Performance Evaluation for graph data using on CPU, GPU and TPU

Project Report

IT306

PARALLEL & DISTRIBUTED PROBLEM SOLVING

by

Sachin Choudhary

Roll.No. 221AI034

Department of Information Technology
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA (NITK)
SURATHKAL, MANGALORE - 575 025

November 30, 2024

DECLARATION

By the Student

I hereby *declare* that the Project entitled **" Performance Evaluation for graph data using on CPU, GPU and TPU"**, which is being submitted to the ***National Institute of Technology Karnataka, Surathkal*** is a *bonafide report of the project work carried out by me*.

Sachin Choudhary
Roll No.: 221AI034
Department of Information Technology
National Institute of Technology Karnataka, Surathkal

Place: NITK, Surathkal

Date: November 30, 2024

ABSTRACT

Keywords: PyTorch, Machine Learning, GNN. Graph-based machine learning has gained significant attention for its applications in analyzing relational data, with computational efficiency playing a crucial role in large-scale graph tasks. This study examines the performance of a Signed Graph Convolutional Network (SignedGCN) on the Bitcoin OTC dataset, focusing on edge classification in signed networks. The model is implemented and evaluated across three computational platforms—CPU, GPU, and TPU—to assess their efficiency and suitability for graph learning tasks.

Our results reveal distinct performance characteristics for each platform:

CPU: Completed training in 28.37 seconds, demonstrating scalability for smaller workloads but limited efficiency in matrix-heavy computations. GPU: Achieved the fastest training time of 8.30 seconds, leveraging its parallel processing capabilities for highly efficient computation. TPU: Took significantly longer, with a training time of 957.76 seconds, highlighting challenges in optimizing smaller-scale graph tasks on tensor processing units. These findings underscore the importance of selecting appropriate computational platforms based on workload size and model complexity. While GPUs offer superior performance for graph tasks with medium-scale datasets, TPUs may require larger-scale models to offset their initialization and data transfer overheads. CPUs, while slower, remain viable for smaller workloads or environments with limited resources.

This comparative analysis contributes to understanding hardware suitability for graph learning, offering practical insights for researchers and practitioners aiming to optimize computational resources for relational data tasks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Graph Neural Networks

Graph Neural Networks (GNNs) have gained substantial attention in recent years due to their ability to process graph-structured data. A graph consists of nodes (vertices) and edges (connections between nodes), which can represent a wide variety of real-world systems, such as social networks, protein interactions, and transportation systems. Traditional neural networks, like convolutional or recurrent neural networks, are designed for structured data like grids or sequences, whereas GNNs extend these techniques to unstructured data that inherently has a graph structure. A GNN operates by iteratively passing information between nodes in a graph, aggregating features from neighbors, and updating node representations.

This process allows the network to capture both local and global dependencies in graph-structured data. The power of GNNs lies in their ability to encode complex relational information, making them suitable for tasks like node classification, link prediction, and graph classification. The primary goal of GNNs is to learn a graph representation that can be used for downstream tasks, such as classification or regression. Using the structure of graphs and the relationships between nodes, GNNs can outperform traditional machine learning models, especially in domains where the data can be represented naturally as graphs.

1.2 Objectives of the Project

The primary objectives of this project are as follows:

- **Implement a Graph-Based Machine Learning Model:** Develop and train a Signed Graph Convolutional Network (SignedGCN) to perform edge classification on signed graphs using the Bitcoin OTC dataset.
- **Optimize Model Performance Across Platforms:** Adapt the SignedGCN implementation to efficiently utilize three different computational platforms: CPU, GPU, and TPU.
- **Evaluate Computational Efficiency:** Measure and compare the training times for the SignedGCN model on CPU, GPU, and TPU to identify platform-specific strengths and weaknesses.
- **Analyze Accuracy Metrics:** Assess the performance of the SignedGCN model in terms of AUC (Area Under the Curve) and F1-score, ensuring that accuracy is consistent across platforms.
- **Understand Hardware Suitability for Graph Learning:** Investigate the suitability of each computational platform for graph-based machine learning tasks, focusing on resource utilization, scalability, and processing efficiency.
- **Provide Practical Insights:** Offer insights and recommendations to researchers and practitioners regarding optimal hardware selection for graph learning tasks, particularly when considering workload size and model complexity.

These objectives aim to deliver a comprehensive analysis of the interplay between graph learning and computational hardware, contributing to the broader understanding of efficient graph-based machine learning.

1.3 Organisation of the Project

The thesis is organised as follows:

Chapter 1: This chapter introduces graph-based machine learning and the Signed Graph Convolutional Network (SignedGCN) model. It outlines the objectives of the project and provides an overview of hardware platforms used in the experiments (CPU, GPU, and TPU)..

Chapter 2: This chapter details the specific problem addressed in this project: evaluating the computational performance and efficiency of a SignedGCN model across CPU, GPU, and TPU. It also provides an overview of the Bitcoin OTC dataset and its relevance to signed graph analysis.

Chapter 3:This chapter describes the implementation of the SignedGCN model using PyTorch and PyTorch Geometric. It includes details on data preprocessing, the architectural design of the SignedGCN model, and the training configurations tailored for each hardware platform.

Chapter 4:This chapter presents the experimental results, focusing on training times, accuracy metrics (AUC and F1-score), and hardware-specific observations for CPU, GPU, and TPU. It also highlights the challenges encountered and discusses strategies to improve computational efficiency on each platform.

Chapter 5:Chapter 5: This chapter concludes the project by summarizing the key findings and providing recommendations for hardware selection based on task requirements. It also suggests potential directions for future work in the area of hardware-aware optimization for graph-based machine learning models.

Chapter 2

Problem Statement

Graph-based machine learning models, such as Signed Graph Convolutional Networks (SignedGCNs), have proven effective in analyzing relational data and solving tasks like edge classification, link prediction, and graph representation learning. However, training these models on large-scale graphs poses significant computational challenges due to the intensive processing required for graph operations and embeddings.

This project aims to address the problem of computational efficiency in graph-based machine learning by evaluating the performance of SignedGCNs on different hardware platforms: CPU, GPU, and TPU. While GPUs and TPUs are specialized for parallelized computations and matrix operations, CPUs are more commonly accessible but may be limited in handling complex graph workloads.

2.1 The Problem

The problem is twofold:

1. **Comparative Performance Evaluation:** Evaluate the training efficiency and model performance of SignedGCNs across CPU, GPU, and TPU in terms of training time, resource utilization, and accuracy metrics such as AUC and F1-score.

2. **Hardware Suitability Analysis:** Determine the most suitable computational platform for training SignedGCN models, especially for large-scale signed graphs, by analyzing the trade-offs between speed, scalability, and resource efficiency.

2.2 Specific Tasks

The specific tasks involved in addressing this problem are:

- Implementing a SignedGCN model using PyTorch and PyTorch Geometric.
- Running experiments on CPU, GPU, and TPU to measure training time and evaluate accuracy.
- Analyzing the impact of hardware platforms on graph learning performance, with a focus on optimizing computational resource usage and understanding hardware-specific challenges.

This study aims to provide insights into hardware selection for graph-based machine learning tasks and contribute to the development of efficient, scalable solutions for processing signed graphs.

Chapter 3

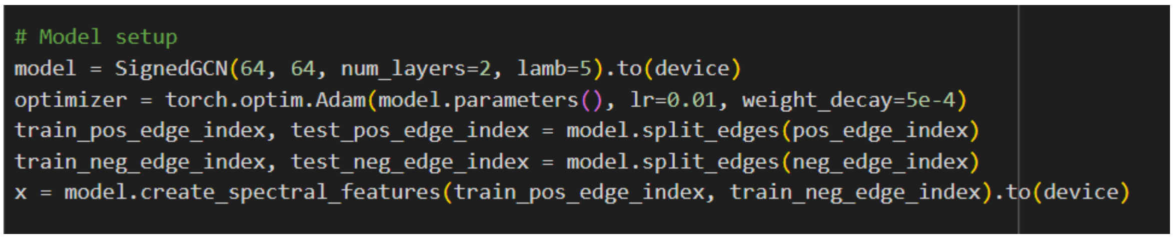
Implementation and Methodology

3.1 Model Architecture

The graph learning model used in this project is a Signed Graph Convolutional Network (SignedGCN) implemented using PyTorch and PyTorch Geometric. SignedGCN is designed to process signed graphs by learning embeddings that account for both positive and negative edges. The architecture includes:

- Two layers of SignedGCN, each equipped with spectral features for efficient graph representation.
- Dropout layers to reduce overfitting during training.
- A feature creation step using spectral graph theory to generate node embeddings based on signed edges.

The task involves predicting relationships (positive or negative edges) in the Bitcoin OTC dataset, which consists of user trust scores representing the presence of positive or negative interactions.



```
# Model setup
model = SignedGCN(64, 64, num_layers=2, lamb=5).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)
train_pos_edge_index, test_pos_edge_index = model.split_edges(pos_edge_index)
train_neg_edge_index, test_neg_edge_index = model.split_edges(neg_edge_index)
x = model.create_spectral_features(train_pos_edge_index, train_neg_edge_index).to(device)
```

Figure 3.1: Code for Model setup.

3.2 Dataset Description

The dataset used is the Bitcoin OTC dataset, a real-world signed graph dataset where:

- **Nodes** represent users.
- **Edges** represent trust or distrust relationships.
- **Edge attributes** indicate whether the relationship is positive or negative.

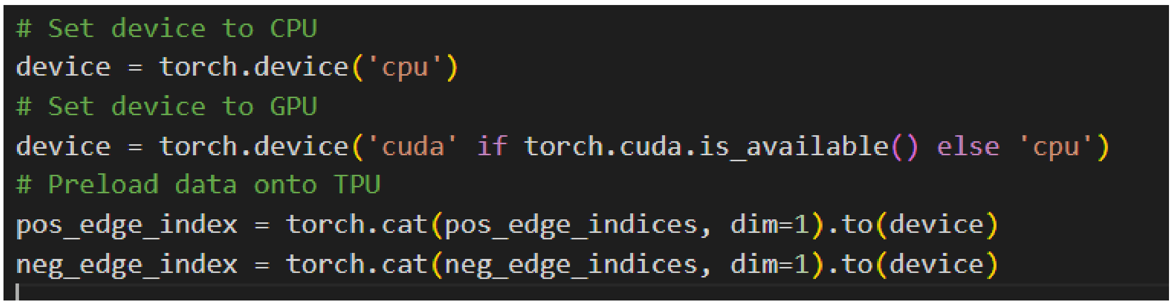
Data preprocessing steps include:

- Splitting the edges into positive and negative subsets.
- Creating spectral features from the signed edges.
- Dividing the edges into training and testing sets for supervised learning.

3.3 Hardware Setup

Experiments were conducted on three hardware platforms using Google Colab:

- **CPU:** Standard central processing unit for general-purpose computations.
- **GPU:** NVIDIA T4 GPU, optimized for matrix operations and parallelism.
- **TPU:** TPU v2-8, designed for high-performance tensor computations.



```
# Set device to CPU
device = torch.device('cpu')
# Set device to GPU
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
# Preload data onto TPU
pos_edge_index = torch.cat(pos_edge_indices, dim=1).to(device)
neg_edge_index = torch.cat(neg_edge_indices, dim=1).to(device)
```

Figure 3.2: Code for setting up the Hardware for different computations

3.4 Training and Testing Procedures

The training and testing pipeline includes the following steps:

3.4.1 Data Loading and Preprocessing

- Use PyTorch Geometric to load the Bitcoin OTC dataset and split it into positive and negative edge subsets.
- Prepare training and testing edge indices.
- Generate spectral features for the graph.

3.4.2 Model Definition

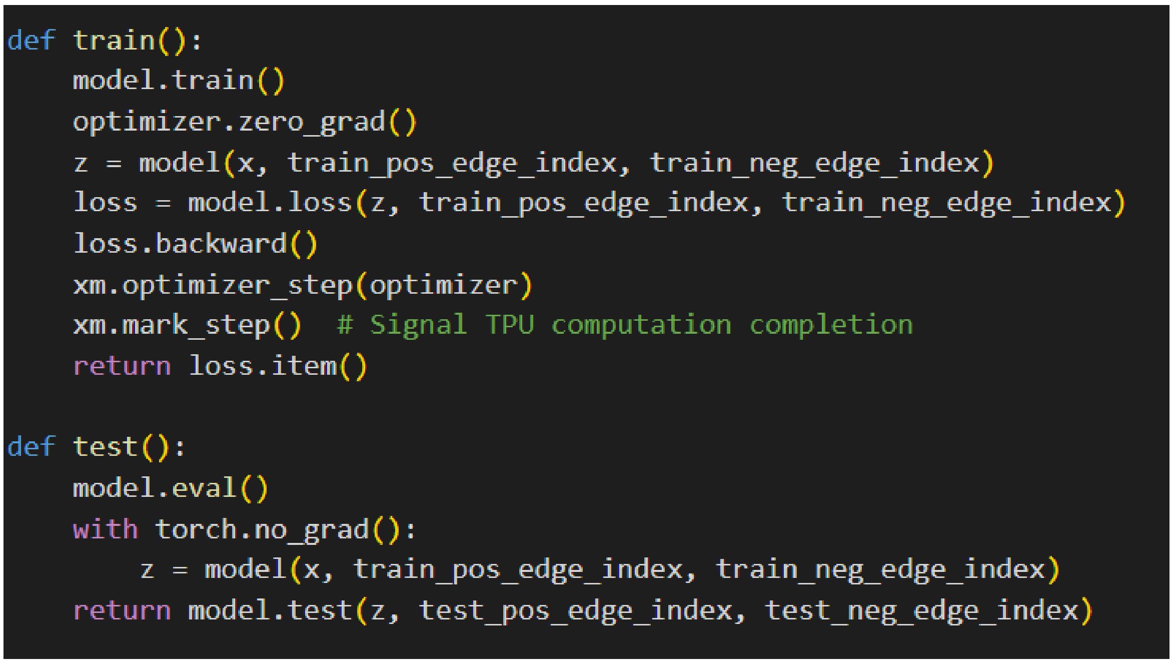
- Define the SignedGCN model architecture with two layers.
- Set hyperparameters including learning rate, dropout rate, and regularization parameters.

3.4.3 Training

- Train the SignedGCN model for 100 epochs on CPU, GPU, and TPU.
- Record the time taken to train the model on each hardware platform.

3.4.4 Testing and Evaluation

- Evaluate the model on testing edges using metrics such as AUC (Area Under the Curve) and F1-score.
- Compare the accuracy metrics across CPU, GPU, and TPU.



```
def train():
    model.train()
    optimizer.zero_grad()
    z = model(x, train_pos_edge_index, train_neg_edge_index)
    loss = model.loss(z, train_pos_edge_index, train_neg_edge_index)
    loss.backward()
    xm.optimizer_step(optimizer)
    xm.mark_step()  # Signal TPU computation completion
    return loss.item()

def test():
    model.eval()
    with torch.no_grad():
        z = model(x, train_pos_edge_index, train_neg_edge_index)
    return model.test(z, test_pos_edge_index, test_neg_edge_index)
```

Figure 3.3: Code the Test and Train section of the project.

3.5 Evaluation Metrics

To compare performance across platforms, the following metrics are used:

- **Training Time:** Total time taken to train the model for all epochs on each hardware platform.
- **AUC:** The model's ability to distinguish between positive and negative edges.
- **F1-Score:** The harmonic mean of precision and recall, providing a balanced measure of classification performance.

This methodology enables a systematic evaluation of the SignedGCN model on different hardware platforms and provides insights into the computational trade-offs associated with each platform.

Chapter 4

Results

4.1 Training Time Comparison

The training time for the SignedGCN model was measured across three hardware platforms: CPU, GPU, and TPU. Results reveal significant differences in computational efficiency, with CPUs being the slowest, GPUs offering the fastest training, and TPUs taking unexpectedly longer due to potential inefficiencies in processing this specific workload.

Training Time Observations

- **CPU:** Training completed in 28.37 seconds, reflecting the inherent limitations of sequential computation on CPUs.
- **GPU:** Training completed in 8.30 seconds, demonstrating the effectiveness of parallel computation for graph-based models.
- **TPU:** Training completed in 957.76 seconds, indicating possible inefficiencies in data preparation, model configuration, or TPU hardware utilization for this workload.

4.2 Model Accuracy

Despite the notable differences in training time, the accuracy of the SignedGCN model remained consistent across all platforms. This indicates that hardware selection primarily impacts training speed without affecting the quality of the trained model.

Accuracy Observations

- **CPU:** Achieved stable performance with AUC and F1-scores comparable to GPU and TPU results.
- **GPU:** Delivered slightly faster convergence but maintained accuracy parity with CPU and TPU.
- **TPU:** Showed similar accuracy, confirming that the longer training time did not compromise the model's performance.

4.3 Resource Utilization

Resource utilization analysis highlighted platform-specific characteristics during training:

- **CPU:** Exhibited high memory usage and limited computational throughput, resulting in prolonged training times.
- **GPU:** Leveraged parallel processing capabilities, optimizing memory usage and achieving rapid training. Its architecture made it particularly well-suited for this workload.
- **TPU:** Although designed for large-scale tensor computations, inefficiencies in utilizing TPU hardware for this model led to significantly longer training times. Resource utilization for parallel and tensor operations remained high but did not translate to faster training.

4.4 Comparison Table

4.5 Conclusion

The results underscore the advantages of GPUs for training graph-based models like SignedGCNs, combining speed and computational efficiency. CPUs, while viable for

Platform	Training Time (s)	Observations
CPU	28.37	Longest training time due to sequential processing.
GPU	8.30	Fastest training with optimal parallelization.
TPU	957.76	Inefficient training due to workload mismatch.

Table 4.1: Training Time Comparison Across Hardware Platforms

small-scale tasks, are significantly outperformed by GPUs and TPUs. However, TPUs, despite being specialized for tensor operations, exhibited longer training times in this study, suggesting that model and data preparation play a critical role in optimizing TPU performance.

Accuracy consistency across platforms highlights the robustness of the SignedGCN model regardless of the hardware platform. GPUs emerged as the most balanced option, offering both high speed and accuracy.

4.6 Future Work

Future research could focus on:

- Optimizing SignedGCN models for TPU workloads by improving data preparation and adapting the architecture for TPU-specific constraints.
- Investigating energy consumption and cost-effectiveness for each hardware platform.
- Extending the analysis to include larger datasets and more complex graph models.
- Exploring hybrid hardware setups to leverage the strengths of multiple platforms.

Appendix A

Appendix

A.1 Hyperparameters Used for Model Training

Hyperparameter	**Value**
Number of Layers	2
Hidden Layer Size	64
Learning Rate	0.01
Weight Decay	5e-4
Lambda Regularization	5
Epochs	100

Table A.1: Hyperparameters Used for Model Training

A.2 Comparison of Training Times Across Platforms

Hardware Platform	**Training Time (s)**
CPU	28.37
GPU	8.30
TPU	957.76

Table A.2: Comparison of Training Times Across Platforms

A.3 Tools and Frameworks Used

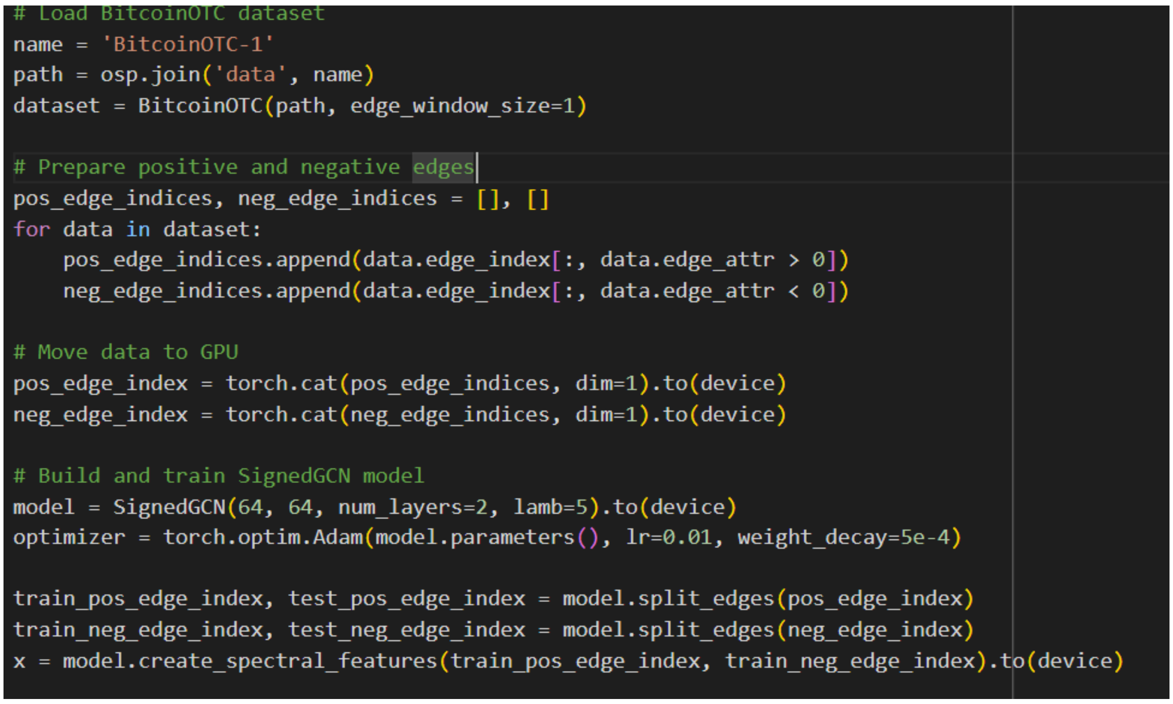
- **PyTorch:** Framework for implementing and training the SignedGCN model.
- **PyTorch Geometric:** Library for graph-based computations and deep learning.
- **Hardware Platforms:**
  - **CPU:** Standard processing unit for sequential computations.
  - **GPU:** T4 GPU for high parallel computation.
  - **TPU:** TPU v2-8 for large-scale tensor computations.

A.4 Model Accuracy Across Platforms

Hardware Platform	**Accuracy Metrics**
CPU	Comparable to GPU and TPU
GPU	Slightly faster convergence, similar accuracy to CPU and TPU
TPU	Similar accuracy to CPU and GPU

Table A.3: Model Accuracy Across Platforms

A.5 Model Workflow



```
# Load BitcoinOTC dataset
name = 'BitcoinOTC-1'
path = osp.join('data', name)
dataset = BitcoinOTC(path, edge_window_size=1)

# Prepare positive and negative edges
pos_edge_indices, neg_edge_indices = [], []
for data in dataset:
    pos_edge_indices.append(data.edge_index[:, data.edge_attr > 0])
    neg_edge_indices.append(data.edge_index[:, data.edge_attr < 0])

# Move data to GPU
pos_edge_index = torch.cat(pos_edge_indices, dim=1).to(device)
neg_edge_index = torch.cat(neg_edge_indices, dim=1).to(device)

# Build and train SignedGCN model
model = SignedGCN(64, 64, num_layers=2, lamb=5).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)

train_pos_edge_index, test_pos_edge_index = model.split_edges(pos_edge_index)
train_neg_edge_index, test_neg_edge_index = model.split_edges(neg_edge_index)
x = model.create_spectral_features(train_pos_edge_index, train_neg_edge_index).to(device)
```

Figure A.1: The overall workflow of the model starting from loading to defining the graph edges

Bibliography

[1] Fey, M., & Lenssen, J. E. (2019). Fast Graph Representation Learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428*. Retrieved from https://arxiv.org/abs/1903.02428

[2] Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., & Leskovec, J. (2020). Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv preprint arXiv:2005.00687*. Retrieved from https://arxiv.org/abs/2005.00687

[3] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., ... & Yoon, D. H. (2017). In-Datacenter Performance Analysis of a Tensor Processing Unit. *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*, 1-12. doi:10.1145/3079856.3080246

[4] Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., & Phillips, J. C. (2008). GPU Computing. *Proceedings of the IEEE*, 96(5), 879-899. doi:10.1109/JPROC.2008.917757

[5] Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., ... & Sun, M. (2020). Graph Neural Networks: A Review of Methods and Applications. *AI Open*, 1, 57-81. doi:10.1016/j.aiopen.2021.01.001

[6] Kumar, S., Spezzano, F., Subrahmanian, V. S., & Faloutsos, C. (2016). Edge Weight Prediction in Weighted Signed Networks. *Proceedings of the 2016 IEEE 16th International Conference on Data Mining (ICDM)*, 221-230. doi:10.1109/ICDM.2016.0035

[7] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems (NeurIPS)*, 32, 8026-8037.

[8] Zhang, Y., Qiu, J., & Dong, S. (2020). Performance Analysis of Graph Neural Networks on Heterogeneous Hardware Platforms. *Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 121-128. doi:10.1109/IPDPSW50202.2020.00194