,j}; \tag{5.1}$$

$$\sum_{j=0}^{K} x_j \cdot d_j \leq B - r_1 \cdot N \tag{5.2}$$

Our target is to maximize the accuracy, which means maximizing the sum of the probabilities that the images can be correctly classified in each interval as formulated in (5.1). K is the number of intervals. The transmission budget constraint is given in (5.2).

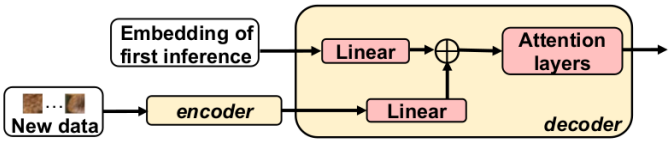


Figure 5.9: The process of the lightweight inference.

5.3.4 Lightweight Inference Module

The newly transmitted data contains important information about the image; however, it will be time-consuming if we conduct the reconstruction process again or implement another complete inference operation with the augmented image. Here we propose the lightweight inference module to reduce the computational cost at the server as shown in Fig. 5.9. First, the data sequence for the second round of offloading is independently encoded by the encoder, and the encoder is configured with the same attention layers as the initial inference model. The overhead of the encoder is low due to the short input sequence. However, the accuracy will be decreased if we only use the new, short data. Then, the lightweight inference module combines the output of the encoder and the embeddings of the first inference. Specifically, they go through the linear layers independently, and the feature values corresponding to the same positions in the image will be added together to generate a new embedding sequence. Finally, the new sequence goes through an attention-based decoder to obtain the final classification result.

The overhead of the second-time inference is proportional to the length of the newly transmitted data and the complexity of the additional decoder. The former is typically shorter than 20% of the original length on average. For the decoder, we set it with three attention layers, and each layer has the same size as the attention layer in the encoder. The complexity of the decoder is about 1.2 GFLOPs. The prior inference model (DeiT-Samll) with the original image input achieves 4.6 GFLOPs. This indicates that the total overhead of the lightweight inference will not exceed 50% of the original inference.

5.4 Evaluation

5.4.1 Experiment settings

We deploy an A-MOT server on an Ubuntu system with an Intel i7-10700 CPU and a Nvidia Geforce RTX 3080 GPU. We use the ImageNet-1K dataset (1000 classes, each with 50 images for the validation, and around 1200 images for the training) for inference. We apply the Large-ViT-based encoder in MAE [16] for the image recovery and the DeiT model (a variant of ViT) [86] to infer the images. The confidence threshold to select the images for the second round of offloading in the image selection part is set at 0.8. With various total traffic budget conditions, we compare our system to the following baselines in accuracy, which also don't require a lot of computing on the device side:

- **Non reconstruction (Non-R).** A natural baseline is to directly infer the transmitted data without any reconstruction operations.
- **Server-driven transmission (SDT).** The DDS [67] is configured to transmit a low-quality image to the server and retransmit high-quality content of the chosen areas again for video analysis. We apply this method to transmit low-quality images first and then offload high-quality content of the target area in the image. We choose the best performance among various combinations of r_1 and r_2 as the final result.
- **Image Super resolution (ISR).** Another possible solution is to transmit a low-quality image to the server, and the server reconstructs a high-resolution (HR) image with a trained model, which is also called the super-resolution (SR) method. The HR image can be further inferred. We also follow this principle to generate HR images with the commonly used SR model proposed in [89].

Besides, we also compare A-MOT with some computation-required solutions in both accuracy and communication overhead:

- **Partition-oriented inference (PO)**. The inference model contains multiple attention layers, and each layer has the same structure. We let the device complete the first attention layer and transmit the intermediate results to the server in a split form like in [11].

- **Data filtering on the device (DF)**. The image can contain lots of redundant content, we let the device run a MobileNet-SSD [69] based model locally, to select the crucial area in the image first. Only the selected region is transmitted to the server, like in [14].

- **Model-based compression (MBC)**. After splitting the inference model between the edge device and the server, it also has an encoder-decoder scheme to compress the intermediate results to reduce communication [13].

5.4.2 Results

With Computation-free Baselines

Fig. 5.10 shows the overall results when compared with these computation-free baselines. The number on the x-axis represents the total traffic budget. Suppose 100% means to transmit all the raw images to the server, we compare the performances of solutions under various given traffic budgets (different percentages). Our solution improves the accuracy by even more than 10% at a 40% transmission budget compared with the Non-R solution. The gap is gradually reduced when the total transmission ratio gets larger, which is reasonable since all solutions should have similar performance when transmitting the total images. It is worth noting that in A-MOT, when the total budget is small (such as 40%), we allocate 10% of the raw data volume as the budget for r_2 in the second round of offloading, and it becomes 20% for the other larger budget settings. This heuristic can bring the best performance in our experiments, and we will discuss more details in the latter subsection. The SDT does better than the Non-R solution and still lags behind A-MOT since the MAE model in A-MOT can help to learn genuine visual

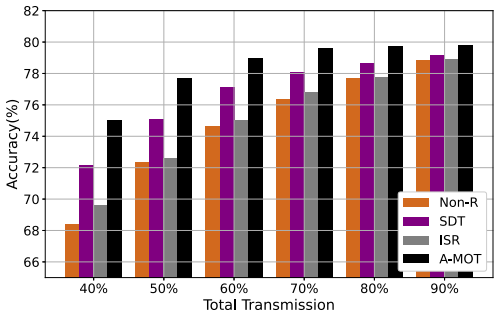


Figure 5.10: The overall results with computation-free baselines.

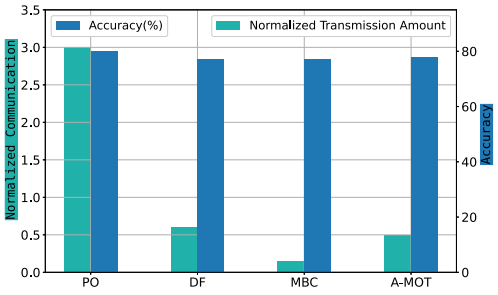


Figure 5.11: The results with computation-contained baselines.

semantics. The gap is also more noticeable when the total transmission budget is low. ISR achieves only a slight advantage over Non-R. Its one-time transmission strategy is far from content-aware offloading.

With computation-required Baselines

Consider transmitting the raw images with a communication cost of one, we normalize all communication costs to the raw data. Fig. 5.11 shows the results when compared with the baseline that contains computation on the device side. For computation, PO demands about 400 MFLOPs, DF demands about 1200 MFLOPs, and MBC demands about 500 MFLOPs, while our method (A-MOT) almost demands no computation on the device. For communication, PO has the highest data transmission requirements, and its accuracy is also the highest. A-MOT has a small data transmit requirement and the accuracy gaps between PO and DF are slight. MBC has a small communication overhead. When the device is weak, A-MOT offers significant advantages in terms of power savings while guaranteeing accuracy.

5.4.3 The Influence of Various Strategies in A-MOT

About Image Selection and SLO-adaptive Modules

There are some key designs in A-MOT, including the image selection module and the SLO-adaptive module. We chose some different settings in A-MOT to

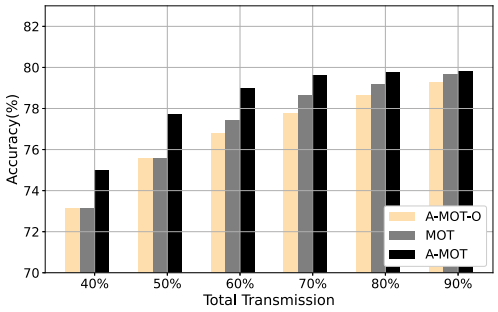


Figure 5.12: The Influence of Various Strategies in A-MOT.

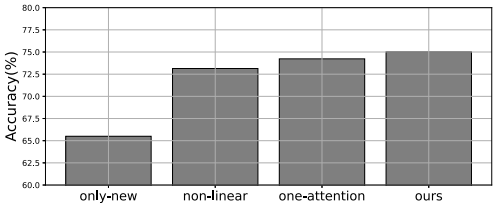


Figure 5.13: Accuracy of different settings in the lightweight inference modules.

evaluate the effort of the above designs.

A-MOT-O. We propose a two-round offloading scheme in A-MOT. We also compare the accuracy when there is only a one-time transmission process, called A-MOT-O. The images also need to be reconstructed and inferred, and the result will be directly transmitted to the device as the final outcome.

MOT. We propose the SLO-adaptive module to set different r_2 for different images. We also compare our solution to the same r_2 setting, which is our work in the previous conference version (MOT) [90].

The results are shown in Fig. 5.12. The OFTT solution and the MOT solution have similar accuracy when the total transmitted data is small, and the benefit of the feedback strategy in the MOT solution will be obvious when the data amount gets larger since it achieves content-aware transmission. Basically, the key designs can further help the mask-reconstruct scheme in A-MOT to improve the accuracy by about 2% when the traffic budget is limited.

About Lightweight Inference Modules

We also evaluate the lightweight inference module by comparing it with various alternative settings, such as only inferring newly offloaded data (only-new), combining the result of the encoder and embeddings from the first inference without linear layers (non-linear), and employing the decoder with only one attention layer (one-attention). Fig. 5.13 shows the accuracy results of these settings when

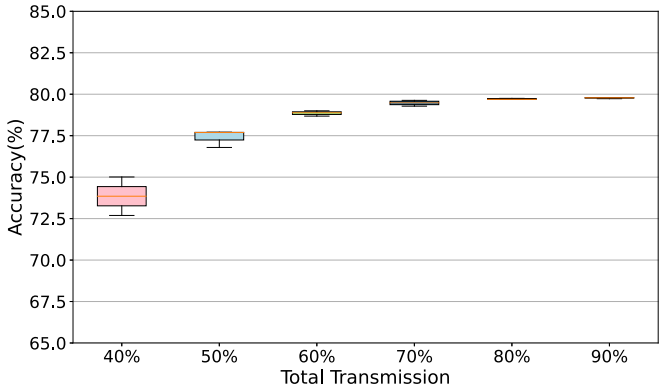


Figure 5.14: The accuracy and various r_1.

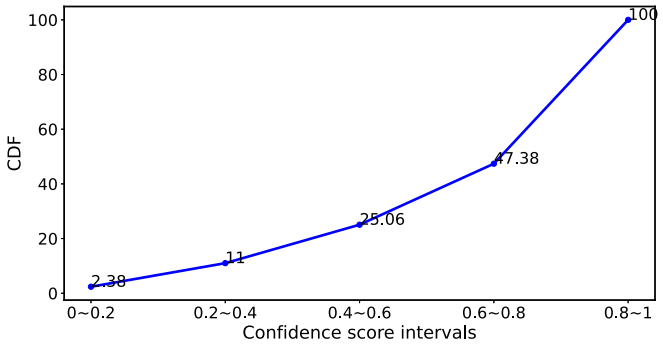


Figure 5.15: Percentage of samples that can be correctly classified in each interval.

the total traffic budget is 40%. We can see that using only new data can result in a significant decrease in accuracy. The non-linear method lags behind our solution as well. Usually, the accuracy is proportional to the number of layers in the decoder model; thus, the one-attention setting is insufficient to achieve the desired accuracy.

5.4.4 The Choice of r1

With a constant transmission budget, there are multiple r_1, r_2 combinations that can be chosen in A-MOT. Different r_1 can lead to various inference accuracy. Fig. 5.14 shows the distribution of accuracy among various r_1 with different transmission budgets. We can see that the accuracy has a more concentrated distribution when the total budget is large. In A-MOT, allocating 10% or 20% of the original data volume as r_2 for the second transfer can achieve the best accuracy. This small amount of r_2 also means a more lightweight inference for

the second round of offloaded data. As we mentioned before, even considering the extra attention-based decoder, the total overhead of the lightweight inference will not exceed 50% of the original inference.

5.4.5 The Choice of Threshold

In A-MOT, we set a threshold to decide if the images need a second round of offloading according to their confidence score. The threshold setting can help reduce transmission since we can obtain the output of the high-confidence image directly. We show the distribution of the raw image samples in the ImageNet data set among various confidence scores in Fig. 5.15. With a set threshold of 0.8 in A-MOT, the upper bound percentage of only once-time transmission samples can be about 53%. This means that many images only require once offload and once inference, allowing the limited budget to be focused on improving the identification accuracy of low-confidence images. The specific percentage of these one-time samples is related to the setting of the first-time transmission ratio r_1 since r_1 can decide the distribution of the confidence scores. The percentage is increased as r_1 becomes larger basically. In A-MOT, we have a larger r_1 when there is more traffic budget; thus, more samples will be inferred only once, and the resource will focus on the low-confidence image.

5.5 Related work

5.5.1 Offloading

For edge devices, their data can be trained with federated learning [91,92], and the data inference with offloading to the server also exhibits an increasing pattern. Li et al. [11] proposed to adaptive split DNN and model right-sizing with an on-demand inference framework to tackle the network latency. Shi et al. [72] further consider the privacy issue during the partition process. The energy consumption situations and task deadline constraints are also added to the partition problem

[73, 93]. Jeong [94] et al. considered the incremental offloading problem and proposed to partition a DNN into various subtasks. The prediction of energy consumption for each DNN's layer is proposed in [95], and then a partition scheme is adopted. These partition-based methods have an irreparable limitation when applied to real computation-constrained edge devices. The drawback is that the intermediate data sizes of the first several layers are still large [96], even larger than the raw data [80]. The device does not have enough capacity or energy budget to compute the whole layers before the small output point. There are also some works considered from data compression [13,96,97] with extra-trained encoder and decoder, and the encoder is conducted by the edge device.

In our scheme, the decoder and encoder are all moved into the server for time-saving. [65] and [66] consider accelerating the inference with the cooperation of multiple edge devices, which is a different scene from ours.

There are some works that focus on the task of video analysis in an offloading scheme. zhang et al. [98] proposed only uploading the frame/camera that has the best view to capture the scene. [99] adjusts the encoding quality on each frame to reduce the transmission latency based on the Regions of Interest (RoIs) detected in the last offloaded frame. The DDS [67] is present to transmit a low-quality image to the server in video analysis. The server conducts an inference model for tasks like object detection or semantic segmentation and chooses the uncertain area. The device transmits the high-quality content of the chosen area again to achieve high inference accuracy. This video-focused work is inefficient to apply to image recognition since the target and input are different. Similar input data redundant reduction works have also been proposed [100, 101]. However, they all demand computing power from the device.

5.5.2 Image Sparsity and Completion

In contrast to language that contains high semantics and information density, an image usually contains heavy spatial redundancy [16]. There are some works

trying to reduce the redundancy in the inference of images to accelerate the process and improve accuracy. Different from the common operation in ViT that divides an image into 14×14 tokens, the DVT [102] claims that some images are suitable for 4×4 tokens, thus reducing computation overhead. It processes the image by sequentially activating a cascade of Transformers with increasing numbers of tokens. DynamicVit uses a prediction module to prune the unimportant tokens in the training to accelerate the process. Similar image sparsity ideas have also been proposed in [103–105]. These works are orthogonal to ours since they reduce the redundancy during the conducting process rather than at the beginning.

There are some works related to image completion or image inpainting that target filling reasonable content with photorealistic appearance into missing regions [85, 87]. These works have different targets for our reconstruction process; they may reconstruct totally unrelated content when we have a large mask ratio.

5.6 Conclusion

In this work, we propose a two-round offloading scheme to save bandwidth resources for weak edge devices. The MAE model is utilized to recover the sampled images transmitted from the edge. All images transmit a small volume of data at the first round; the confidence and attention results from the inference will determine if the second offloading is needed and what content should be offloaded. The SLO-adaptive module is also made to look into how the confidence score and the amount of data sent are related to determine the transmission volume for each image independently. Finally, the lightweight inference module is proposed to save inference time and improve accuracy. We compare our system with different baselines to verify its effectiveness.

Chapter 6

Conclusion and Future Work

In this chapter, we conclude our research contributions and discuss our further work. Our research explores the data sparsity in both the training and inference processes for graph learning. In federated graph training, graph data is distributed among different owners, and information exchange among nodes is required. We propose an efficient system that eliminates unnecessary transmission and dynamically tunes the priority of flows according to the training stage and completion time. Additionally, we introduce a homomorphic encryption (HE)-based security protocol. In graph inference, we find that repeated loading of nodes into the GPU significantly reduces inference speed. To address this, we propose a more efficient system that uses adaptive sampling neighbors and reuses the loaded data. The LSH-based two-level clustering scheme is also proposed to reorder the batches and reuse more nodes. Lastly, we explore the sparsity of data in images and propose an offloading system that doesn't require computation on the edge device and only transmits part of the image data to the server by utilizing the image sparsity. The masked autoencoder (MAE)-based model is used to recover the masked image. To achieve content-aware transmission, we propose a feedback-driven scheme that utilizes threshold and attention values to tackle the dilemma between random mask patterns and high accuracy requirements. All proposed systems are compared to various baselines to verify their effectiveness.

Our future research directions will focus on the optimization of data scheduling mechanisms tailored for users engaging with expansive models like GPT. Due to the computational intensity associated with large models, latency in inference can be a notable challenge, whereas smaller models demonstrate adeptness in swiftly processing rudimentary input data. A systematic approach to model selection contingent on the complexity of the dataset can substantially curtail superfluous computational expenditures.

References

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

[2] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[3] H. H. Yang, Z. Liu, T. Q. Quek, and H. V. Poor, "Scheduling policies for federated learning in wireless networks," *IEEE Transactions on Communications*, vol. 68, no. 1, pp. 317–333, 2019.

[4] Y. Zhan, P. Li, and S. Guo, "Experience-driven computational resource allocation of federated learning by deep reinforcement learning," *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 234–243, 2020.

[5] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, pp. 1024–1034, 2017.

[6] Z. Lin, C. Li, Y. Miao, Y. Liu, and Y. Xu, "Pagraph: Scaling gnn training on large graphs via computation-aware caching," *Proceedings of the 11th ACM Symposium on Cloud Computing*, pp. 401–415, 2020.

[7] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems,"

Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 974–983, 2018.

[8] Z. Liu, C. Chen, X. Yang, J. Zhou, X. Li, and L. Song, "Heterogeneous graph neural networks for malicious account detection," *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pp. 2077–2085, 2018.

[9] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile networks and Applications*, vol. 18, no. 1, pp. 129–140, 2013.

[10] Y. Zhan, S. Guo, P. Li, and J. Zhang, "A deep reinforcement learning based offloading game in edge computing," *IEEE Transactions on Computers*, vol. 69, no. 6, pp. 883–893, 2020.

[11] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2019.

[12] Y. Xiao, L. Xiao, K. Wan, H. Yang, Y. Zhang, Y. Wu, and Y. Zhang, "Reinforcement learning based energy-efficient collaborative inference for mobile edge computing," *IEEE Transactions on Communications*, 2022.

[13] Y. Matsubara, D. Callegaro, S. Singh, M. Levorato, and F. Restuccia, "Bottlefit: Learning compressed representations in deep neural networks for effective and efficient split computing," *arXiv preprint arXiv:2201.02693*, 2022.

[14] K. Du, Q. Zhang, A. Arapin, H. Wang, Z. Xia, and J. Jiang, "Accmpeg: Optimizing video encoding for video analytics," *arXiv preprint arXiv:2204.12534*, 2022.

[15] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613, 1998.

[16] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16 000–16 009, 2022.

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

[18] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, pp. 1–35, 2018.

[19] A. Z. Broder, "On the resemblance and containment of documents," *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pp. 21–29, 1997.

[20] J. D. M.-W. C. Kenton and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *Proceedings of NAACL-HLT*, pp. 4171–4186, 2019.

[21] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[23] B. Fu, F. Chen, P. Li, and D. Zeng, "Tcb: Accelerating transformer inference services with request concatenation," *Proceedings of the 51st International Conference on Parallel Processing*, 2023. [Online]. Available: https://doi.org/10.1145/3545008.3545052

[24] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[25] T. Liu, P. Li, and Y. Gu, "Glint: Decentralized federated graph learning with traffic throttling and flow scheduling," *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, pp. 1–10, 2021.

[26] F. Chen, P. Li, T. Miyazaki, and C. Wu, "Fedgraph: Federated graph learning with intelligent sampling," *IEEE Transactions on Parallel & Distributed Systems*, vol. 33, no. 08, pp. 1775–1786, aug 2022.

[27] Z. Li, H. Zhou, T. Zhou, H. Yu, Z. Xu, and G. Sun, "Esync: Accelerating intra-domain federated learning in heterogeneous data centers," *IEEE Transactions on Services Computing*, pp. 1–1, 2020.

[28] S.-J. Yang and Y.-R. Chen, "Design adaptive task allocation scheduler to improve mapreduce performance in heterogeneous clouds," *Journal of Network and Computer Applications*, vol. 57, pp. 61–70, 2015.

[29] J.-w. Lee, G. Jang, H. Jung, J.-G. Lee, and U. Lee, "Maximizing mapreduce job speed and reliability in the mobile cloud by optimizing task allocation," *Pervasive and Mobile Computing*, vol. 60, p. 101082, 2019.

[30] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 435–446, 2013.

[31] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," *Proceedings of the 2014 ACM conference on SIGCOMM*, pp. 443–454, 2014.

[32] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "Coda: Toward automatically identifying and scheduling coflows in the dark," *Proceedings of the 2016 ACM SIGCOMM Conference*, pp. 160–173, 2016.

[33] H. Tan, S. H.-C. Jiang, Y. Li, X.-Y. Li, C. Zhang, Z. Han, and F. C. M. Lau, "Joint online coflow routing and scheduling in data center networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 5, pp. 1771–1786, 2019.

[34] F. Chen, P. Li, D. Zeng, and S. Guo, "Edge-assisted short video sharing with guaranteed quality-of-experience," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2021.

[35] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu, "Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning," *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, pp. 493–506, 2020.

[36] T. Nishio, R. Shinkuma, and N. B. Mandayam, "Estimation of individual device contributions for incentivizing federated learning," *arXiv preprint arXiv:2009.09371*, 2020.

[37] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma *et al.*, "Deep graph library: Towards efficient and scalable deep learning on graphs," *arXiv preprint arXiv:1909.01315*, 2019.

[38] https://snap.stanford.edu/data/p2p-Gnutella08.html.

[39] C. Hu, J. Jiang, and Z. Wang, "Decentralized federated learning: A segmented gossip approach," *arXiv preprint arXiv:1908.07782*, 2019.

[40] 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.

[41] M. Mohri, G. Sivek, and A. T. Suresh, "Agnostic federated learning," *arXiv preprint arXiv:1902.00146*, 2019.

[42] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-iid data with reinforcement learning," *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 1698–1707, 2020.

[43] K. Lei, M. Qin, B. Bai, G. Zhang, and M. Yang, "Gcn-gan: A non-linear temporal link prediction model for weighted dynamic networks," *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 388–396, 2019.

[44] A. Li, Z. Qin, R. Liu, Y. Yang, and D. Li, "Spam review detection with graph convolutional networks," *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 2703–2711, 2019.

[45] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," *arXiv preprint arXiv:1710.10568*, 2017.

[46] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266, 2019.

[47] J. Chen, T. Ma, and C. Xiao, "Fastgcn: fast learning with graph convolutional networks via importance sampling," *arXiv preprint arXiv:1801.10247*, 2018.

[48] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[49] W. Jiang and J. Luo, "Graph neural network for traffic forecasting: A survey," *Expert Systems with Applications*, p. 117921, 2022.

[50] Z. Liu, Y. Wang, X. Liang, Y. Ma, Y. Feng, G. Cheng, and Z. Liu, "A graph neural networks-based deep q-learning approach for job shop scheduling problems in traffic management," *Information Sciences*, 2022.

[51] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," *International Conference on Machine Learning*, pp. 942–950, 2018.

[52] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method," *International Conference on Learning Representations*, 2019.

[53] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *Advances in neural information processing systems*, vol. 33, pp. 22 118–22 133, 2020.

[54] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[55] E. Zhu and V. Markovtsev, "Datasketch: big data looks small," 2018, https://www.zenodo.org/record/290602.

[56] C. Chen, K. Li, X. Zou, and Y. Li, "Dygnn: Algorithm and architecture support of dynamic pruning for graph neural networks," *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 1201–1206, 2021.

[57] T. Liu, Y. Chen, D. Li, C. Wu, Y. Zhu, J. He, Y. Peng, H. Chen, H. Chen, and C. Guo, "Bgl: Gpu-efficient gnn training by optimizing graph data i/o and preprocessing," *arXiv preprint arXiv:2112.08541*, 2021.

[58] H. Zhou, A. Srivastava, H. Zeng, R. Kannan, and V. Prasanna, "Accelerating large scale real-time gnn inference using channel pruning," *Proceedings of the VLDB Endowment*, vol. 14, no. 9, pp. 1597–1605, 2021.

[59] H. Zeng, M. Zhang, Y. Xia, A. Srivastava, A. Malevich, R. Kannan, V. Prasanna, L. Jin, and R. Chen, "Decoupling the depth and scope of graph neural networks," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[60] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems journal*, vol. 5, no. 2, pp. 78–101, 1966.

[61] M. Brehob, S. Wagner, E. Torng, and R. Enbody, "Optimal replacement is np-hard for nonstandard caches," *IEEE Transactions on computers*, vol. 53, no. 1, pp. 73–76, 2004.

[62] S. Albers, "New results on web caching with request reordering," *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, pp. 84–92, 2004.

[63] L. Hu and Q. Ni, "Iot-driven automated object detection algorithm for urban surveillance systems in smart cities," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 747–754, 2017.

[64] J. Yi and Y. Lee, "Heimdall: mobile gpu coordination platform for augmented reality applications," *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pp. 1–14, 2020.

[65] R. Hadidi, J. Cao, M. Woodward, M. S. Ryoo, and H. Kim, "Real-time image recognition using collaborative iot devices," *Proceedings of the 1st on Reproducible Quality-Efficient Systems Tournament on Co-designing Pareto-efficient Deep Learning*, p. 1, 2018.

[66] R. Hadidi, B. Asgari, J. Cao, Y. Bae, D. E. Shim, H. Kim, S.-K. Lim, M. S. Ryoo, and H. Kim, "Lcp: A low-communication parallelization method for fast neural network inference in image recognition," *arXiv preprint arXiv:2003.06464*, 2020.

[67] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, "Server-driven video streaming for deep learning inference," *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pp. 557–570, 2020.

[68] J. S. Jeong, J. Lee, D. Kim, C. Jeon, C. Jeong, Y. Lee, and B.-G. Chun, "Band: coordinated multi-dnn inference on heterogeneous mobile processors," *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services*, pp. 235–247, 2022.

[69] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[70] R. J. Wang, X. Li, and C. X. Ling, "Pelee: A real-time object detection system on mobile devices," *Advances in neural information processing systems*, vol. 31, 2018.

[71] VMW Research Group, "The gflops of the various machines," https://web.eece.maine.edu/~vweaver/group/green_machines.html, 2022.

[72] C. Shi, L. Chen, C. Shen, L. Song, and J. Xu, "Privacy-aware edge computing based on adaptive dnn partitioning," *2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2019.

[73] Z. Xu, L. Zhao, W. Liang, O. F. Rana, P. Zhou, Q. Xia, W. Xu, and G. Wu, "Energy-aware inference offloading for dnn-driven applications in mobile edge clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 4, pp. 799–814, 2020.

[74] J. Qiu, R. Wang, A. Chakrabarti, R. Guérin, and C. Lu, "Adaptive edge offloading for image classification under rate limit," *IEEE Transactions on*

Computer-Aided Design of Integrated Circuits and Systems, vol. 41, no. 11, pp. 3886–3897, 2022.

[75] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for iot devices with energy harvesting," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1930–1941, 2019.

[76] Y. Zhan, S. Guo, P. Li, and J. Zhang, "A deep reinforcement learning based offloading game in edge computing," *IEEE Transactions on Computers*, vol. 69, no. 6, pp. 883–893, 2020.

[77] B. Fu, F. Chen, P. Li, and D. Zeng, "Tcb: Accelerating transformer inference services with request concatenation," in *Proceedings of the 51st International Conference on Parallel Processing*, 2022, pp. 1–11.

[78] X. Zhou, W. Liang, I. Kevin, K. Wang, and L. T. Yang, "Deep correlation mining based on hierarchical hybrid networks for heterogeneous big data recommendations," *IEEE Transactions on Computational Social Systems*, vol. 8, no. 1, pp. 171–178, 2020.

[79] F. A. Salaht, F. Desprez, and A. Lebre, "An overview of service placement problem in fog and edge computing," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–35, 2020.

[80] W. Shi, Y. Hou, S. Zhou, Z. Niu, Y. Zhang, and L. Geng, "Improving device-edge cooperative inference of deep learning via 2-step pruning," *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 1–6, 2019.

[81] J. Yu, Z. Wang, V. Vasudevan, L. Yeung, M. Seyedhosseini, and Y. Wu, "Coca: Contrastive captioners are image-text foundation models," *arXiv preprint arXiv:2205.01917*, 2022.

[82] M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith *et al.*, "Model

soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time," *International Conference on Machine Learning*, pp. 23 965–23 998, 2022.

[83] X. Chen, X. Wang, S. Changpinyo, A. Piergiovanni, P. Padlewski, D. Salz, S. Goodman, A. Grycner, B. Mustafa, L. Beyer *et al.*, "Pali: A jointly-scaled multilingual language-image model," *arXiv preprint arXiv:2209.06794*, 2022.

[84] K. Han, Y. Wang, J. Guo, Y. Tang, and E. Wu, "Vision gnn: An image is worth graph of nodes," *arXiv preprint arXiv:2206.00272*, 2022.

[85] Q. Dong, C. Cao, and Y. Fu, "Incremental transformer structure enhanced image inpainting with masking positional encoding," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11 358–11 368, 2022.

[86] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jegou, "Training data-efficient image transformers & distillation through attention," *International Conference on Machine Learning*, vol. 139, pp. 10 347–10 357, July 2021.

[87] C. Zheng, T.-J. Cham, J. Cai, and D. Phung, "Bridging global context interactions for high-fidelity image completion," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11 512–11 522, 2022.

[88] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," *International conference on machine learning*, pp. 1321–1330, 2017.

[89] X. Wang, L. Xie, C. Dong, and Y. Shan, "Real-esrgan: Training real-world blind super-resolution with pure synthetic data," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 1905–1914.

[90] T. Liu, P. Li, Y. Gu, and P. Liu, "Efficient transformer inference for extremely weak edge devices using masked autoencoders," in *2023 IEEE International Conference on Communications (ICC)*, 2023.

[91] X. Zhou, W. Liang, I. Kevin, K. Wang, Z. Yan, L. T. Yang, W. Wei, J. Ma, and Q. Jin, "Decentralized p2p federated learning for privacy-preserving and resilient mobile robotic systems," *IEEE Wireless Communications*, vol. 30, no. 2, pp. 82–89, 2023.

[92] X. Zhou, X. Ye, I. Kevin, K. Wang, W. Liang, N. K. C. Nair, S. Shimizu, Z. Yan, and Q. Jin, "Hierarchical federated learning with social context clustering-based participant selection for internet of medical things applications," *IEEE Transactions on Computational Social Systems*, 2023.

[93] X. Chen, J. Zhang, B. Lin, Z. Chen, K. Wolter, and G. Min, "Energy-efficient offloading for dnn-based smart iot systems in cloud-edge environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 3, pp. 683–697, 2021.

[94] H.-J. Jeong, H.-J. Lee, C. H. Shin, and S.-M. Moon, "Ionn: Incremental offloading of neural network computations from mobile devices to edge servers," *Proceedings of the ACM Symposium on Cloud Computing*, pp. 401–411, 2018.

[95] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[96] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and T. Abdelzaher, "Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency," *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, pp. 476–488, 2020.

[97] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool, "Conditional probability models for deep image compression," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4394–4402, 2018.

[98] T. Zhang, A. Chowdhery, P. Bahl, K. Jamieson, and S. Banerjee, "The design and implementation of a wireless video surveillance system," *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pp. 426–438, 2015.

[99] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," *The 25th annual international conference on mobile computing and networking*, pp. 1–16, 2019.

[100] K. Huang and W. Gao, "Real-time neural network inference on extremely weak devices: agile offloading with explainable ai," *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, pp. 200–213, 2022.

[101] S. Jiang and L. Zhang, "Quality-aided annotation service selection in mlaas market," *2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*, pp. 1–11, 2022.

[102] Y. Wang, R. Huang, S. Song, Z. Huang, and G. Huang, "Not all images are worth 16x16 words: Dynamic transformers for efficient image recognition," *Advances in Neural Information Processing Systems*, vol. 34, pp. 11 960–11 973, 2021.

[103] L. Wang, X. Dong, Y. Wang, X. Ying, Z. Lin, W. An, and Y. Guo, "Exploring sparsity in image super-resolution for efficient inference," *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4917–4926, 2021.

[104] L. Yang, Y. Han, X. Chen, S. Song, J. Dai, and G. Huang, "Resolution adaptive networks for efficient inference," *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2369–2378, 2020.

[105] M. Najibi, B. Singh, and L. S. Davis, "Autofocus: Efficient multi-scale inference," *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9745–9755, 2019.