CREATING LARGE REAL AND SYNTHETIC GRAPH DATASETS FOR GNN APPLICATIONS

BY

ARPANDEEP KHATUA

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Bachelor of Science in Electrical and Computer Engineering
in the Grainger College of Engineering of the
University of Illinois Urbana-Champaign, 2022

Urbana, Illinois

Adviser:

Professor Wen-mei Hwu

ABSTRACT

Graphs are powerful data structures that are used in algorithms for solving complex problems like recommender systems, fraud detection, and influence prediction. However, graphs alone have limited explicit information, and mining this information to get additional insights is often challenging. Graph Neural Nets (GNNs), a class of deep neural networks, have become a popular method to process graph data for such insights and to solve a wide variety of emerging downstream tasks like classification, clustering, molecular and protein structure prediction, recommendation, and predicting user behavior in e-commerce and social networks and extracting meaningful insights from unstructured data. GNNs have proliferated in these emerging applications due to their unique capabilities, like incorporating node, edge, and graph-level information into the output prediction. However, the lack of datasets of realistic sizes has plagued the development of GNNs due to the proprietary nature of industry data, limited size, and the non-availability of high-fidelity synthetic datasets.

In this work, we introduce the **Illinois Graph Benchmark (IGB)**, a collection of enormous graph datasets for node classification tasks. IGB incorporates the most extensive real-world homogeneous graph with 260 million nodes and more than three billion edges, including 220 million labeled nodes for node classification tasks. Compared to the largest graph dataset publicly available, IGB provides over 162× more labeled data for deep learning practitioners and developers to create and evaluate the model with higher accuracy. IGB captures relational information in the Microsoft Academic Graph for the edges and nodes and the Semantic Scholar database for the node labels. IGB also comprises synthetic and real graph datasets where the synthetic dataset has randomly initialized node embeddings while the real graph dataset has variable dimension node embeddings generated using Sentence-BERT models. IGB provides a comprehensive study on the impact

of embedding generation and large labeled nodes on various GNN models. IGB is compatible with popular GNN frameworks like DGL and PyTorch Geometric and comes with predefined popular models like graph Convolutional Neural Networks (GCN), GraphSAGE, and Graph Attention Network (GAT) for easy model development. IGB is open-sourced, including its methodology, and is available at `https://github.com/IllinoisGraphBenchmark`.

To my parents, for their love and support.

ACKNOWLEDGMENTS

I would like to thank **Prof. Wen-mei Hwu** and **Dr. Vikram Sharma Mailthody** for their continued support and for making this possible. I would also like to thank **Dr. Xiang Song** from Deep Graph Library and AWS for providing advice and feedback along with resources to store and test this dataset. Finally, I would like to extend my gratitude to **Dr. Tengfei Ma** (IBM) and **Dr. Piotr Bigaj** (NVIDIA) for providing valuable insights and industry requirements.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Deep learning has been successful in a variety of domains including images, video, and natural language processing. The expressive power of deep learning enables extracting meaningful insights from vast unstructured data. While deep learning can easily capture hidden patterns in the data, more and more applications in the enterprise domain are being represented in the form of graphs. This is because graphs are powerful data structures for representing valuable relations information and can be used effectively by algorithms that solve complex problems like recommender systems, fraud detection, and influence prediction.

However, graphs alone have limited explicit information as it models a set of objects and their relationships. Additional processing is needed to extract insights needed for supporting downstream applications. To address this, more recently, considerable research attention on extending deep learning approaches to graph data has become popular. In this segment, graph neural networks (GNNs), a class of deep neural networks, have become widely used techniques to work on graph data and solve many emerging complex tasks.

GNNs have been shown to perform well on downstream tasks such as recommending and predicting user behavior in e-commerce and social networks [1, 2, 3], predicting molecular and protein structure [4, 5], and more recently helping in fine-tuning large language models [6]. However, existing dataset properties used for GNN model development have plagued their development due to the tiny nature of the publicly available graph datasets and also due to the proprietary nature of industry data.

To address the limited dataset size limitation, recent work such as OGBN and MAG [7, 8]. have proposed open-source, large graph benchmark suites providing up to 121 million nodes and 1.6 billion edge graphs. However, compared to the industrial use cases, OGBN datasets are still small. Furthermore, most existing datasets including OGBN datasets provide a tiny set of labeled data. As GNN downstream tasks are often trained as supervised learning tasks, having a large labeled data matters. However, both OGBN and MAG use Arxiv [7, 8] class labels which provide only 1.4 million labeled nodes, about 1% of the overall dataset is labeled! With such small labeled data usage during training, it becomes difficult to judge when the model fails to converge if the model is inherently incapable of learning or we are not giving sufficient data for the model to learn [9, 10, 11, 12, 13].

Furthermore, prior datasets lack flexibility and cannot be used to perform an in-depth study to understand the impact of node embedding generation on the GNN model accuracy. As embedding vectors generated from NLP models are directly used to train GNN models, understanding the impact of node embedding generation becomes essential at a large scale.

Besides these limitations, a consequence of GNN's popularity has resulted in developing optimized hardware and system solutions tailored to the GNN's need. However, most of these system designs (such as multi-GPU or multi-node scaling) are still executed with tiny datasets. As the challenges executing GNN with tiny-datasets are quite different than large datasets, it is entirely possible to build a system that may not address a real-world problem.

To this end, this thesis proposes **Illinois Graph Benchmark (IGB)**, a collection of enormous graph datasets for node classification tasks. IGB incorporates the most extensive real-world homogeneous graph with 269 million nodes and more than three billion edges, including 220 million labeled nodes for node classification tasks. Compared to the largest graph dataset publicly available, IGB provides over 162× more labeled data for deep learning practitioners and developers to create and evaluate the model with higher accuracy.

This thesis also does extensive ablation study on the impact of node embedding generation on the GNN model accuracy. Through this study, we show Roberta NLP embeddings provides better accuracy on GNN models and larger embedding dimension assists in the GNN model accuracy. However, if the user is memory constrained, PCA dimensionality reduction can be applied to reduce the Roberta embedding vectors from 1024-dimensions to 384-dimensions saving memory footprint by 2.67× with GNN model accuracy loss up to 3.55%.

This dataset is a large citation network and is made for paper field of study prediction. Each paper node is labeled with a field of study for a multi-class supervised classification task. The number of unique labels in a dataset directly impacts the performance of models since more types indicate a finer prediction precision. In this dataset, we provide two sets of labels for tasks of varying difficulty. The first set consists of 19 classes while the second set consists of 2983 classes. We see an average of 14.8% drop in performance from the 19 class task to the 2983 class task in IGB-tiny and 7% in IGB-small.

Finally, this thesis discusses the limitations in the existing systems where we show as the IGB dataset size increases, the effective GPU utilization is lowered, as the GPUs are waiting for the embedding gather operation to finish. This is especially true with the IGB260M (full dataset) which requires more than 1.2TB of memory space and requires memory mapping from the storage. This shows existing systems fail to adequately support efficient training and inference operation on GNN models when the models do not fit in the host memory of a single node.

Overall this thesis makes the following key contributions:

1. We propose a methodology to create a large graph dataset using real-world data and provide a set of 5 homogeneous graph datasets ranging from 100K nodes to 260M nodes. We provide real-world node embeddings for GNN research and synthetic versions for system design exploration and evaluation.

2. We study the impacts of various factors like the amount of labeled data, embedding model type, and dimensionality, and language implications.

IGB is compatible with popular GNN frameworks like DGL and PyTorch Geometric and comes with predefined popular models like graph Convolutional Neural Networks (GCN), GraphSAGE, and Graph Attention Network (GAT) for easy model development. IGB is open-sourced, including its methodology, and is available at:
`https://github.com/IllinoisGraphBenchmark`.

CHAPTER 2

BACKGROUND

This chapter provides an overview of the materials and background required to understand the main contributions of this thesis. Initially, we will define what graphs are and then discuss how they are useful. We will then formally define graph neural networks (GNNs) and describe the widely used GNN models for various tasks. Lastly, we will motivate why we should build extremely large graph datasets and their benefit for the community.

2.1 Graphs and Graph Tasks

A graph is a collection of entities that have relations with each other. Graphs are non-linear structures consisting of nodes and edges. Typically nodes represent a person (or an animal) or event, or row in a table or topic or similar in an abstract representation of knowledge while the edge represents how the two nodes are related and connected.

Apart from a network of relations, the graph can store additional information that can provide useful insights to understand the behavior of interactions across two entities. For example, in an author-paper citation graph, a node represents an author and can have additional information, such as affiliation, that may represent the author's relationship with an organization. As this additional information can easily be added to the graph, the graph becomes a powerful data structure.

Graphs can be directed or undirected depending on the type of their edges. For instance, in the author-paper citation graph, `paperA` cites `paperB` does not necessarily imply that the `paperB` should cite `paperA`. Thus, this graph makes a directed graph where one direction is known. An example of an undirected graph would be a road network where a road between two cities forms an undirected edge between the two city nodes.

Because of their rich information, prior research has used different methods to perform various tasks based on use cases.[14, 15, 16] Broadly, we can classify these different graph tasks into three types: graph-level, node-level, and edge-level. A graph-level task uses entire graph knowledge to perform a specific task. For example, assume we want to classify an organic chemical based on its structure. The graph used to represent the chemical would have atoms as nodes and chemical bonds between atoms as edges. This application can be solved by using a graph-level task operation. A node-level task uses information from a cluster of nodes and performs a specific task that predicts the property of nodes or classifies a group of nodes. An excellent example of this operation is influenced maximization algorithms to determine important nodes.[17] An edge-level task uses edge information to perform a specific task. A good example in this category would be predicting an edge between two entities in the case of social recommender systems.

2.2 Graph Neural Networks (GNN)

Traditional algorithms have been used on graphs for over half a century now. We have the Strongly Connected Components Algorithm and Floyd's Cycle Detection Algorithm, which can be used for graph classification. For node tasks like influence maximization, Page Ranking can be used to find the best nodes from a graph. While these classical algorithms do a great job, they are unable to learn any insights from the data. To this end, researchers have proposed a graph neural network, a family of neural networks that operates on graph-structured data to provide meaningful insights from graph data.

Over just the last three years, the number of Graph Neural Network (GNN) publications constitutes more than 10% of all Machine Learning publications as shown in Figure 2.1 taken from [18]. This shows the popularity and traction the community is currently experiencing.

Let's take the house price prediction application as shown in Figure 2.2 to understand why GNNs has been an area of interest.

Figure 2.2 shows the house price prediction use case with a regular condo and a mansion. We will also assume certain home prices as shown in Figure 2.2. To apply GNN for this task, first, we need to convert text information such as location, price, and type of property into embeddings. These embed-

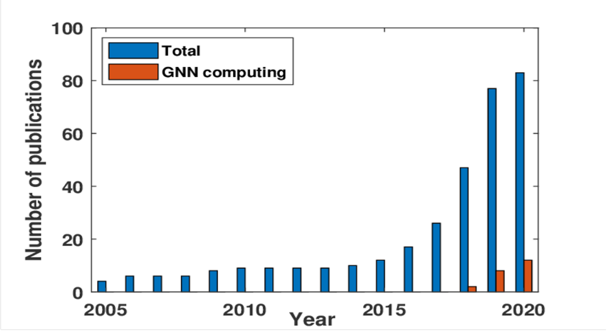


Figure 2.1: GNN-related publications trend as a fraction of the ML publications in the last decade as described by [18]. The unit of the y-axis is thousands.

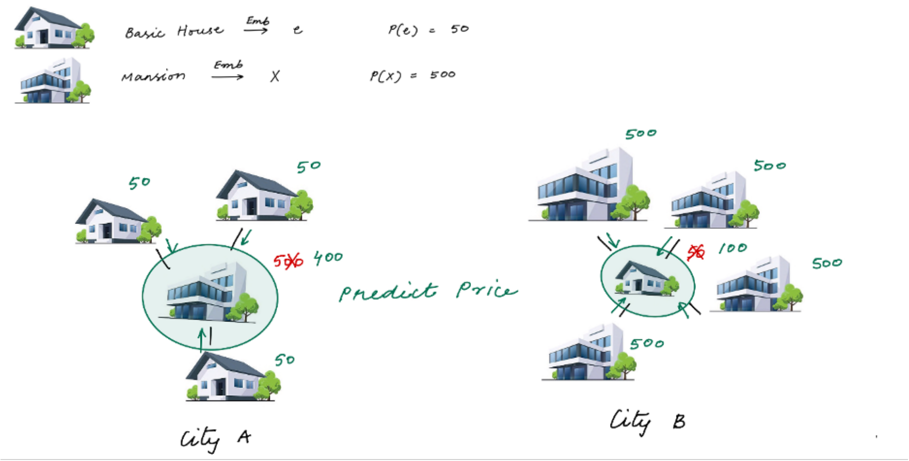


Figure 2.2: Example for applications for Graph Neural networks for predicting house price.

dings become input to the GNN model. For the sake of example, let's assign the ground truth price, which can be x dollars for the condo and 10x for the mansion.

Now let us consider two settings – a small town and a rich suburb near a large metropolitan city. In a small town, City A in Figure 2.2, where the majority of the dwelling are condos, using our current model of embeddings, we predict the price of the only mansion in the town to be 10x, where x is the prices of the condos in the small town. In a rich suburb primarily with mansions, City B in Figure 2.2, we would predict the price of the condo to be x. However, this is not the case. If we were to consider the surroundings, using neighboring houses, we would get the process of the mansion in the small town to be lower than 10x, whereas the price of the condo in the big city would be higher than x. Traditional algorithms can be used for graph tasks like cycle detection and influence maximization; however, they are unable to provide additional insight for predicting prices. As GNN uses multi-layer models where the output of each layer becomes the input to the next layer, the GNN essentially updates each node based on the structural information of the graph, and hence the final graph can then be passed through a dense network to make accurate location-specific predictions. This is the power of GNNs which makes them so popular in the machine-learning community.

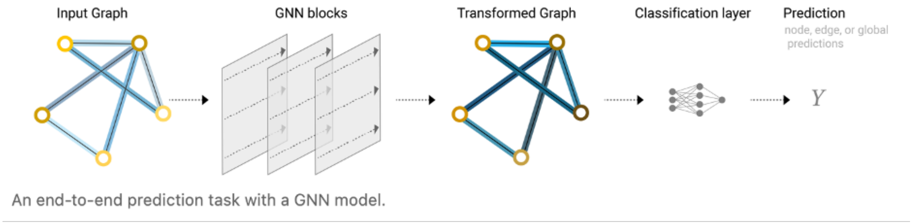

Input Graph
GNN blocks
Transformed Graph
Classification layer
Prediction
node, edge, or global predictions
Y
An end-to-end prediction task with a GNN model.

Figure 2.3: Graph Neural Net pipeline overview

Figure 2.3 shows a classic pipeline of the GNN model. First let's consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N nodes $v_i \in \mathcal{V}$, edges $(v_i, v_j) \in \mathcal{E}$ with an adjacency matrix $\mathcal{A} \in \mathbb{R}^{N \times N}$. The vector representation can be for nodes or edges depending on the downstream task. We can use NLP methods to map nodes into an N-dimensional embedding space such that nodes that are similar would have similar embeddings. This stage, with a list of embeddings and labels along with the adjacency matrix, becomes the input of the model. Next, we feed the adjacency matrix and embeddings into the GNN model, which transforms these embeddings using the structural information provided in the adjacency matrix. Using neighborhood information, the GNN essentially transforms the node embedding of similar nodes closer to each other in the N-dimensional space, thus producing more expressive embeddings for downstream tasks. This output graph's node embeddings can now be passed into a dense, fully connected layer to evaluate tasks like node classification. Next, let's investigate specific graph neural network architectures and algorithms, particularly for node classification tasks. The input graph has node embedding and we can construct an adjacency matrix for the undirected graph with each node labeled into different classes.

2.3 Types of Graphs and GNNs

Graph neural network models require to be modeled differently based on the type of the graph. Graphs can be classified into two major types:

- **Homogeneous Graph**: one type of node and one type of relation expressed as an edge.

- **Heterogeneous Graph**: multiple types of nodes and multiple types of relations.

The most widely used GNN models on homogeneous graphs are Graph Convolutional Neural Network (GCN), GraphSage, and Graph Attention Neural Network (GAT) [19, 20, 21]. Similarly, for heterogeneous graphs, we have Relational-GCN and Relational-GAT and Heterogeneous Graph Transformer (HGT) [22, 23]. In the next few subsections, we will deep dive into each of these models and understand their differences. This is important because, at the end of the section, we will show how one single equation can be used to represent all these models in a generalized form.

2.3.1 Homogeneous Graph Neural Network Models

1. Naive GNN model

```
Algorithm 1: GNN forward propagation
Data: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $x_v, \forall v \in \mathcal{V}$; layers $L$; weight
      matrices $\mathbf{W^k}, \forall k \in \{1, \ldots, K\}$; non-linearity $\sigma$
Result: Vector representations $z_v, \forall v \in \mathcal{V}$
$h_v^0 \leftarrow x_v, \forall v \in \mathcal{V}$;
for $k = 1 \ldots K$ do
    for $v \in \mathcal{V}$ do
        $h_v^k \leftarrow \sigma\Big(\sum_{u \in \mathcal{N}(v)} \mathbf{W}^k h_u^{k-1}\Big)$;
    end
end
$z_v \leftarrow h_v^K, \forall v \in \mathcal{V}$;
```

A naive GNN algorithm can be implemented such that node embeddings of each layer get updated using the neighbor's node embedding. The update function of such a naive model can be represented as

$$H^{(l+1)} = \sigma(\mathcal{A}H^{(l)}W^{(l)}) \tag{2.1}$$

where $H^{(l)} \in \mathbb{R}^{N\times \dim(emb)}$ is the matrix of node embedding of layer l, $W^{(l)}$ is layer-specific trainable weight matrix and $\sigma(\cdot)$ denotes the activation function. $\mathcal{A}$ is the adjacency matrix for the graph.

However, this algorithm fails to provide good accuracy as it misses to have a normalizing term. Since we keep summing the embeddings of the neighbors to update a node embedding, the embeddings would explode after a few layers and the model would fail to learn. To address this limitation, prior work proposed a graph convolution network (GCN) [19].

2. Graph Convolutional Neural Network (GCN)

```
Algorithm 2: GCN forward propagation
  Data: Graph $\mathcal{G}(\mathcal{V},\mathcal{E})$; input features $x_v, \forall v \in \mathcal{V}$; layers $L$; weight
        matrices $\mathbf{W^k}, \forall k \in \{1,\dots,K\}$; adjacency matrix $\tilde{a}_{uv} \in \mathcal{A}+I$;
        degree diagonal matrix $\tilde{d}_{uu} = \sum_v \tilde{a}_{uv}$; non-linearity $\sigma$
  Result: Vector representations $z_v, \forall v \in \mathcal{V}$
  $h_v^0 \leftarrow x_v, \forall v \in \mathcal{V}$;
  for $k = 1 \dots K$ do
      for $v \in \mathcal{V}$ do
          n $\leftarrow \tilde{a}_{uv}/\sqrt{\tilde{d}_{uu}\tilde{d}_{vv}}$;
          $h_v^k \leftarrow \sigma\Big(\sum_{u\in\mathcal{N}(v)} \text{n}\mathbf{W}^k h_u^{k-1}\Big)$;
      end
  end
  $z_v \leftarrow h_v^K, \forall v \in \mathcal{V}$;
```

Graph Convolutional Neural Network [19] adds a normalizing term in each layer based on the degrees of the node. This model approximates spectral convolutions of graphs to get the update function

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}) \tag{2.2}$$

where $H^{(l)} \in \mathbb{R}^{N\times \dim(emb)}$ is the matrix of node embedding of layer l, where $\tilde{A}$ is the modified adjacency matrix with self-loops on every node $\mathcal{A}+I$. Here

$\mathcal{A}$ is the adjacency matrix and I is the identity matrix of the graph $\mathcal{G}$ and $\tilde{D}_{uu} = \sum_v \tilde{A}_{uv}$ is the degree matrix. $W^{(l)}$ is layer-specific trainable weight matrix and $\sigma(\cdot)$ denotes the activation function. Using the degree of the self-looped nodes, this update function normalizes the sum of the neighboring node embeddings and the self-loop keeps the current nodes embedding into account. However, updating a node embedding by considering all the neighbors is a very expensive operation. In large dense graphs with nodes having very high degrees, this model becomes infeasible.

3. GraphSAGE

GraphSAGE [20] addresses the problems associated with the GCN model for highly dense graphs by sampling neighboring nodes and aggregating them instead of taking all the nodes. Given a node we can use different kinds of sampling based on random walks, breadth-first search, depth-first search, mean, pooling, LSTM, and gcn [20] to either sample some of the directly neighboring nodes or nodes that are n-hops away from the current node. After we sample neighboring nodes, we can use an aggregate function like mean, max-pooling, or an LSTM layer. The update function can be represented as

$$H^{(l+1)} = \sigma(Aggregate(Sampling(H^{(l)}))W^{(l)}) \tag{2.3}$$

where various different types of aggregators and samplers can be used depending on the use case.

However, these sampling methods are näive and give all the sampled neighbors of a node equal importance. For example in a citation graph where a paper node has other paper nodes which it cites as neighbors, all the cited papers do not have equal importance to the query paper. If we randomly select some of the neighbors or take all the neighbors and give them equal importance we are not taking into consideration cited papers which are more important than other cited papers.

4. Graph Attention Network (GAT)

The graph attention network [21] solves the issue of naive neighborhood sampling where equal importance is given to every sampled neighbor by

Algorithm 3: GraphSAGE forward propagation

```
Data: Graph $\mathcal{G}(\mathcal{V},\mathcal{E})$; input features $x_v, \forall v \in \mathcal{V}$; layers $L$; weight
      matrices $\mathbf{W^k}, \forall k \in \{1,\ldots,K\}$; non-linearity $\sigma$
Result: Vector representations $z_v, \forall v \in \mathcal{V}$
$h_v^0 \leftarrow x_v, \forall v \in \mathcal{V}$;
for $k = 1 \ldots K$ do
    for $v \in \mathcal{V}$ do
        $h^k_{\mathcal{N}(v)} \leftarrow AGGREGATE_k(h_u^{k-1}, \forall u \in \mathcal{N}(v))$;
        $h_v^k \leftarrow \sigma\Big(\mathbf{W^k} \cdot CONCAT(h_v^{k-1}, h^k_{\mathcal{N}(v)})\Big)$;
    end
    $h_v^k \leftarrow h_v^k / \|h_v^k\|_2, \forall v \in \mathcal{V}$;
end
$z_v \leftarrow h_v^K, \forall v \in \mathcal{V}$;
```

using a self-attention layer in order to weigh the neighbors of a node by importance. The model uses a shared attention mechanism a. In the initial state, the model gives equal importance to all the neighbors of the node. The weights are refined as part of the backward propagation steps of the training process. For every node, the model computes the weighted attention of all the neighbors using

$$e_{uv} = a(\mathbf{W}h_v, \mathbf{W}h_u) \tag{2.4}$$

where e_{uv} gives the "importance" of node v to node u. This attention is then normalized to get the normalized attention coefficient

$$\alpha_{vu} = softmax_u(e_{vu}). \tag{2.5}$$

The final update function for each layer of the GAT with multi-headed attention is

$$H^{(l+1)} = \sigma(\mathbf{A}H^{(l)}W^{(l)}) \tag{2.6}$$

where $\mathbf{A}$ is a matrix of all the normalized attention coefficients.

2.3.2 Heterogeneous Graph Neural Network Models

The GNN models reviewed in Section 2.3.1 are designed for homogeneous graphs, in which all nodes and edges belong to the same types, making them infeasible to represent heterogeneous structures.[23] So while homogeneous

Algorithm 4: GAT forward propagation

```
Data: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $x_v, \forall v \in \mathcal{V}$; layers $L$; weight
      matrices $\mathbf{W^k}, \forall k \in \{1, \ldots, K\}$; non-linearity $\sigma$
Result: Vector representations $z_v, \forall v \in \mathcal{V}$
$h_v^0 \leftarrow x_v, \forall v \in \mathcal{V}$;
for $k = 1 \ldots K$ do
    for $v \in \mathcal{V}$ do
        $e_{v\mathcal{N}(v)} = a(\mathbf{W}^k h_v^{k-1}, \mathbf{W}^k h_u^{k-1})$;
        $\alpha_{v\mathcal{N}(v)} = softmax_u(e_{v\mathcal{N}(v)})$;
        $h_v^k \leftarrow \sigma\Big(\sum_{u \in \mathcal{N}(v)} \alpha_{vu} \mathbf{W}^k h_u^{k-1}\Big)$;
    end
end
$z_v \leftarrow h_v^K, \forall v \in \mathcal{V}$;
```

GNN models perform well, it has huge limitations since practically most real-world graph datasets are heterogeneous, and we cannot apply GCN, GraphSAGE, and GAT to them. Here are primarily two main ways to implement GNN models on heterogeneous graphs: (1) simplify into multi-relational homogeneous graphs, (2) use heterogeneous GNN models.

1. Relational - Graph Convolutional Neural Net (R-GCN)

Algorithm 5: RGCN forward propagation

```
Data: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $x_v, \forall v \in \mathcal{V}$; layers $L$; weight
      matrices $\mathbf{W^k}, \forall k \in \{1, \ldots, K\}$; adjacency matrix $\tilde{a}_{uv} \in \mathcal{A} + I$;
      degree diagonal matrix $\tilde{d}_{uu} = \sum_v \tilde{a}_{uv}$; non-linearity $\sigma$
Result: Vector representations $z_v, \forall v \in \mathcal{V}$
$h_v^0 \leftarrow x_v, \forall v \in \mathcal{V}$;
for $k = 1 \ldots K$ do
    for $r = 1 \ldots R$ do
        for $v \in \mathcal{V}$ do
            n $\leftarrow \tilde{a}_{uv} / \sqrt{\tilde{d}_{uu} \tilde{d}_{vv}}$;
            $h_v^k \leftarrow \sigma\Big(\sum_r \sum_{u \in \mathcal{N}(v)} \mathrm{n} \mathbf{W}_r^k h_u^{k-1}\Big)$;
        end
    end
end
$z_v \leftarrow h_v^K, \forall v \in \mathcal{V}$;
```

Heterogeneous graphs with $\mathcal{X}$ types of nodes and $\mathcal{Y}$ types of edges, can be recursively decomposed by transforming a $\mathcal{N}_{x_i} - \mathcal{E}_{y_j} - \mathcal{N}_{x_k} - \mathcal{E}_{y_j} - \mathcal{N}_{x_i}$ edge into a new type of edge relation and remove all the nodes of type x_k. Here $x_i, x_k \in \mathcal{X}$ and $y_j \in \mathcal{Y}$ and $\mathcal{N}_{x_i}$ is a node of type x_i and $\mathcal{E}_{y_j}$ is an edge relation of the type y_j. This is called meta path decomposition and the types of edges in the graph get updated with the new edge which combines $-\mathcal{E}_{y_j} - \mathcal{N}_{x_k} - \mathcal{E}_{y_j}-$ into $-\mathcal{E}_{y_j}-$ and we can remove the $\mathcal{N}_{x_k}$ nodes. If we do this for $\mathcal{X} - 1$ times we are left with only one type of node and $\mathcal{Y}$ types of edges. These are called homogeneous multi-relational graphs.

A classic graph convolutional neural network uses a single weight matrix $\mathbf{W}^k$ per layer of the model and cannot utilize the information from various different types of relations in a multi-relational graph. In order to extract all the information usefully, the Relational GCN [22] uses a different weight matrix for each layer and each relation where $\mathbf{W}^k_r$ represents the weight matrix of the k^{th} layer and r^{th} relation type. The update function is represented as

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}\mathcal{R}W^{(l)}) \tag{2.7}$$

where $\mathcal{R}$ is the matrix representing the various relations in order to select the relation represented weight matrix.

However, this meta-path decomposition is done by GNN researchers who may not have domain expertise on the data in the datasets. Additionally, since we remove all but one node, we lose a lot of the information present in the original dataset thus leading to poorer learning. In order to overcome these hurdles we need to use heterogeneous GNNs like Heterogeneous Graph Transformer (HGT) [23] which uses attention similar to GAT and meta-relations.

2. Heterogeneous Graph Transformer (HGT)

Relational Graph Convolutional Neural Network works on the principle of implementing GCN for every type of relationship present in the heterogeneous graph. Additionally, the meta-path decomposition causes a loss of information since we lose out on the embeddings of the other types of nodes in the graph. In order to utilize all the information Heterogeneous Graph Transformer [23] uses meta-relations. Given a query node, we take all of its neighbors (either of the same types or different types) and normalize the embedding dimension of the neighboring nodes to that of the query node. Then we compute the attention coefficient of each of the neighboring nodes with respect to the query node. Additionally, we implement a message-passing operation from the neighboring nodes to the query node. The result from the message passing is then multiplied by the respective attention coefficient and summed. This final result is passed through an activation layer which finally gives the new embedding of the query node.

2.4 Generalizing GNN Models

Comprehending all the above groups of models and their variations is a daunting task. To address this, in this thesis, we formalize the GNN models discussed above into a generalizable equation described as:

$$h_v^k \leftarrow \texttt{UPDATE}(h_v^k, \texttt{SAMPLE}(h_u^{k-1})) \tag{2.8}$$

Depending on the number of types of relations, and neighbor sampling we can get the updated equations of the popular graph neural net models as shown in Table 2.1.

Table 2.1: Popular GNN models' update function comparison

`Graph Convolutional Network`	$h_v^k \leftarrow \sigma \overbrace{\sum_{u\in\mathcal{N}(v)}}^{\texttt{SAMPLE}} \underbrace{(1/c_{uv})\mathbf{W}^k}_{\texttt{UPDATE}} h_u^{k-1}$
`GraphSage`	$h_v^k \leftarrow \sigma \underbrace{\mathbf{W}^k AGGREGATE}_{\texttt{UPDATE}} \overbrace{{}_{u\in\mathcal{N}'(v)} h_u^{k-1}}^{\texttt{SAMPLE}}$
`Graph Attention Network`	$h_v^k \leftarrow \sigma \overbrace{\sum_{u\in\mathcal{N}(v)}}^{\texttt{SAMPLE}} \underbrace{\alpha_{uv}\mathbf{W}^k}_{\texttt{UPDATE}} h_u^{k-1}$
`Relational GCN`	$h_v^k \leftarrow \sigma \overbrace{\sum_{r\in\mathcal{R}} \sum_{u\in\mathcal{N}^r(v)}}^{\texttt{SAMPLE}} \underbrace{(1/c_{uv})\mathbf{W}_{\mathbf{r}}{}^k}_{\texttt{UPDATE}} h_u^{k-1}$

Note that these equations ignore the self-node consideration and the bias term which remains the same for all models. But from the understanding perspective, Table 2.1 does provide a simple representation to compare various models.

2.5 GNN Frameworks

Due to the growing popularity of graph neural network models, researchers developed dedicated libraries on top of existing machine learning frameworks like PyTorch [24] and Tensorflow [25]. Among them, two GNN libraries are popular: Deep graph Library (DGL) [26] and PyTorch Geometric (PyG) [27].

Deep Graph Library (DGL) [26] is a high-performance and scalable Python package and can be used with any major deep learning framework like PyTorch, Apache MXNet, or Tensorflow. It provides powerful graph objects which can be loaded into the CPU or GPU. DGL provides many state-of-the-art GNN layers and modules for building new graph model architectures and includes benchmarks for many standard datasets including Open Graph Benchmark (OGB) [8].

PyTorch Geometric (PyG) [27] is a library built on PyTorch which can be used to implement GNN models very easily. It consists of batched data loaders, multi-GPU support, distributed learning, and benchmarks for common datasets.

2.6 Graph Datasets

In § 2.2 we discussed the most popular supervised GNN models. Supervised learning models require data with ground truth labels which can be used for training and setting up test benchmarks. Furthermore, understanding graph structure greatly helps to understand model performance. Thus, it is essential to define the commonly used metrics while studying the graph structure relevant to GNN. Although these metrics are comprehensive, it provides sufficient details to help understand GNN model accuracy.

The most important metric for each dataset is the number of nodes/edges, the number of classes, and node embedding dimensions. These metrics allow us to determine the scale of GNN models and the expressiveness of the input data. Average degree distribution can also be easily calculated from the specified information. Apart from these metrics, homophily (β) for a graph is a very important metric as shown by many researchers.[28]. Homophily is defined as the fraction of neighbors for each node in a graph with the same label as that node itself [29]. This can be represented in an equation as,

$$\beta = \frac{1}{|V|} \sum_{v \in V} \frac{\#\{n_u : \mathcal{L}(n_u) == \mathcal{L}(n_v), u \in \mathcal{N}(v)\}}{\#\{n_u : u \in \mathcal{N}(v)\}} \tag{2.9}$$

where V is the total number of nodes in the graph and $\mathcal{L}(x)$ is the label of node x.

Apart from homophily, the number of connected components directly affects the performance of GNN models on the dataset [28, 30]. As this thesis proposes a new dataset for GNN, we must compare eventually to understand how these metrics stand across different graph datasets.

The Open Graph Benchmark (OGB) and Deep Graph Library (DGL) have a set of graph datasets for various types of GNN tasks like node/graph classification and edge prediction. These sources provide easy access to data loaders for these datasets and also include benchmarks and leaderboards. Some of the popular node classification graph datasets are [31, 32, 33] listed along with their metrics in Table 2.3.

Table 2.2: Popular node classification datasets

Dataset	#Nodes	#Labelled	#Edges	Homophily
`citeseer`	3,327	1,620	9,228	73.91%
`pubmed`	19,717	2,100	88,651	80.24%
`cora`	19,793	19,793	126,842	81.00%
`ogbn-arxiv`	169,343	169,343	1,166,243	65.51%
`ogbn-mag`	1,939,743	736,389	21,111,007	30.40%
`ogbn-papers100M`	111,059,956	1,500,000	1,615,685,872	-
`mag-240m [7]`	121,751,666	1,400,000	1,297,748,926	-

2.6.1 Problems With Existing Datasets

In the previous section, we summarized the most widely used graph neural network datasets in various domains. However, most of these datasets are tiny and are not representatives of real-world graph sizes that are used in

Table 2.3: Popular node classification datasets metrics.

Dataset	#Classes	Emb	Connected Comp
citeseer	6	$3,703$	438
pubmed	3	500	1
cora	70	$8,710$	78
ogbn-arxiv	40	128	1
ogbn-mag	349	128	$3,798$
ogbn-papers100M	172	128	-
mag-240m [7]	153	768	-

Table 2.4: Largest available public dataset vs industry dataset.

Dataset	Date	#Nodes	#Labelled	#Edges	Emb-dim
ogbn-papers100M [33]	2020	111 M	1.4 M	1.6 B	128
mag-240m [7]	2021	260 M	1.4 M	1.3 B	768
PinSAGE [1]	2018	3 B	UNK	18 B	128 – 1K

industry [1] as shown in Table 2.4. PinSage [1] dataset is one of the ”few” publicly disclosed datasets that have more than three billion nodes and eighteen billion edges. These dataset properties were disclosed in 2018 and as researchers, we can only assume that the dataset has grown significantly since then. To address the limited dataset size limitation, more recent work such as OGBN and MAG have proposed open-large graph benchmark suites that provide up to 121 million nodes and 1.6 billion directed edge graphs. However, compared to the industrial use cases, these datasets are still small. This may be because generating large datasets consumes significant computing resources that are not accessible to all researchers.

Apart from limited size, the existing dataset provides a tiny set of labeled data. As graph neural networks are trained using supervised training algorithms, the number of labeled nodes present in the dataset significantly matters for the overall accuracy of the model. Both OGBN and MAG use Arxiv classes as labels and provide a meager 1.4 million labeled nodes. This constitutes less than 1% of the overall dataset labeled. With such a small number of usable nodes, we cannot accurately train and test models and it is difficult to judge whether a model is invalid or we simply aren't giving it enough data for the model to learn [9, 10, 11, 12].

Besides limited labeled nodes, none of the prior work made an in-depth study on the impact of node embedding generation for determining the GNN accuracy. Given the fact that the embedding vectors directly get used during GNN training, the quality of embedding vectors and how they are generated are currently least studied at a large scale.

Apart from these, with increasing graph neural networks popularity, researchers in the system and high-performance community are developing hardware and system solutions assuming existing tiny datasets. As the challenges associated with tiny datasets are significantly different than working on large datasets, it is entirely possible to build a system that may not be able to address a real-world problem.

Without the existence of large datasets that do not fit into the host memory, it is not possible to build scale-out solutions that stress test system parameters such as IO and communications in a distributed training setup. For instance, if the dataset does not fit in the host memory, then one must perform graph partitioning operation in order to use scale-out solutions. Efficient graph partitioning is a non-trivial task and if the GNN model is used in inference, then the entire GNN query execution time can be dominated by the graph partitioning preprocessing step and not in the actual GNN computing. Such a level of analysis requires access to a large dataset where researchers can evaluate the efficiency of both preprocessing stages and the computing stage for optimized system building.

Thus, in order to test and develop optimized systems which can utilize the complete computation resources of a machine, and enable GNN researchers to build novel efficient models, this thesis proposes a new homogeneous graph dataset that will help the community in two main ways:

- Given a dataset schema, create a methodology to generate arbitrarily sized graphs and node embeddings with a prescribed set of a number of nodes and edges which will enable researchers to create their own sub-datasets.

- Provide a dataset with both synthetic and real-world node embeddings for system developers and GNN researchers to develop new efficient performant graph neural net models that can be practical.

CHAPTER 3

ILLINOIS GRAPH BENCHMARK DATASET

We propose Illinois Graph Benchmark (IGB) dataset to address the limitations present in the existing Graph Neural Network datasets. This chapter will cover in-depth how we generate IGB using publicly available data and then describe various studies performed on the generated dataset that will guide how future GNN models can be designed. We hope the IGB dataset becomes the frontier in designing future models and systems that meet industry standards.

3.1 Challenges To Creating Graph Datasets

Creating large GNN datasets may look easy but is a non-trivial task. We must address the following major challenges while creating a large GNN dataset:

1. The dataset should comply with open-source bylaws, and the relation and properties associated with the dataset should be from the real world. This poses a unique challenge as not all data in the public domain are open to modifying or generating derivative products. Even if such data is available, most of the open-source datasets are small.

2. Data preparation from raw text files looks trivial when the dataset is small. However, with large datasets, the dataset preparation consumes significant resources.

3. Most GNN models use supervised learning methods to learn the tasks. As supervised learning fundamentally depends on the amount of labeled data, it is required to have very large ground-truth (GT) labels generated by humans from specific domains. Generating labels for millions of nodes needs to be automated and is infeasible to perform manually.

4. A single dataset might not have all the information we want. If we want to merge multiple datasets, we need to do so carefully in order to preserve the accuracy of information.

3.2 IGB Dataset Generation Goals

As IGB is created as a new large-scale dataset for GNN models, it brings unique opportunities that can assist in further understanding the impact of dataset properties on the accuracy of the GNN models. Moreover, it must also adhere to the following set of goals.

1. IGB dataset generation process must address the challenges described in § 3.1.

2. IGB dataset generation must study the impact of dataset size and how it affects the training efficiency with respect to accuracy.

3. As IGB provides a massively large number of human-annotated labeled data for GNN-supervised training, IGB must study its impact on various model accuracy.

4. Impact of node embedding must be studied in-depth before selecting the right embedding model generation scheme. Embedding generation depends on the type of model used, the size of the embedding vector, and the language that was used to train the model. The existing datasets do not give us an opportunity to explore these questions.

5. Impact of the number of classes to predict in a multi-class classification task.

6. IGB must elucidate the limitations in existing system hardware as it stresses out the entire system stack while performing GNN training on state-of-the-art systems.

3.3 IGB Dataset Generation Methodology

3.3.1 Selecting Real World Database

In order to create a real-world graph dataset, we need to curate real-world data. Although there are several real-world graph data publicly available [34, 35, 36, 8, 37], the graph datasets either is too small or does not have all the required information that meets IGB generation goals. Among many, Microsoft Academic Graph (MAG) and SemanticScholar Corpus are the two publicly available raw data that meet our criteria. We start by looking into the Microsoft Academic Graph (MAG). The MAG dataset has a lot of relations and information about the different types of data points. These data points can be paper and author tables in addition to various other tables providing information about the paper citation, authorship, affiliation, field of study, and many more. This enables building complex, very large knowledge graphs and appending them with property information specific to the node. The SemanticScholar dataset also provides a similar set of functionalities but is slightly smaller than the MAG dataset.

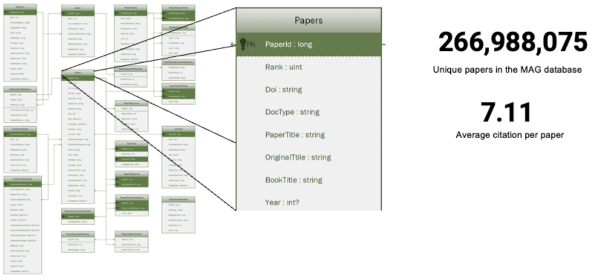


Figure 3.1: MAG papers table and metrics

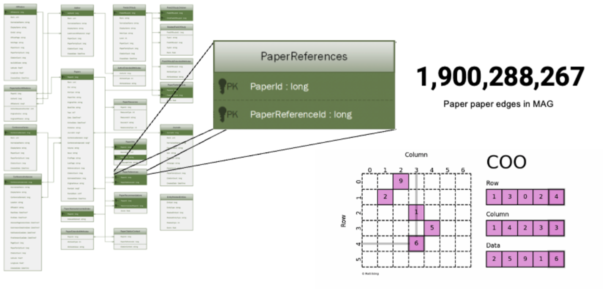


Figure 3.2: MAG papers relations table and metrics

The MAG dataset has over 260M papers and over 1.9B undirected edges representing paper-to-paper citations. The paper table has information about the title, published date, author, citation, and conferences/journals. The paper citations i.e., edges, are stored in a Coordinate (COO) format where the first paper ID cites the paired paper ID.

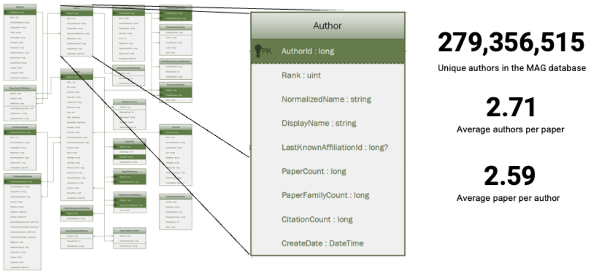


Figure 3.3: MAG authors table and metrics

The author table has a large number of authors who have published papers in the last few decades. The author table includes data about their names, affiliations, and papers they wrote.

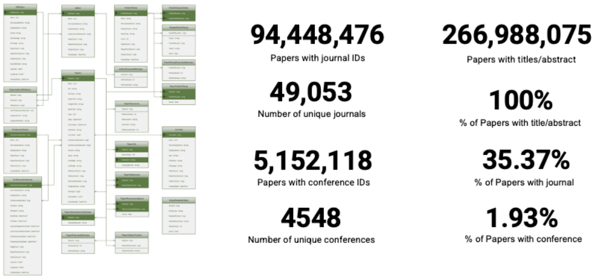

94,448,476
Papers with journal IDs
49,053
Number of unique journals
5,152,118
Papers with conference IDs
4548
Number of unique conferences
266,988,075
Papers with titles/abstract
100%
% of Papers with title/abstract
35.37%
% of Papers with journal
1.93%
% of Papers with conference

Figure 3.4: MAG extra information table and metrics

3.3.2 Graph Dataset Schema

The next step in creating a graph database is to decide the schema – the type of graph, number of nodes and edges, types of nodes and edges, and number of relations. Graphs can be broadly divided into homogeneous or heterogeneous depending on the number of types of nodes present in the graph. Within a homogeneous graph, if we have multiple types of edges between the nodes, it is called a multi-relational graph. IGB design goal requires the generation of both homogeneous and heterogeneous graphs. However, for the thesis, we will limit our discussion to the generation of homogeneous graphs.

3.3.3 IGB Downstream Tasks

The IGB dataset collections are for multi-class classification tasks on three different levels of an increasing number of classes − 19, 292, 2983. The input to the GNN model is the graph dataset with paper nodes initialized with $1024 - dim$ embeddings, and the expected output task is to predict the topic of the paper. This dataset can also be utilized for edge prediction tasks which can be used for citation recommendation.

3.3.4 IGB Ground-Truth Label Generation

IGB supports model training for node-level downstream tasks by providing ground-truth node labels with multiple classes for multi-class node classification tasks. The goal is to predict the paper topics for a paper node given the node embedding and the citation graph. However, finding a collection of a large number of ground truth labels in the real world is a pressing challenge as it is impossible to annotate them manually. For instance, the current largest graph datasets `MAG240M` and `ogbn-papers100M` use `arXiv` labels and have up to $1.4M$ labeled nodes (0.5% of the dataset is labeled). These 1.4M paper nodes are labeled with 172 paper topics which are broadly divided into eight subject areas. Although this looks sufficient, through our discussions with industry partners, we learned that in practice there are tens to thousands of distinct classes required to be predicted for a given GNN model.

IGB uses the MAG dataset [7] and Semantic Scholar [36] dataset in order to annotate over 220M paper nodes into three distinct levels of classes for increasing classification challenges which cover 100% of the nodes for IGB datasets up to large and over 84% coverage for IGB260M (full). We provide 19 for distinct subject areas for the first level of the classification task. In order to increase the difficulty and resemble real-world scenarios we further provide tasks with 292 and 2983 paper topics. Semantic Scholar database includes a reverse mapping for MAG paper ids which makes it simple to connect the datasets in order to maximize the number of labeled nodes we can provide. Thus, the 19-class task is curated by combining classes from MAG and Semantic Scholar and mapping them into a common structure. The 292-class task is extracted from only the MAG dataset. Semantic Scholar also provides multiple labels for each paper. The 2983-class task is created by bucketing all papers which have the same set of paper topics from the set of labels provided by Semantic Scholar. Overall the lower number of class tasks is meant for developing and testing models while the high number of classes is to stress test models and push for more robust GNN models which can be implemented on noisy real-world data.

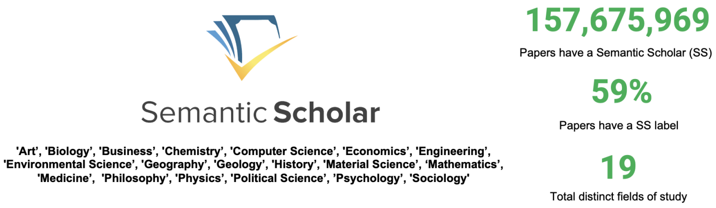

Semantic Scholar
'Art', 'Biology', 'Business', 'Chemistry', 'Computer Science', 'Economics', 'Engineering', 'Environmental Science', 'Geography', 'Geology', 'History', 'Material Science', 'Mathematics', 'Medicine', 'Philosophy', 'Physics', 'Political Science', 'Psychology', 'Sociology'
157,675,969
Papers have a Semantic Scholar (SS)
59%
Papers have a SS label
19
Total distinct fields of study

Figure 3.5: Semantic Scholar paper labels

Thus, IGN enables users to make datasets in a methodological manner – giving users more options to choose the difficulty of problem setup for testing their models.

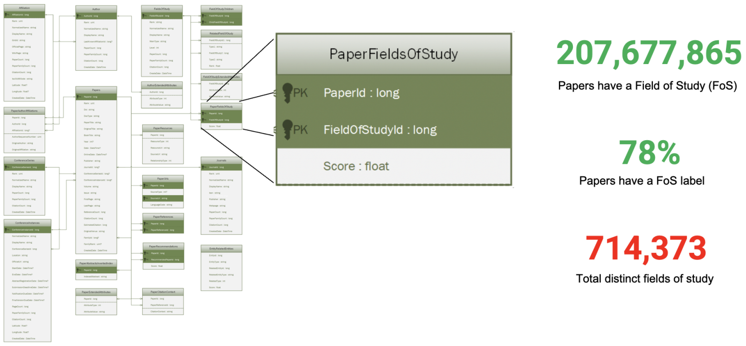
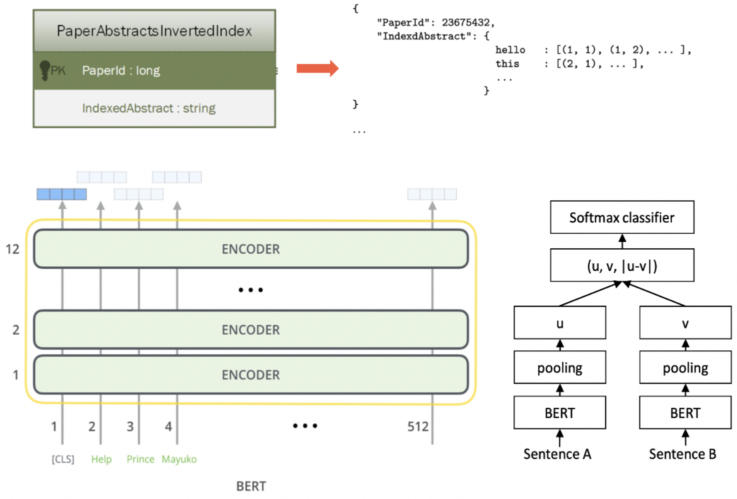


Figure 3.6: MAG Field of Study paper labels

3.3.5 IGB Embedding Generation

GNNs operate based on the structure of the graph; it doesn't have any information about the node itself. In order to give the GNN model information about the node, we need to generate an embedding. GNN models train by "updating" node embeddings using some form of aggregation algorithm from the neighbors. So the GNN itself does not generate any node embeddings and starts from the initialized node embeddings. Since the GNN models don't look inside the node and only work by "learning" from the edges, we need to initialize an NLP-based node embedding that takes in the title and abstract so that the GNN models can perform well. In the past, text embeddings were generated using binary values and word dictionaries. Each text was given an embedding of dimension equal to the size of the dictionary, and if a particular word appeared in the text, that index was assigned 1; otherwise, 0. For Citeseer and Cora, each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. The dictionary consists of 3, 703 unique words in citeseer. [32]. However, with the introduction of word2vec embeddings and, more recently, deep learning-based word and text embeddings, GNNs are being initialized with node embeddings generated using foundational language models. Thus, for our datasets, we need to create node embedding. To do this, we use the paper titles and abstracts and pass them through a sentence-BERT [38] model and generate a $1024 - dim$ node embedding.



```
{
    "PaperId": 23675432,
    "IndexdAbstract": {
                        hello   : [(1, 1), (1, 2), ... ],
                        this    : [(2, 1), ... ],
                        ...
                     }
}

...
```

Figure 3.7: Creating on node embedding using Sentence BERT

In the past few years BERT [39] and RoBERTa [40] have been used to generate embeddings; however, they have a huge computational overhead. Sentence-BERT is based on a Siamese net which can derive semantically meaningful sentence embeddings [38] by modifying a pre-trained BERT model. Additionally, it provides accuracy similar to BERT but reduces runtime to a few seconds. This makes it an obvious choice for us to use to generate node embeddings.

Unlike general language model-based embeddings, it is entirely possible to create domain-specific embeddings. One can use the state-of-the-art emb model SPECTER [41], an embedding model created for scientific text data. SPECTER uses SciBERT [42], a variant of the BERT model. In order to optimize the embeddings for scientific text, SPECTER uses positive and negative samples of papers from the Semantic Scholar citation dataset [36]. SPECTER achieves a 1.5% gain in F1 score compared to Sentence-BERT tested on papers from the MAG dataset.

However, for the IGB dataset, we particularly picked Sentence-BERT embedding trained on web data. This enables validation of the GNN model using generalized text embedding from foundational language models and not from fine-tuned scientific text using the citation networks. The reasoning behind this is that we want to benchmark the GNN model's ability to extract structural information on a dataset initialized with embeddings that have no inherent structural information beforehand. Moreover, in a practical industrial setup, fine-tuning the model for each domain-specific task is a very time-consume process. Thus, foundational model language embeddings play a more important role.

3.4 IGB datasets

The IGB benchmark provides a collection consisting of five datasets of varying sizes. Each dataset is an order of magnitude larger than the previous one and is meant for unique use cases.

Table 3.1: IGB dataset collection metrics

Dataset	#Nodes	% Labeled	#Edges	Degree
`IGB-tiny`	100,000	100	547,416	234/1/5.47
`IGB-small`	1,000,000	100	12,070,502	4,292/1/12.07
`IGB-medium`	10,000,000	100	120,077,694	22,315/1/12.00
`IGB-large`	100,000,000	100	1,223,571,364	73,248/1/12.20
`IGB260M`	269,346,174	84.3	3,995,777,033	277,194/1/14.90

The tiny dataset is provided for dry running and sanity testing of graph neural net models and does not require many resources or storage. It can be easily run on a laptop or edge device and is representative of larger datasets. IGB-small and IGB-medium provide excellent baseline models with minimal training and can be trained with high utilization of a small GPU or powerful

Table 3.2: IGB dataset collection metrics continued

Dataset	Homophily	Emb-dim	#Classes	CC	Storage
IGB-tiny	56.79%	1,024	19/2983	10,943	400 MB
IGB-small	47.75%	1,024	19/2983	677,466	4.6 GB
IGB-medium	59.93%	1,024	19/2983	111,011	40.8 GB
IGB-large	58.27%	1,024	19/2983	-	400 GB
IGB260M	51.79%	1,024	19/2983	-	1.15 TB

CPU. This is ideal for training and testing new GNN models while developing. IGB-large can be trained on a high-end GPU (40GB A100 NVIDIA) with excellent GPU utilization and is ideal for training robust GNN models and can be used for testing by system designers. IGB260M (full) is a massive dataset for GNN developers to build practical models and is also meant to be used for stress testing systems and building efficient distributed training systems for GNN.

Each of the datasets is initialized with $1024-dim$ node embeddings and has three different sets of output classes of increasing difficulty in order to stress test GNN models and optimize model development. The dataset is randomly split with 60% for training, 20% for validations, and 20% for testing. The dataset provides the year of publication meta dataset for every paper node in case users want to specify a splitting rule. Additionally, the dataset has an inherent skew which is characteristic of real-world data. The idea of preserving this skew in all the sizes of the IGB dataset collection is to make them reflect true real-world data.

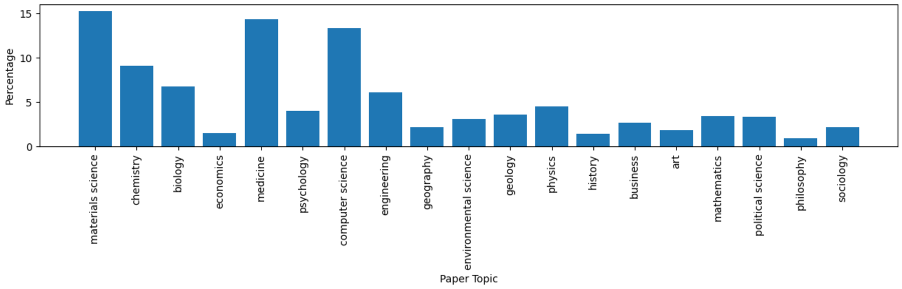


Figure 3.8: Skew in IGB 19-class labels along with the paper topics

CHAPTER 4

EVALUATION

IGB provides five different sizes of datasets for model and system developers to work with. We evaluated IGB datasets on several different types of GNN models using the system described in § 4.1. The high-level summary across GCN, GraphSage, and GAT models with a different number of output classes is provided in Table 4.1.

Apart from model accuracy for the IGB dataset, the thesis goal is also to study the impact of the dataset generation on GNN model accuracy. Our evaluation shows:

- Labeled data provides significant improvement in performance i.e., 150× more labeled data provides increases the node classification accuracy by up to 10%.
- Using an NLP model for initializing node embeddings is crucial as it provides over 40% increase in GNN performance compared to randomly initialized embeddings.
- It is feasible to use dimensionality reduction algorithms like PCA to reduce the dimension of the node embedding by more than half and yet provide similar performance on the IGB dataset.
- GNNs can play an important role in multilingual text embedding finetuning due to their reliance on structure and not the language.

IGB-tiny t-SNE view: t-Distributed Stochastic Neighbor Embedding t-SNE [43] is an unsupervised method that can be used to perform dimensionality reduction by maximizing variance and preserving local similarities. This allows it to cluster similar embeddings closer while pushing dissimilar embeddings away providing a better visualization. We created a t-SNE representation of IGB-tiny to illustrate the graph is a 2D space with similar nodes clustered together.

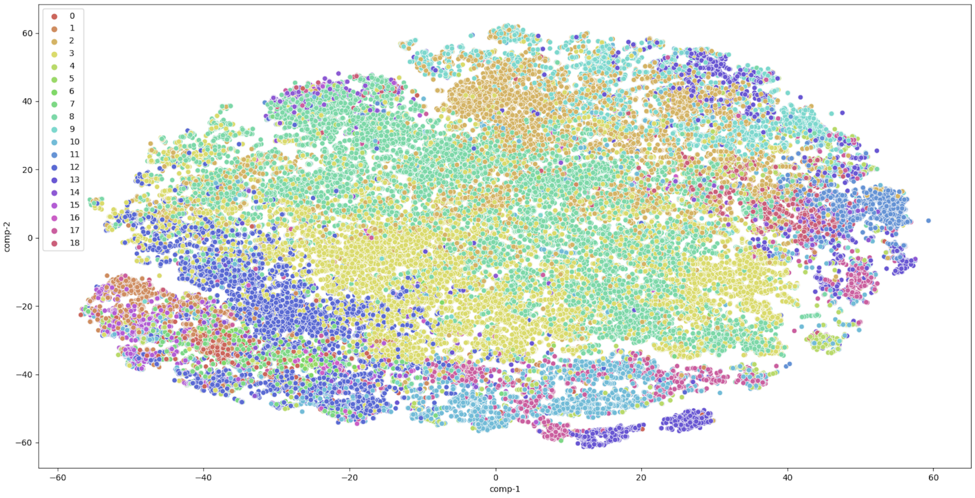


Figure 4.1: t-SNE view of IGB-tiny

Table 4.1: IGB Datasets homogeneous benchmark using same model size all the datasets (* trained for 3 epochs) accuracy metric

Model	GCN		SAGE		GAT	
Dataset	19 acc	2983 acc	19 acc	2983 acc	19 acc	2983 acc
`IGB-tiny`	68.06	53.13	71.87	59.49	68.92	51.74
`IGB-small`	70.46	63.28	75.49	68.70	70.93	63.70
`IGB-medium*`	70.63	62.70	70.35	62.55	70.06	61.81
`IGB-large*`	50.29	–	64.89	–	64.59	–
`IGB260M*`	48.59	–	54.95	–	55.51	–

4.1 Setup

All the models were trained on a system with 255 AMD EPYC 7702 64-Core Processors and an NVIDIA A100-PCIE-40GB GPU. We used a docker container for training and testing and the image used is `nvcr.io/nvidia/pytorch:22.08-py3`. All the datasets were run on batched GCN, SAGE, and GAT models implemented using DGL primitives. Along with the dataset, we provide a DGL and PyG data loader[1]. We used a batch size of 10, 240 in order to maximize GPU SM utilization and shared memory. All the runs use a 0.01 learning rate. IGB-tiny and IGB-small datasets are run on models with two layers with a hidden dimension of 128 and for GAT, there are four heads. All the other models have two layers with a hidden dimension of 16

[1] PyG data loader will be released soon

Table 4.2: System specification

Configuration	Specification
System	Supermicro AS-4124GS-TNR
CPUs	2× AMD EPYC 7702 64-Core Processors
DRAM	1TB Micron DDR4-3200
GPU	NVIDIA A100-80GB PCIe
SSDs	8× Samsung 980pro
Operating System	Ubuntu 20.04 LTS
CUDA Driver and Tools	NVIDIA Driver 470.82, CUDA Toolkit 11.4

and two heads in the GAT model. IGB-tiny and IGB-small are trained for 15 epochs, while the larger models are trained for three epochs.

4.2 Impact Of Labelled Nodes

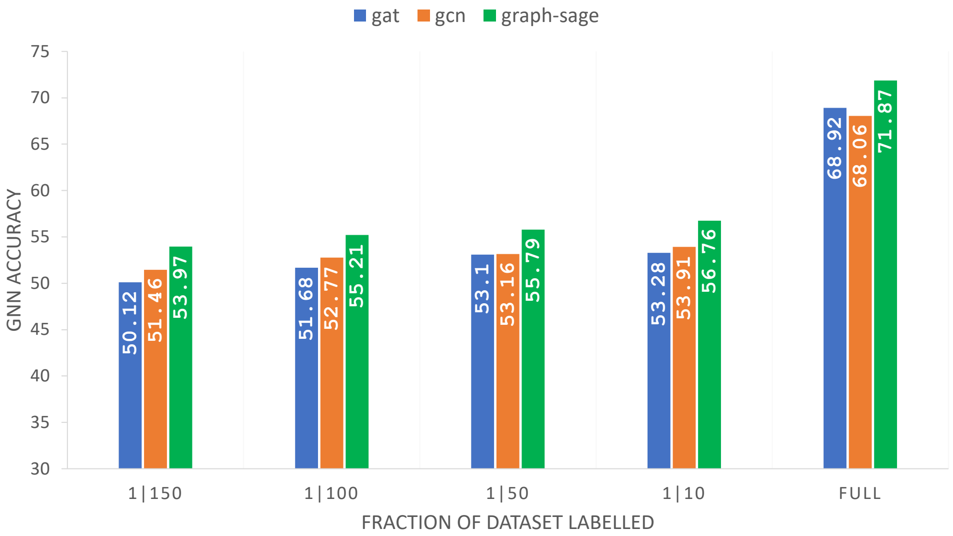


Figure 4.2: Comparison of GNN model performance on IGB-tiny dataset with different fractions of labeled nodes to understand the effect of labeled nodes.

We motivated the importance of labeled data in §2.6.1. In this section, we will evaluate the impact of labeled data on IGB datasets. To perform this experiment, we created several sub-dataset with different fractions of labeled data. GCN, Graph-SAGE, and GAT were then trained and evaluated on these various sub-datasets. Our expectation is the accuracy of the models should improve as we add more labeled data as the tasks are performed using supervised learning techniques. Figure 4.2 shows the improvement in average GNN model accuracy observed when we vary the number of labeled nodes from 1 to 1/150 of the labels present in the dataset. As expected, across all models, we observe significant performance improvement (10%) as we add more labeled data.

Table 4.3: GNN performance compared with the fraction of dataset labeled. The performance is presented as accuracy in percentage points.

Model	Fraction	GCN		SAGE		GAT	
Dataset	Labeled	Val	Test	Val	Test	Val	Test
IGB-tiny	1/10	53.88	53.91	57.03	56.76	52.82	53.28
19 classes	1/50	52.94	53.16	55.63	55.79	52.92	53.10
	1/100	52.19	52.77	55.14	55.21	51.26	51.68
	1/150	51.15	51.46	53.71	53.97	49.68	50.12
IGB-small	1/10	68.68	69.25	72.40	72.89	67.91	68.76
19 classes	1/50	62.55	62.80	65.77	65.88	57.63	57.60
	1/100	56.10	56.60	55.35	57.00	51.80	55.25
	1/150	55.51	57.80	56.11	56.30	53.64	52.85

To enable experiments whose results can be extrapolated to the larger IGB datasets, we created two sub-variant of the dataset for IGB-small and IGB-medium and varied the number of labeled nodes from 1 to 1/150. We evaluated the performance improvement using the three models and then took the average. From Table 4.4 we see that there is a steady decrease in the difference between the accuracy of the full dataset and that of the 1/150 fraction. This is because, as the dataset increases in size, the models can learn more from the structure and depends less on the labeled nodes. As expected, a similar performance improvement was noted. This shows that we can theoretically extrapolate this difference to the large and full model as well, showing a significant bump in test accuracy over datasets with a much smaller fraction of the labeled data. The absolute number may be bigger. Additionally, with more labeled data, we can finally use the whole dataset for training which allows us to work on the GNN training speed and test architectural inefficiencies.

Table 4.4: Node label experiment on various IGB datasets taking an average of GCN, SAGE, and GAT to compare the accuracy difference between the full labeled dataset and 1/150 of the dataset labeled.

Dataset	Full-labelled acc	1/150 labelled acc	Difference
IGB-tiny	69.62%	51.85%	17.77
IGB-small	72.28%	55.65%	16.63
IGB-medium*	70.34%	58.15%	12.18

4.3 Impact Of Node Embedding Generation

We will study the effect of node embeddings on GNN accuracy and the impact of embedding dimension, generation model, and language in this section.

4.3.1 Importance Of Embeddings From Large Language Models

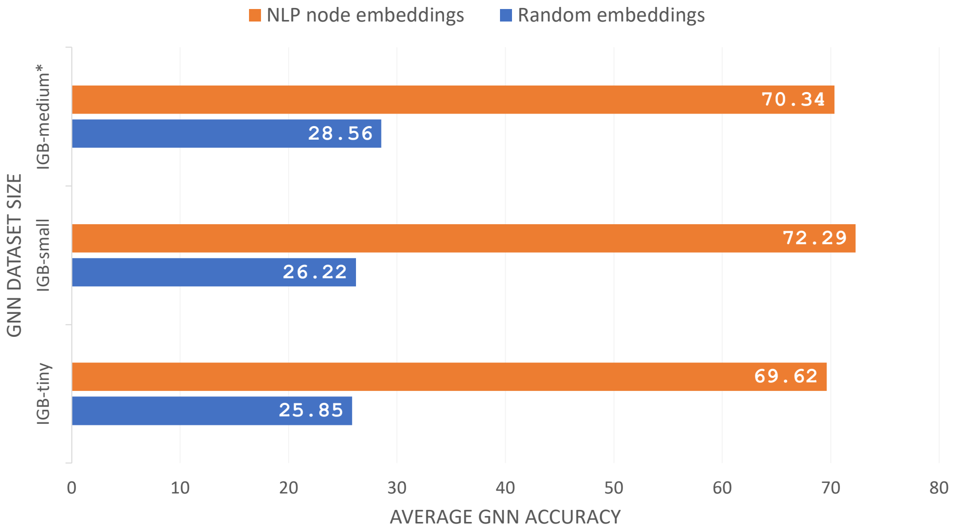


Figure 4.3: Random vs 1024-dim NLP embeddings affecting average GNN performance in IGB-tiny, IGB-small, and IGB-medium.

In order to provide the GNN model with information about the node itself, we need to initialize each node with representative embeddings. Of course, the question arises what happens if we initialize the embeddings with random vectors. **GNN:** In a GNN, the neural model has only 2 main steps: aggregation and update. For a given node, **Aggregation** is getting

Table 4.5: GNN performance compared with random embeddings vs $1024 - dim$ embeddings

Model	GCN		SAGE		GAT	
Dataset	random	real	random	real	random	real
`IGB-tiny`						
19 classes	25.09	68.06	25.37	71.87	27.10	68.92
2983 classes	23.49	53.13	22.29	59.49	23.29	51.74
`IGB-small`						
19 classes	26.02	70.46	25.17	75.49	27.46	70.93
2983 classes	23.88	63.28	23.38	68.70	24.88	63.70

(all/sampled/etc.) neighbors of that node. **Update** is (mean/max/weighted att./etc.) of all the aggregated nodes to "update" the embedding of the given node. In either of these steps, the GNN only has information about the edges (i.e., nodes that are connected to one another) and nothing about the node itself. In most settings, nodes contain important information. In our case, each node is a paper, and the paper has some inherent information, including title and abstract, and it cannot be defined simply by the papers it cites. So for training/testing a GNN dataset, we MUST initialize all the nodes with embeddings having information about the node itself since the *GNN only learns the structural information.* **RecSys Models:** The DLRM-based RecSys takes text or rich information directly as input. So it can be initialized with random embeddings and "learn" the optimal embedding. In our case, this would be that the recsys takes the paper title/abstract to generate the optimal ranked embeddings. Our evaluation shows that the difference in GNN performance with random embeddings and NLP-based embeddings is very stark.

We ran IGB-tiny, small, and medium with roberta (1024-dim) embeddings and random embeddings to compare the difference in test accuracy. Using the large language models for embeddings increases the model performance by more than 3× in the IGB-tiny, IGB-small, and IGB-medium. Extrapolating this, we can predict that this holds for IGB-large and IGB260M too since without the NLP node embeddings the GNN doesn't have any information about the node so it can only learn from the graph structure.

4.3.2 Selecting Right Embedding Model

In order to understand the effect of various large generic language models, we evaluated GNN model accuracy with a few widely used sentence transformer models from the Huggingface library [44]. Table 4.6 summarizes the reported average performance on NLP tasks and model size of generic language models along with their embedding dimensions.

Table 4.6: Sentence Transformer Models from the Huggingface Library

Dataset	Emb dim	Avg. perf	Model size
`all-MiniLM-L6-v2`	384	58.80%	80 MB
`distiluse-base-multilingual`	512	45.59%	480 MB
`all-mpnet-base-v2`	768	63.30%	420 MB
`all-roberta-large-v1`	1,024	61.64%	1,360 MB

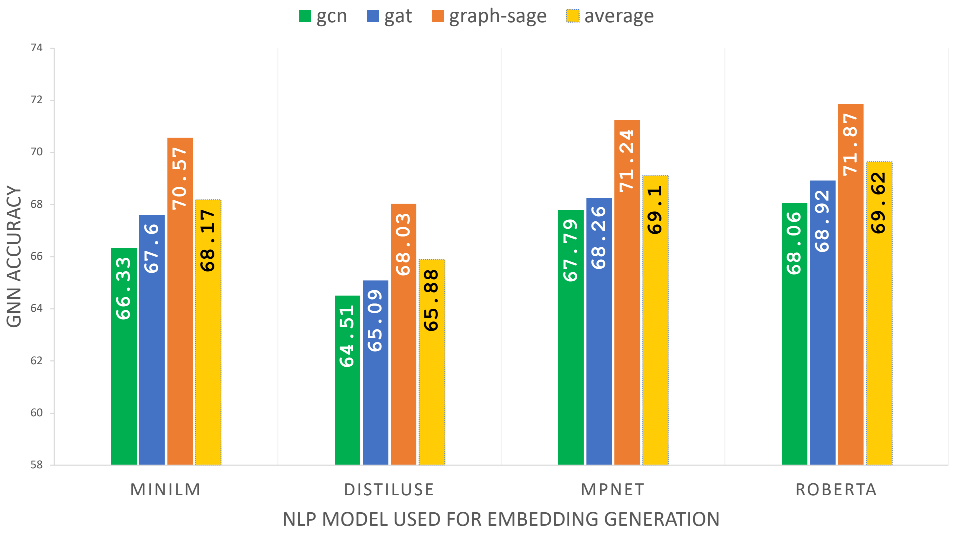


Figure 4.4: GNN accuracy on IGB-tiny using different NLP embedding models for initializing node embeddings.

Figure 4.4 summarizes the impact of different language models on the GNN accuracy by taking the average of GCN, GraphSAGE, and GAT to get model-independent trends. As the embedding dimensions are different, it is impossible to conclude if such a large performance range is due to the model's accuracy or embedding dimensions. However, what can be concluded is that the NLP model in itself has a direct impact on the GNN performance. In the next experiment, we will isolate the impact of the model by normalizing the embedding dimensions using the dimensionality reduction technique. As the embedding dimensions are not the same, it is hard to determine if the model played the role or if the additional information learned by the model by having a larger footprint helped in increasing the accuracy.

4.3.3 Impact Of Embedding Dimension

The previous experiment compared different-sized embeddings from various NLP models. To understand how the models affect the performance we must normalize the embedding dimensions. For models with higher dimensions we used the dimensionality reduction technique, principal decomposition analysis (PCA) [45], and normalized the embedding dimensions. We used the GPU implementation of PCA from the cuML API library [46].

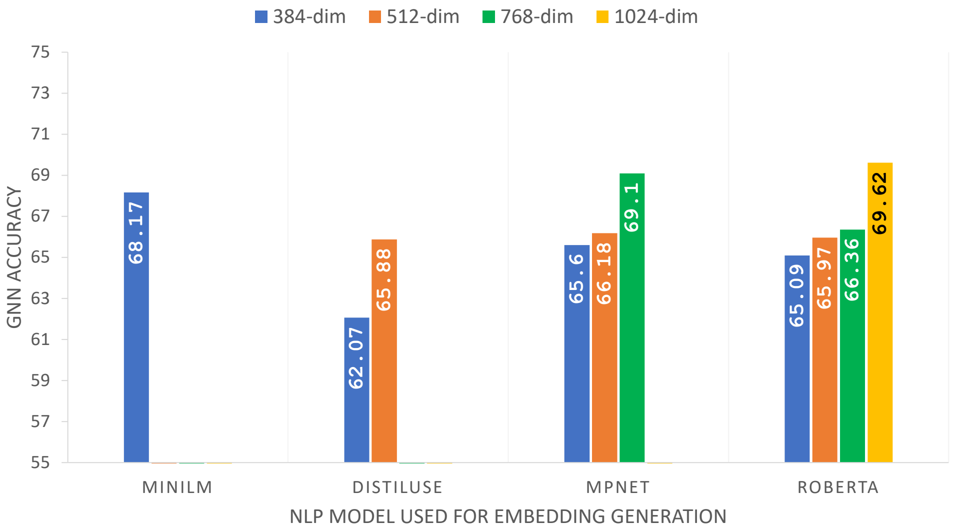


Figure 4.5: Average GNN performance on IGB-tiny based on NLP embedding model after PCA.

Table 4.7: GNN performance on IGB-tiny NLP model after PCA dim reduction. The x rows represent the original NLP model dimension and the columns represent the final emb dimension after PCA reduction.

	GCN				SAGE				GAT			
Dimension	384	512	768	1024	384	512	768	1024	384	512	768	1024
miniLM-384	66.33	-	-	-	70.57	-	-	-	67.60	-	-	-
DISTILUSE-512	58.21	64.51	-	-	64.98	68.03	-	-	63.03	65.09	-	-
MPNET-768	62.30	63.10	67.79	-	68.54	69.27	71.24	-	65.95	66.17	68.26	-
ROBERTA-1024	60.90	62.52	62.94	68.06	68.73	69.48	70.00	71.87	65.64	65.92	66.15	68.92

We ran IGB-tiny with all the possible combinations of dimensions using GCN, GraphSAGE, and GAT GNNs models, and the results are described in Table 4.7. Reducing the dimensions of embeddings using PCA impacts the GNN model accuracy across all GNN models. This is nothing surprising as the PCA is a lossy operation. For Roberta embeddings, when the embedding dimensions are reduced from 1024 to 384, we observe 3.55%, 1.47%, and 0.58% reduction in accuracy for GCN, GraphSAGE, and GAT GNN models. A similar observation is noted in the case of mpnet embeddings and others. This is interesting because from the data we note that the GAT GNN model is more resilient to embedding dimension reductions compared to the GraphSAGE and GCN models.

Moreover, for Roberta embeddings, 2.6× reduction in embedding dimension only resulted in up to 3.55% accuracy loss in the GNN models on the IGB-tiny datasets. This is important as 1024-dim embeddings consume massive memory capacity for storing embeddings vectors during training and inference operations and 2.6× is a significant cost saving especially for IGB260 full. Obviously, the next question is, can we save more memory by further reducing the dimensionality. To understand this, we reduced the embedding dimensions from 1023 to 8 using PCA and evaluate different GNN model accuracies for Roberta embedding vectors as shown in Figure 4.6.

Reducing the embedding dimension by 2× results in 1.03%, 0.72%, and 0.30% reduction in accuracy for GCN, GraphSAGE, and GAT GNN models. Projecting this to $IGB260M$ dataset translates to about a 50% reduction in memory space required for running these models. Further reducing the embedding dimensions deteriorates the performance of GNN models. With embedding dimensions of 128 and 8, the accuracy drops by 4.1%, 3.39%, 2.17% and 13.41%, 12.24%, 9.35% for GCN, GraphSAGE, and GAT GNN models respectively.

Figure 4.5 captures the average of three GNN model accuracy for the IGB-tiny dataset for all NLP embeddings when their embedding dimensions are normalized. The mpnet and Roberta outperform the other miniLM and distiluse models across all GNN models. This is because both these NLP models used for embedding generations inherently are better at language modeling tasks (see Table 4.6) and thus their respective accuracy improvements also get reflected in the GNN model's accuracy. This is good as it aligns with the industrial trend towards building more accurate generic foundational lan-

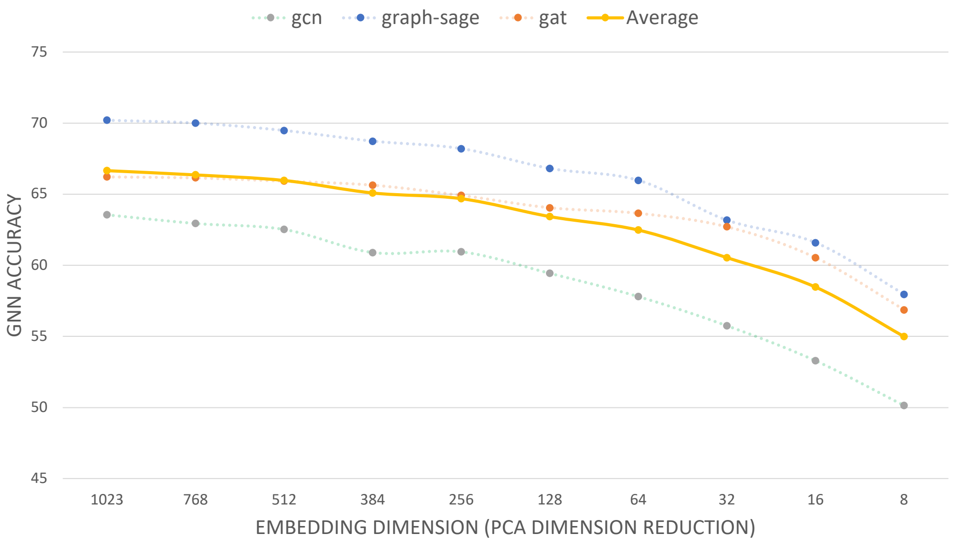


Figure 4.6: GNN performance on Roberta embeddings from 1024-dim to 8-dim across all GNN models

guage models which can further improve the GNN model performance.

Based on this analysis, by default, IGB provides 1024-dim Roberta embeddings vectors which can be converted using the provided toolchains in the IGB dataset. Interested users who want to generate embeddings using other models, can easily generate them by using the tools provided in our open-source codebase [47].

4.4 Language Influence On Embeddings

Often large language models (LLMs) are trained on web corpus data [48, 40]. The majority of the data used for training NLP models are in the English language and thus the embedding generated can have a bias towards a specific language. The MAG dataset used to generate the IGB dataset comprises more than 80 languages of papers. More than 12 languages have over 1M nodes and 50% of nodes have English papers in the MAG dataset as shown in Figure 4.7. Thus, it is of utmost importance to understand if the language in which LLMs are trained matters on the overall accuracy of the GNN models.

Our goal is to determine if the language used in the dataset to train the NLP model makes any difference in the GNN performance. To do this, we must use the NLP model trained exclusively on a specific language. However,

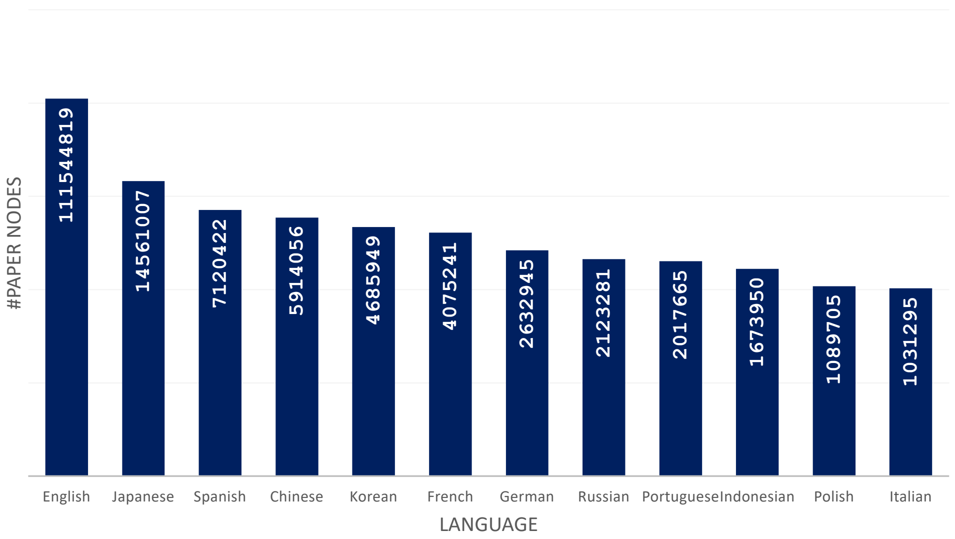


Figure 4.7: Paper language distribution in the MAG database

finding a pre-trained large language model exclusively on a specific language is nearly impossible as of today and thus, we ended up using multilingual language models that include or exclude specific language for this experiment. We picked three languages: Japanese, Spanish and French in our study and created sub-datasets by generating node embeddings with the pre-trained multilingual models from the HuggingFace repository [44]. The models used are

1. `paraphrase-MiniLM-L12-v2` **: (384-eng)** Trained on only English with 384 embedding dimensions.

2. `paraphrase-multilingual-MiniLM-L12-v2` **: (384-all)** Multilingual version of paraphrase-MiniLM-L12-v2, trained on parallel data for 50+ languages and producing 384-embedding dimensions.

3. `paraphrase-mpnet-base-v2` **: (768-eng)** Trained on only English. The embedding dimensions are 768.

4. `paraphrase-multilingual-mpnet-base-v2` **: (768-all)** Multilingual version of paraphrase-MiniLM-L12-v2, trained on parallel data for 50+ languages including English. The embedding dimensions are 768. This includes Japanese.

5. `distiluse-base-multilingual-cased-v1`**: (512-v1)** Multilingual knowledge distilled version of multilingual Universal Sentence Encoder. Supports 13 languages: Arabic, Chinese, Dutch, English, French, German, Italian, Korean, Polish, Portuguese, Russian, Spanish, and Turkish. The embedding dimensions are 512.

6. `distiluse-base-multilingual-cased-v2`**: (512-v2)** Multilingual knowledge distilled version of multilingual Universal Sentence Encoder. This version supports 50+ languages. The embedding dimensions are 512.

We generated two types of embeddings, one with the NLP model trained in the English language (english embedding) and the other with the NLP model trained on multi-lingual that includes Japanese/Spanish/French (multi-lingual embedding). The final created dataset properties are shown in Table 4.8.

Table 4.8: Test datasets created from IGB-full based on three different languages.

Dataset	#Nodes	#Edges	Emb-dim
`IGB-japanese`	$1,975,296$	$3,291,665$	384/512/768
`IGB-spanish`	$1,207,296$	$1,266,183$	384/512/768
`IGB-french`	$841,728$	$1,688,202$	384/512/768

Table 4.9: Performance of average GNN model with different languages showing that language doesn't affect GNN model accuracy significantly

Model*	`IGB-japanese`	`IGB-spanish`	`IGB-french`
$384-eng$	62.89	59.77	60.39
$384-all$	62.83	59.01	59.48
Average	$\mathbf{62.86}_{\pm\mathbf{0.03}}$	$\mathbf{59.39}_{\pm\mathbf{0.38}}$	$\mathbf{59.94}_{\pm\mathbf{0.46}}$
$768-eng$	64.82	61.63	61.15
$768-all$	64.74	61.77	61.25
Average	$\mathbf{64.78}_{\pm\mathbf{0.04}}$	$\mathbf{61.70}_{\pm\mathbf{0.07}}$	$\mathbf{61.20}_{\pm\mathbf{0.05}}$
$512-v1$	61.82	58.86	57.48
$512-v2$	61.61	58.42	57.04
Average	$\mathbf{61.72}_{\pm\mathbf{0.11}}$	$\mathbf{58.64}_{\pm\mathbf{0.22}}$	$\mathbf{57.26}_{\pm\mathbf{0.22}}$

Table 4.9 shows the performance achieved by averaged GNN model with english-embedding and multi-lingual-embedding. From Table 4.9[2], irrespective of model

[2] We also normalized the embedding dimensions using PCA and did not notice a differ-

dimensions and NLP model, there is no significant performance improvement achieved by including specific language of interest. To confirm that this isn't a language-dependent artifact we ran different embedding dimension models on the French and Spanish datasets and noticed a similar trend on GCN, graphSAGE, and GAT. We believe this is because the GNNs learn from the structure of the graph whereas the NLP models rely on the provided text information. Because of this unique ability, GNN models can become language agnostic and help in making NLP models better.

4.5 Limitations In Existing Systems And Framework

IGB is the largest GNN dataset publicly available as of today. The full dataset consumes over 1.2TB in storage space and due to this, many systems fail to adequately support efficient training and inferencing. In this section, we will describe the problems that we encountered while running IGB on state-of-the-art systems and describe potential solutions to the problem.

IGB260M consists of 269+M nodes with 1K-dim embedding vectors in single precision representation requiring more than 1TB of memory capacity in a single node. This memory capacity requirement of IGB260M exceeds the host memory capacity available in modern high-end data center single node-servers. This makes the training within a node infeasible. To address this, one can memory-map the embedding vectors and keep the embedding vectors in a fast-storage medium like SSDs. This enables frameworks like DGL to work on embedding tables that exceed host memory in a single node. One can enable this using Numpy [49] memory-map abstraction.

Although the memory-map abstraction enables training or inferencing to execute on a single node, the entire execution becomes slow when compared to host-memory execution. This is because the memory-map abstraction uses a page-fault mechanism to generate a read request to the storage and move the embedding vectors from storage to CPU memory and then to GPU memory. This incurs significant overhead and overall execution time increases dramatically reducing average GPU utilization as shown in Figure 4.8[3].

If the model fits in the host memory as in the case of IGB-large, then the GPU utilization is significantly improved and reaches up to 80%. Here, our codebase exploits the DGL-UVA [50, 51] techniques where the GPU can directly sample the

ence in the trend.

[3] GPU utilization is captured using `nvidia-smi` tool over a period of 1800 seconds with one second sampling frequency.

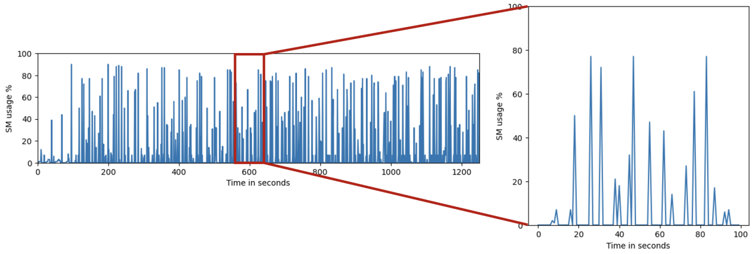


Figure 4.8: GPU streaming multiprocessor utilization for IGB-full.

embedding vectors stored in the CPU memory. This helps to improve the overall training execution time. However, there are still time periods in the training iteration when the GPU utilization is not reaching a peak and is mainly limited by the CPU throughput. This requires further profiling and analysis, which we leave as future work.

If the model fits in the GPU memory, as in the case of IGB-tiny, we can train the entire model in 120 seconds, and the average GPU utilization is high. This is because embedding tables can be directly accessed, which increases the average GPU utilization. Thus, enabling efficient embeddings gather operation is fundamental to the performance of the GNN training algorithm. Hence, from the programmer's perspective, depending on the model size, the GNN model developer must make necessary changes to the source code to ensure it fits in the respective memory hierarchy for efficient execution. However, ideally, what GNN model developers require is a way to efficiently execute GNN models using a single embedding loader class for the IGB dataset that is memory-hierarchy aware and can provide the best performance based on where the embedding vectors are. Such a system design is beyond the scope of this thesis and will be left as future work.

In Fig 4.9, we see the average utilization to be 30%. However, this is due to the sampling rate. `nvidia-smi` samples the GPU utilization every 1s to get the average utilization. To further understand the GPU utilization, we use `nsys profile` to sample the GPU at a higher frequency as shown in Fig 4.10 each of the major operations of the GNN model utilizes 100% of the SM. However, due to the idle periods between operations where the model is waiting for data, the SM utilization is very low. This causes the overall utilization to drop down to 30% as shown in Fig 4.9.

Furthermore, as the graph size increases, the system must evolve and support efficient access to gather embedding vectors from wherever it is stored, be it in memory or storage. However, current system designs are focused exclusively on how to improve performance while gathering data from memory. More recently, the works like BaM [52, 53, 54] can help address this gap directly where the GPU threads can directly read the data where it is stored. How one can integrate BaM into frameworks like DGL while running the IGB dataset is beyond the scope of the thesis and left as future work.

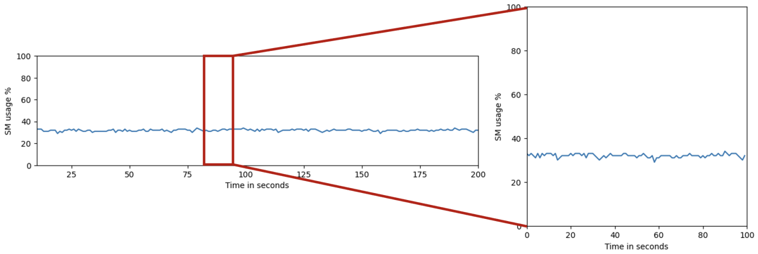


Figure 4.9: GPU streaming multiprocessor utilization for IGB-tiny.

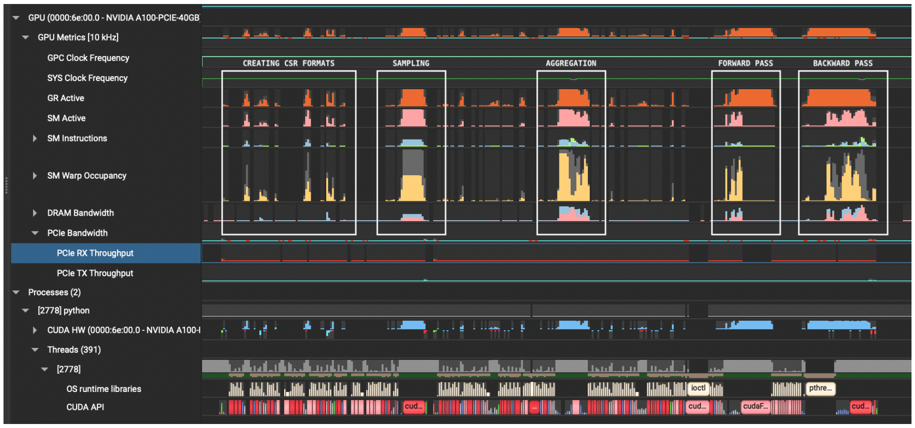


Figure 4.10: IGB-tiny GPU profiling using `nsys` where IGB-tiny is in GPU memory

In this thesis, we focused on single-node GNN model training. We did not evaluate multi-node and multi-GPU training and inferencing due to the lack of system availability and the complexity involved. We hope to address them in the future.

4.6 IGB Dataset Model Perf Summary

In this section, we will summarize the overall results of IGB datasets across all the GNN models and two different complex tasks as shown in Table 4.10. The IGB-tiny and small have each been trained for 20 epochs with the 19 and 2983 class tasks using GCN, GAT, and GraphSAGE GNN models. The accuracy of the models generally sees a dip in performance in the 2983 class task due to the additional complexity of the task where the model has to find a finer difference in

Table 4.10: IGB Datasets homogeneous benchmark using same model size all the datasets (* trained for 3 epochs)

Model	GCN		SAGE		GAT	
Dataset	19 acc	2983 acc	19 acc	2983 acc	19 acc	2983 acc
`IGB-tiny`	68.06	53.13	71.87	59.49	68.92	51.74
`IGB-small`	70.46	63.28	75.49	68.70	70.93	63.70
`IGB-medium*`	70.63	62.70	70.35	62.55	70.06	61.81
`IGB-large*`	50.29	–	64.89	–	64.59	–
`IGB260M*`	48.59	–	54.95	–	55.51	–

the nodes to classify them into a such large number of classes. Increasing the class tasks from 19 to 2983 sees a drop in performance by 14.93%, 12.38%, and 17.18% in GCN, SAGE, and GAT respectively, and 7.18%, 6.79%, and 7.23% in GCN, SAGE and GAT respectively. Some of these accuracies drop can be recovered by training for longer epochs. However, fundamentally, better models are required to predict fine-class tasks.

The IGB-small dataset is the most similar in terms of size to the homogeneous version of MAG240M [8]. In terms of performance, they also closely correlate as both have been curated with similar input data [8]. IGB-medium and the larger models have been trained for 3 epochs each with all three models. IGB-large and full are currently extremely hard to train in the available hardware resources at our disposal. We could train for three epochs with 19 out-class tasks. However, training 2983 class tasks in a single node system requires system innovations.

CHAPTER 5

FUTURE WORK

This chapter discusses the next step in IGB dataset generation work.

5.1 IGB260M Availability

IGB datasets are hosted on AWS S3. We have disabled the public download of the dataset at the moment and will be enabled in the near future after the publication of the work is released. We are also committed to releasing parts of raw text data that were used for generating embedding vectors along with this release. We are currently working through the legal steps to ensure we do not violate any policies. On completion, the IGB users will have access to the raw text and embedding vectors. Hopefully, this enables more groundbreaking work where we can see combined training of NLP and GNN models together.

5.2 Heterogeneous Graph Dataset

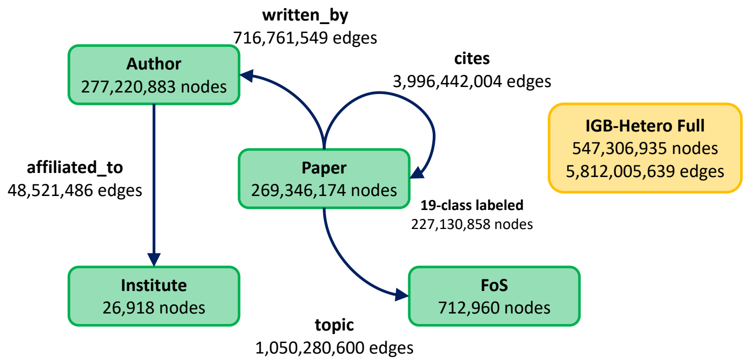


Figure 5.1: Illinois Graph Benchmark Heterogeneous Dataset Schema

When we started the thesis, we wanted to generate the largest homogeneous and heterogeneous GNN dataset. This thesis covered an in-depth discussion on homogeneous dataset generation. We are currently working to curate a heterogeneous dataset. Our current estimate of the heterogeneous dataset will have more than 600M nodes and six billion edges with multiple relation types and nodes. We will be providing a multiclass classification task to start with where the models require to predict paper topics. Further tasks such as reviewer recommendation and citation recommendation that can be useful for the general community are also in pipeline.

5.3 Integration With Framework DataLoaders

IGB dataset loaders are currently integrated with the DGL dataset loaded. We are actively working to enable PyG dataset loaders. We hope to release them in the near future.

5.4 MultiGPU And MultiNode Support

IGB-dataset has only been evaluated with single-node settings. Multi-GPU and multi-node supports such as distributed training are not yet implemented. We are working closely with the NVIDIA GNN tool team to enable multi-GPU and multi-node-based distributed training frameworks. We hope to release these supports in the near future.

5.5 Discussion

As discussed in our §4, there is a huge scope for improvements and developments in both GNN model design and building efficient systems. We showed that labeled dataset helps in improving the performance of GNN models but yet we must create better GNN models to classify complex multiclass labels which is an open-ended problem in the IGB dataset. We hope model developers try more complex tasks provided in IGB and build better models tomorrow.

Apart from the GNN model development, the system's community can make an immediate impact by improving the overall training and inferencing execution time. The IGB dataset provides a set of tools and shows the limitations in the existing systems when working on a massive scale and we hope the system community builds solutions to them.

There is a lot of promise in the application of GNNs. For example, the IGB dataset with the raw text can be used to fine-tune large language models and multi-lingual models or creation of combined training tasks. IGB datasets can serve as knowledge graphs where we can build future systems like reviewer and citation recommendations systems. These require creating custom tasks that are not currently defined in IGB datasets. We hope to add them in the future.

CHAPTER 6

CONCLUSION

Graphs are powerful data structures, and the ubiquity of Graph Neural Network applications in NLP, healthcare, social analysis, knowledge extraction, and fraud detection makes this an important field to study. However, the scarcity of large labeled datasets and lack of system optimizations make it challenging to develop new GNN models. With the introduction of the IGB dataset suite for both GNN researchers and system designers, we hope to alleviate this issue. In the process of creating the IGB dataset, we also performed several ablation studies pertaining to dataset creation and provided insights. The IGB dataset comes with raw text data that we believe will foster the emerging intersection of both NLP and GNN research topics. IGB dataset can be publicly accessed at `https://github.com/IllinoisGraphBenchmark`.

REFERENCES

[1] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," CoRR, vol. abs/1806.01973, 2018. [Online]. Available: http://arxiv.org/abs/1806.01973

[2] J. Wang, K. Ding, L. Hong, H. Liu, and J. Caverlee, "Next-item recommendation with sequential hypergraphs," in Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, ser. SIGIR '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3397271.3401133 p. 1101–1110.

[3] M. Yoon, T. Gervet, B. Shi, S. Niu, Q. He, and J. Yang, "Performance-adaptive sampling strategy towards fast and accurate graph neural networks," in Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery Data Mining, ser. KDD '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3447548.3467284 p. 2046–2056.

[4] W. Jin, J. Wohlwend, R. Barzilay, and T. S. Jaakkola, "Iterative refinement graph neural network for antibody sequence-structure co-design," in International Conference on Learning Representations, 2022. [Online]. Available: https://openreview.net/forum?id=LI2bhrE_2A

[5] H. Stärk, O.-E. Ganea, L. Pattanaik, R. Barzilay, and T. Jaakkola, "Equibind: Geometric deep learning for drug binding structure prediction," 2022. [Online]. Available: https://arxiv.org/abs/2202.05146

[6] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," 2019. [Online]. Available: https://arxiv.org/abs/1911.06455

[7] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec, "Ogb-lsc: A large-scale challenge for machine learning on graphs," 2021. [Online]. Available: https://arxiv.org/abs/2103.09430

[8] 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," CoRR, vol. abs/2005.00687, 2020. [Online]. Available: https://arxiv.org/abs/2005.00687

[9] Y. Li, J. Yin, and L. Chen, "Informative pseudo-labeling for graph neural networks with few labels," arXiv preprint arXiv:2201.07951, 2022.

[10] Z. Hu, Y. Dong, K. Wang, K.-W. Chang, and Y. Sun, "Gpt-gnn: Generative pre-training of graph neural networks," in Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery Data Mining, ser. KDD '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3394486.3403237 p. 1857–1867.

[11] J. Wang, Y. Guo, X. Wen, Z. Wang, Z. Li, and M. Tang, "Improving graph-based label propagation algorithm with group partition for fraud detection," Applied Intelligence, vol. 50, 10 2020.

[12] Y. Rao, X. Mi, C. Duan, X. Ren, J. Cheng, Y. Chen, H. You, Q. Gao, Z. Zeng, and X. Wei, Know-GNN: An Explainable Knowledge-Guided Graph Neural Network for Fraud Detection, 12 2021, pp. 159–167.

[13] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," AI Open, vol. 1, pp. 57–81, 2020.

[14] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," AI Open, vol. 1, pp. 57–81, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666651021000012

[15] W. Hamilton, P. Bajaj, M. Zitnik, D. Jurafsky, and J. Leskovec, "Embedding logical queries on knowledge graphs," in Advances in Neural Information Processing Systems, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/ef50c335cca9f340bde656363ebd02fd-Paper.pdf

[16] M. Zitnik, M. Agrawal, and J. Leskovec, "Modeling polypharmacy side effects with graph convolutional networks," Bioinformatics, vol. 34, no. 13, pp. i457–i466, 06 2018. [Online]. Available: https://doi.org/10.1093/bioinformatics/bty294

[17] S. Kumar, A. Mallik, A. Khetarpal, and B. Panda, "Influence maximization in social networks using graph embedding and graph neural network," Information Sciences, vol. 607, pp. 1617–1636, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025522006697

[18] Z. Chen, J. Xu, C. Alippi, S. X. Ding, Y. Shardt, T. Peng, and C. Yang, "Graph neural network-based fault diagnosis: a review," arXiv preprint arXiv:2111.08185, 2021.

[19] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," CoRR, vol. abs/1609.02907, 2016. [Online]. Available: http://arxiv.org/abs/1609.02907

[20] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," CoRR, vol. abs/1706.02216, 2017. [Online]. Available: http://arxiv.org/abs/1706.02216

[21] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2017. [Online]. Available: https://arxiv.org/abs/1710.10903

[22] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. v. d. Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," 2017. [Online]. Available: https://arxiv.org/abs/1703.06103

[23] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," in Proceedings of The Web Conference 2020, 2020, pp. 2704–2710.

[24] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019. [Online]. Available: https://arxiv.org/abs/1912.01703

[25] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," 2016. [Online]. Available: https://arxiv.org/abs/1603.04467

[26] M. Y. Wang, "Deep graph library: towards efficient and scalable deep learning on graphs," ICLR Workshop on Representation Learning on Graphs and Manifolds. [Online]. Available: https://par.nsf.gov/biblio/10311680

[27] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," 2019. [Online]. Available: https://arxiv.org/abs/1903.02428

[28] Y. Ma, X. Liu, N. Shah, and J. Tang, "Is homophily a necessity for graph neural networks?" 2021. [Online]. Available: https://arxiv.org/abs/2106.06134

[29] H. Pei, B. Wei, K. C. Chang, Y. Lei, and B. Yang, "Geom-gcn: Geometric graph convolutional networks," CoRR, vol. abs/2002.05287, 2020. [Online]. Available: https://arxiv.org/abs/2002.05287

[30] Z.-A. Shen, T. Luo, Y.-K. Zhou, H. Yu, and P.-F. Du, "NPI-GNN: Predicting ncRNA–protein interactions with deep graph neural networks," Briefings in Bioinformatics, vol. 22, no. 5, 04 2021, bbab051. [Online]. Available: https://doi.org/10.1093/bib/bbab051

[31] C. L. Giles, K. D. Bollacker, and S. Lawrence, "Citeseer: An automatic citation indexing system," in Proceedings of the Third ACM Conference on Digital Libraries, ser. DL '98. New York, NY, USA: Association for Computing Machinery, 1998. [Online]. Available: https://doi.org/10.1145/276675.276685 p. 89–98.

[32] P. Sen, G. M. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," AI Magazine, vol. 29, no. 3, pp. 93–106, 2008.

[33] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, June 2014.

[34] K. Lo, L. L. Wang, M. Neumann, R. Kinney, and D. S. Weld, "S2orc: The semantic scholar open research corpus," arXiv preprint arXiv:1911.02782, 2019.

[35] A. D. Wade, "The semantic scholar academic graph (s2ag)," in Companion Proceedings of the Web Conference 2022, 2022, pp. 739–739.

[36] S. Fricke, "Semantic scholar," Journal of the Medical Library Association: JMLA, vol. 106, no. 1, p. 145, 2018.

[37] S. P. Kolodziej, M. Aznaveh, M. Bullock, J. David, T. A. Davis, M. Henderson, Y. Hu, and R. Sandstrom, "The suitesparse matrix collection website interface," Journal of Open Source Software, vol. 4, no. 35, p. 1244, 2019.

[38] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," CoRR, vol. abs/1908.10084, 2019. [Online]. Available: http://arxiv.org/abs/1908.10084

[39] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018. [Online]. Available: https://arxiv.org/abs/1810.04805

[40] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019. [Online]. Available: https://arxiv.org/abs/1907.11692

[41] A. Cohan, S. Feldman, I. Beltagy, D. Downey, and D. S. Weld, "Specter: Document-level representation learning using citation-informed transformers," 2020. [Online]. Available: https://arxiv.org/abs/2004.07180

[42] I. Beltagy, K. Lo, and A. Cohan, "Scibert: A pretrained language model for scientific text," 2019. [Online]. Available: https://arxiv.org/abs/1903.10676

[43] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," Journal of Machine Learning Research, vol. 9, no. 86, pp. 2579–2605, 2008. [Online]. Available: http://jmlr.org/papers/v9/vandermaaten08a.html

[44] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Huggingface's transformers: State-of-the-art natural language processing," 2019. [Online]. Available: https://arxiv.org/abs/1910.03771

[45] K. P. F.R.S., "Liii. on lines and planes of closest fit to systems of points in space," The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, vol. 2, no. 11, pp. 559–572, 1901.

[46] S. Raschka, J. Patterson, and C. Nolet, "Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence," arXiv preprint arXiv:2002.04803, 2020.

[47] A. Khatua, V. S. Mailthody, and W.-m. Hwu, "IGB260M - Massive Homogeneous Graph Dataset." [Online]. Available: https://github.com/IllinoisGraphBenchmark/IGB260M-Datasets

[48] P. Budzianowski and I. Vulić, "Hello, it's gpt-2 – how can i help you? towards the use of pretrained language models for task-oriented dialogue systems," 2019. [Online]. Available: https://arxiv.org/abs/1907.05774

[49] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," Nature, vol. 585, no. 7825, pp. 357–362, sep 2020. [Online]. Available: https://doi.org/10.1038%2Fs41586-020-2649-2

[50] S. W. Min, V. S. Mailthody, Z. Qureshi, J. Xiong, E. Ebrahimi, and W.-m. Hwu, "Emogi: Efficient memory-access for out-of-memory graph-traversal in gpus," 2020. [Online]. Available: https://arxiv.org/abs/2006.06890

[51] S. W. Min, K. Wu, S. Huang, M. Hidayetoğlu, J. Xiong, E. Ebrahimi, D. Chen, and W.-m. Hwu, "Pytorch-direct: Enabling gpu centric data access for very large graph neural network training with irregular accesses," 2021. [Online]. Available: https://arxiv.org/abs/2101.07956

[52] Z. Qureshi, V. S. Mailthody, I. Gelado, S. W. Min, A. Masood, J. Park, J. Xiong, C. Newburn, D. Vainbrand, I.-H. Chung, M. Garland, W. Dally, and W.-m. Hwu, "GPU-Orchestrated On-Demand High-Throughput Storage Access in the System Architecture(FULL DETAILED VERSION)." arXiv, 2022. [Online]. Available: https://arxiv.org/abs/2203.04910

[53] V. S. Mailthody, "Application support and adaptation for high-throughput accelerator orchestrated fine-grain storage access," Ph.D. dissertation, University of Illinois Urbana-Champaign, 2022.

[54] Z. Qureshi, “Infrastructure to enable and exploit gpu orchestrated high-throughput storage access on gpus,” Ph.D. dissertation, University of Illinois Urbana-Champaign, 2022.