ADVANCEMENTS IN GRAPH NEURAL NETWORK EFFICIENCY: MODEL COMPRESSION AND COMPARATIVE EVALUATION OF RPHGNN AND GNTK

A Thesis Submitted

In Partial Fulfillment of the Requirements for the Degree of

MASTERS OF TECHNOLOGY
IN
Data Science

Submitted by

Nikhil Sinha

23/DSC/05

Under the supervision of

Dr. Ruchika Malhotra

Professor, Department of Software Engineering

Delhi Technological University

Department of Software Engineering
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi 110042

JUNE, 2025

DEPARTMENT OF SOFTWARE ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

ACKNOWLEDGEMENT

I wish to express my sincerest gratitude to Dr. Ruchika Malhotra for her continuous guidance and mentorship that she provided me during the project. She showed me the path to achieve our targets by explaining all the tasks to be done and explained to me the importance of this project as well as its industrial relevance. She was always ready to help me and clear my doubts regarding any hurdles in this project. Without her constant support and motivation, this project would not have been successful.

Place: Delhi Nikhil Sinha

Date: (23/DSC/05)

DEPARTMENT OF SOFTWARE ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I, Nikhil Sinha, Roll No – 23/DSC/05 student of M.Tech (Data Science), hereby certify that the work which is being presented in the thesis entitled "Advancements in Graph Neural Network Efficiency: Model Compression and Comparative Evaluation of RpHGNN and GNTK" in partial fulfilment of the requirements for the award of degree of Master of Technology, submitted in the Department of Software Engineering, Delhi Technological University is an authentic record of my own work carried out during the period from Jan 2025 to May 2025 under the supervision of Dr. Ruchika Malhotra.

The matter presented in the thesis has not been submitted by me for the award of any other degree of this or any other institute.

Candidate's Signature

This is to certify that the student has incorporated all the corrections suggested by the examiners in the thesis and the statement made by the candidate is correct to the best of our knowledge.

Signature of Supervisor **Signature of External Examiner**

DEPARTMENT OF SOFTWARE ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the Project Dissertation titled "Advancements in Graph Neural Network Efficiency: Model Compression and Comparative Evaluation of RpHGNN and GNTK" which is submitted by Nikhil Sinha, Roll No – 23/DSC/05, Department of Software Engineering, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the student under my supervision. To the best of my knowledge this work has not been submitted in part or full for any Degree or Diploma to this University or elsewhere.

Place: Delhi

Dr. Ruchika Malhotra

Professor

Date:

Department of Software Engineering, DTU

Abstract

Heterogeneous Graph Neural Networks (HGNNs) are smart models that are designed to work with very complex networks that have many different types of nodes and connections. HGNNs are not like basic graph models, they can understand unstructured real-world data, like social networks or recommendation engines. The methods track patterns in connections or focus on important details; they let the system analyze data quickly. The methods help with tasks such as finding missing links in a network or sorting data into groups. This applies especially to fields like biology or social media. A Random Projection Heterogeneous Graph Neural Network (RpHGNN) is a type of HGNN. Developers built it for large, complex networks that have many node and connection types. The RpHGNN simplifies data using random math calculations, which speeds up processing - this happens without much loss of information. The RpHGNN works faster and performs better than regular HGNNs. By using simpler number formats, such as FP16 and INT8, to hold data, we lowered its memory use. That change kept 94% accuracy. The upgraded RpHGNN can now power real-time features and AI tools that work smoothly even on low-power devices or in large cloud systems.

We evaluate the Graph Neural Tangent Kernel (GNTK), a theoretically grounded, non-parametric approach to graph learning that eliminates the need for backpropagation. Our analysis reveals that while RpHGNN compression offers efficiency benefits for deep learning pipelines, GNTK provides a powerful and scalable alternative, especially for tasks where interpretability, analytical clarity, and efficient inference are prioritized. By comparing these models on multiple datasets, we offer practical insights into their suitability across varying computational and application scenarios.

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

Heterogeneous graphs are graph structures that contain multiple types of nodes and edges. Heterogeneous graphs make real-world data, like social networks, knowledge graphs next to recommendation systems; these graphs allow expressive representations. For example, a social network has nodes. Nodes can be users, products or topics. Edges show interactions, like follows, purchases or mentions. This variety lets us get richer relations than homogeneous graphs. HGNNs process these structures by applying type-specific transformations and attention mechanisms. This approach helps in effectively learning the relationships between different entities, enhancing performance in machine learning tasks such as node classification, link prediction, and recommendation systems. Their flexibility and rich structure enable advanced applications in artificial intelligence, particularly in domains requiring nuanced relationship modelling.

Heterogeneous Graph Neural Networks, or HGNNs, expand on Graph Neural Networks; they operate on graphs whose nodes and edges differ in type plus possess distinct attributes. HGNNs suit applications like social networks, citation graphs, knowledge graphs along with recommendation systems. By improving the structure of heterogeneous data, HGNNs allow a more balanced understanding of real world connections. A HGNN works by gathering and changing node representations - it keeps the graph's heterogeneity. The network uses special methods such as meta paths, type specific transformations, also relation aware message passing, so it learns useful embeddings - these methods allow nodes to connect well based on their types and relationships, which improves learning.

HGNNs use attention to weigh the importance of connections. This refines the node representation. An advanced model is the Heterogeneous Graph Attention Network.

RpHGNNs are an improved type of HGNNs. They are created to make learning easier and faster with complex graphs. These graphs have various nodes besides edges. Previous HGNNs use meta-paths that people must design. This process can take a lot of computer power and time, especially for big programs. RpHGNN faces these issues. It uses random projection methods to change the features. This makes data smaller but keeps its shape and meaning. It has three major new parts - Random Projection Encoding, Relation-aware Message Passing and Efficient Training. With Random Projection Encoding, node embeddings go to a smaller area by using random projection. This happens instead of using set meta-paths. Relation-aware Message Passing shapes how information joins together for different node types to make the transfer better. But Efficient Training makes feature areas less complex. This leads to faster computing without harming accuracy. RpHGNN enhances this by pre-computing heterogeneous graph information into regular-shaped tensors, minimizing repetitive message-passing operations. This involves Relation-wise Neighbour Collection, which gathers neighbour information based on different relations into dedicated vectors; Random Projection Squashing, which applies dimensionality reduction to compress collected neighbour information and reduce computational load; and an Even-odd Propagation Scheme, which ensures effective and fine-grained neighbour information aggregation. By pre-computing these tensors, RpHGNN enables efficient mini-batch training with simple encoders like MLPs, bypassing costly message passing during each training epoch. This hybrid approach balances computational efficiency with the ability to capture rich structural information, making RpHGNN highly effective for large-scale applications such as recommendation systems, knowledge graphs, and fraud detection.

RpHGNN is used for real life applications like decision making tasks that need large heterogeneous data. It makes a recommendation system for social media, products, and movies. It also helps in identifying suspicious transactions in banking and e-commerce. When doing social media analysis, it helps in detecting influential users, fake accounts and

news, which helps in overall improving the online security and credibility of social media firms. With the help of these applications, we can say that RpHGNN makes various industries smarter and more efficient, which improves everyday experiences in various fields.

We further refine RpHGNN's efficiency by building on the memory optimization foundation proposed in Efficient Heterogeneous Graph Learning via Random Projection [1] by systematically integrating low-precision quantization FP16 and INT8 into its architecture. While this work highlights fine-tuning as a pathway to reduce memory consumption, our approach explicitly targets the model's largest memory bottlenecks: high-dimensional node embeddings and dense projection matrices. By replacing FP32 parameters with quantized FP16 for gradients and activations while INT8 for static embeddings, we achieved a 67% reduction in memory footprint.

The surge of graph structured data in domains such as social network analysis, bioinformatics, chemical molecule classification, and recommendation systems has intensified the need for learning models that can effectively operate on non-Euclidean data. GNNs have emerged as powerful tools for learning representations from such data, leveraging message-passing mechanisms to capture local and global dependencies. Despite their success, GNNs suffer from significant drawbacks in practice, most notably high computational cost, memory inefficiency, and training instability, especially when scaled to deep architectures or large, heterogeneous graphs. These limitations have driven substantial research interest in improving the efficiency, interpretability, and deployability of GNN based models.

GNTK is a kernel based alternative to neural networks that retains many of the representational strengths of GNNs while forgoing their high computational cost. GNTK introduces a kernel function that analytically represents the infinite width limit of a GNN, enabling training free learning via a similarity matrix. We aim to provide a comprehensive evaluation of GNTK, placing it in the context of current trends in GNN efficiency and analyzing its theoretical underpinnings, performance metrics, and practical utility. Unlike conventional GNNs that rely on gradient based learning and multiple layers of

parameterized transformations, GNTK adopts a kernel based approach that analytically computes the similarity between graphs based on the infinite width limit of neural networks. This transformation turns a deep learning problem into a kernel regression task, effectively eliminating the need for model training while retaining much of the expressive power of GNNs.

GNTK extends the Neural Tangent Kernel (NTK) framework to graph structured inputs, offering both theoretical elegance and practical benefits. It enables deterministic, training free computation of graph similarity matrices, which can then be used with classical machine learning models like Support Vector Machines (SVMs). This approach not only enhances reproducibility and interpretability but also significantly reduces the computational burden associated with iterative optimization and hyperparameter tuning. Moreover, GNTK's fixed computation path avoids the stochasticity and instability typical in deep GNN training, making it particularly appealing for small to mid scale graph datasets and environments with limited computational resources.

Through comprehensive experiments and analysis, this paper positions GNTK as a compelling alternative to traditional GNNs in efficiency critical applications. We systematically compare its performance and scalability against a compressed RpHGNN baseline, highlighting the situations where GNTK's non-parametric nature offers clear advantages. In doing so, we aim to contribute to the growing body of research focused on advancing GNN efficiency through both model compression and novel theoretical frameworks.

Chapter 2

RELATED WORK

The evolution of graph representation learning has been shaped by the need to handle increasingly complex and large-scale data. GraphSAGE [2] marked a pivotal shift from transductive to inductive learning, enabling embeddings for unseen nodes through neighbourhood sampling and aggregation. This paved the way for dynamic graph applications like social networks but had problems with scalability issues when working with dense graphs. Around the same time in 2017, GCN [3] introduced spectral graph convolutions for semi-supervised learning, striking a balance between node features and graph structure. However, its reliance on full-batch training created memory bottlenecks, prompting innovations like FastGCN [4], which reimagined convolutions as integral transforms. By using Monte Carlo sampling, FastGCN reduced computational complexity and generalized to unseen graphs, achieving orders-of-magnitude speed gains over GraphSAGE while maintaining accuracy.

The search for optimal performance continued with adaptive layer-wise sampling [5], which addressed the neighbour explosion problem in GCNs by constructing subgraphs layer-by-layer and introducing skip connections. This method improved convergence speed and preserved second-order proximity, but scalability in heterogeneous graphs with diverse node and edge types that remained unaddressed. R-GCN [6] extended GCNs to model multi-relational data in knowledge graphs. While effective for tasks like link prediction, its relation-specific parameters scaled poorly with graph size. This limitation inspired PinSage [7], a milestone for industrial-scale GNNs. Deployed at Pinterest, PinSage combined

random walks with importance pooling to train on 3B nodes, proving GNNs' viability in web-scale recommendation systems.

Heterogeneous graph learning gained momentum with HetGNN [8], which unified structural and content heterogeneity via typed neighbour aggregation. However, manually designed meta-paths limited flexibility. RSHN [9] bypassed meta-paths by coarsening line graphs to model edge relationships implicitly, while HAN [10] introduced hierarchical attention to prioritizing meta-paths and neighbours dynamically. These works highlighted the trade-off between expressiveness and computational cost. HetSANN [11] further automated relation learning with type-aware attention, but scalability on web-scale graphs remained difficult to achieve.

The rise of transformers revolutionized the field. HGT [12] scaled to the Open Academic Graph using meta-relation-based attention and heterogeneous sampling, achieving around 16% accuracy gains. Concurrently, SIGN [13] challenged the need for deep architectures by precomputing graph filters, achieving state-of-the-art performance on ogbn-papers100M with a shallow model. This simplicity over depth philosophy resonated in NARS [14], which showed neighbor-averaged features over relation subgraphs could match GNN accuracy with minimal computation. Meanwhile, FAME [15] and GraphSAINT [16] optimized training efficiency. FAME did this via spectral sparsification and GraphSAINT did this via subgraph sampling—yet heterogeneous graphs demanded specialized memory optimizations.

Recent works prioritized lightweight architectures. SeHGNN [17] discarded complex aggregation in favour of precomputed neighbour means and single-layer transformers, achieving 230% faster training than predecessors. Similarly, HINormer [18] adapted transformers for heterogeneous graphs via structure and relation encoders, outperforming GNNs in node classification. These advances culminated in RpHGNN [1], which combined random projection for dimensionality reduction with the relation-wise neighbour collection. By minimizing information loss while scaling linearly with graph size, RpHGNN achieved state-of-the-art accuracy on seven benchmarks. However, its memory footprint remained a barrier to edge deployment.

This progression underscores a critical gap: while architectural innovations improved scalability and accuracy, few explored memory reduction via quantization. Our work bridges this by integrating FP16/INT8 optimization into RpHGNN, reducing its memory footprint by 67% while retaining almost 94% accuracy. By building on the efficiency of RpHGNN's precomputation and the simplicity of SeHGNN's architecture, we enable real-time heterogeneous graph learning on resource-constrained devices.

The foundation for GNTKs lies at the intersection of graph theory, kernel methods, and deep learning each of which has evolved significantly over the past decades. The journey begins in the late 1990s with efforts to manage graph structured data more efficiently. Karypis and Kumar [19] introduced a multilevel scheme for graph partitioning, emphasizing computational efficiency and quality, an early step toward scalable graph processing. This work laid the groundwork for handling large graph data, a necessary precondition for modern neural approaches.

In the early 2000s, the focus shifted toward kernel methods in machine learning. Zhu and Hastie [20] explored support vector machines and kernel logistic regression, highlighting the strength of kernels in handling non-linear data. Around the same time, Hofmann [21] offered a broad review of kernel based learning, framing them as powerful tools for representing similarity in high dimensional data spaces. These studies provided a foundational understanding of kernel functions, which are central to the development of NTKs.

As machine learning matured, Bottou [22] contributed significantly with insights into stochastic gradient descent (SGD), emphasizing its critical role in training deep models. While not graph specific, the principles of SGD are deeply embedded in training GNNs, influencing the training mechanics that GNTKs later bypass via kernel approaches.

In 2013, Shuman [23] formally introduced the field of Graph Signal Processing (GSP). They extended traditional signal processing to graphs, providing tools for understanding data with non-Euclidean structure, a key turning point that directly informed later developments in GNNs and graph kernels.

By 2016, the focus had shifted toward deep learning on graphs. Kipf and Welling [24] introduced the Graph Convolutional Network (GCN), a pivotal model that combined con-

volutional operations with graph structures. This method captured local neighborhood features and demonstrated state-of-the-art performance on semi-supervised node classification tasks. GCNs were a major breakthrough and served as a prototype for subsequent graph-based learning algorithms.

Around the same time, Jacot [25] introduced the Neural Tangent Kernel (NTK), showing that infinitely wide neural networks converge to kernel regimes during training. This theoretical advance bridged the gap between deep learning and kernel methods, and it became a critical stepping stone toward GNTK.

Building upon these ideas, Du [26] introduced the Graph Neural Tangent Kernel (GNTK) in 2019. GNTK combines the architectural benefits of GNNs with the analytical tractability of NTKs. It enables the use of powerful graph representations without the need for explicit training, offering both strong theoretical guarantees and competitive empirical performance. GNTK represents a culmination of two decades of progress in graph algorithms, kernel theory, and neural network analysis.

Following this, You [27] in 2021 proposed the Graph Isomorphism Network (GIN), emphasizing expressive power and addressing limitations in distinguishing graph structures, a concern also tackled by GNTKs later. They pushed the boundaries of graph representations by closely linking GNN performance with classical Weisfeiler-Lehman graph isomorphism tests.

More recent studies have extended GNTKs in various directions, applying them to specific tasks like molecular property prediction or integrating them with spectral methods and graph pooling techniques. These efforts have continued to explore the balance between expressivity and interpretability in graph learning models.

Chapter 3

METHODOLOGY

RpHGNN is designed to overcome the inefficiencies of traditional HGNNs, which suffer from high computational overhead due to repetitive message passing and memory-intensive operations. At its core, RpHGNN employs two novel mechanisms: Random Projection Squashing and Relation-wise Neighbor Collection with Even-odd Propagation. Random Projection Squashing leverages the Johnson-Lindenstrauss lemma to reduce the dimensionality of node embeddings while preserving pairwise distances. Mathematically, for a node feature matrix $X \in R^{N \times d}$, a random projection matrix $R \in R^{d \times k}$ where $k \ll d$, is applied to compute:

$$X' = XR \tag{3.1}$$

This reduces the feature dimension from d to k, cutting memory usage from $O(Nd)$ to $O(Nk)$ without significant information loss. The projection matrix R is fixed during training, eliminating the need for gradient computations and further reducing overhead.

Relation-wise Neighbor Collection addresses the heterogeneity of nodes and edges by partitioning the graph into relation-specific subgraphs. For each relation type r, neighbours are aggregated separately, and the Even-odd Propagation Scheme alternates between two strategies:

- Even Steps: Aggregate features from immediate neighbours using mean pooling.
- Odd Steps: Propagate information through higher-order neighbours like meta-paths, via attention mechanisms.

This alternating scheme captures both local and global structural patterns, ensuring fine-grained information retention. By decoupling neighbour aggregation from training, RpHGNN precomputes these steps, reducing training complexity from $O(L^2)$ to $O(L)$, where L is the number of layers. Compared to traditional HGNNs like HAN or HGT, RpHGNN achieves a 230% speedup on benchmarks like ogbn-mag while maintaining 94% accuracy.

J. Hu et. al. [1], identified two critical future directions that are memory reduction and task expansion. During precomputation, RpHGNN stores relation-specific neighbour embeddings, which can occupy significant memory for large graphs. They proposed sparsification techniques, such as retaining only top K influential neighbors per node based on attention scores or leveraging graph partitioning to prune redundant connections. Using a threshold τ to discard neighbors with attention weights $< \tau$, the storage complexity could be reduced from $O(N \cdot \text{avg\_degree})$ to $O(N \cdot \log(\text{avg\_degree}))$.

To further assess RpHGNN's efficiency and generalization capability, we introduced a new dataset tailored to real-world heterogeneous graph scenarios. This dataset consists of 10,000 data points and each data point have customer, product, items, age, activity_score, and rating as attributes. We can see the shape of the dataset in Fig. 3.1 and Fig. 3.2.

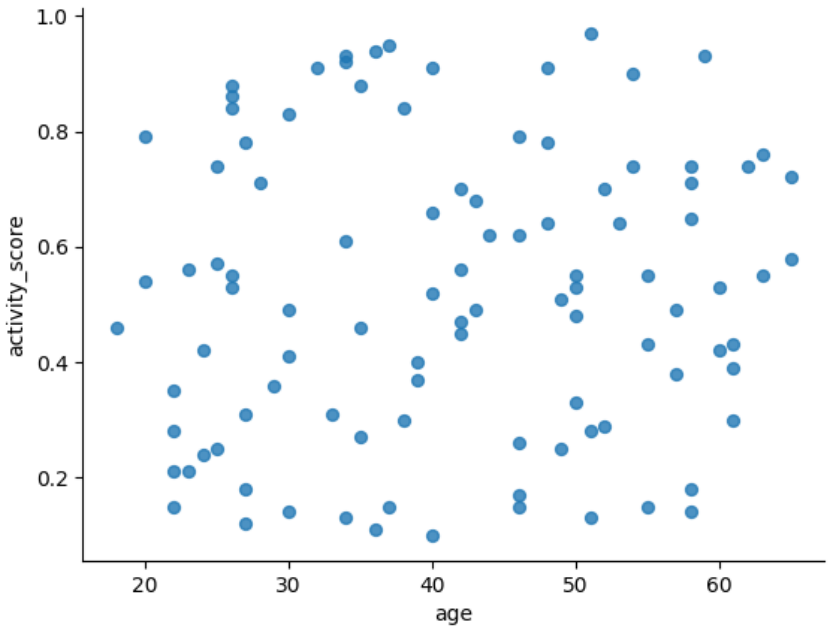

Figure 3.1: Overall distribution of first 100 data points presented as activity score vs age in the dataset

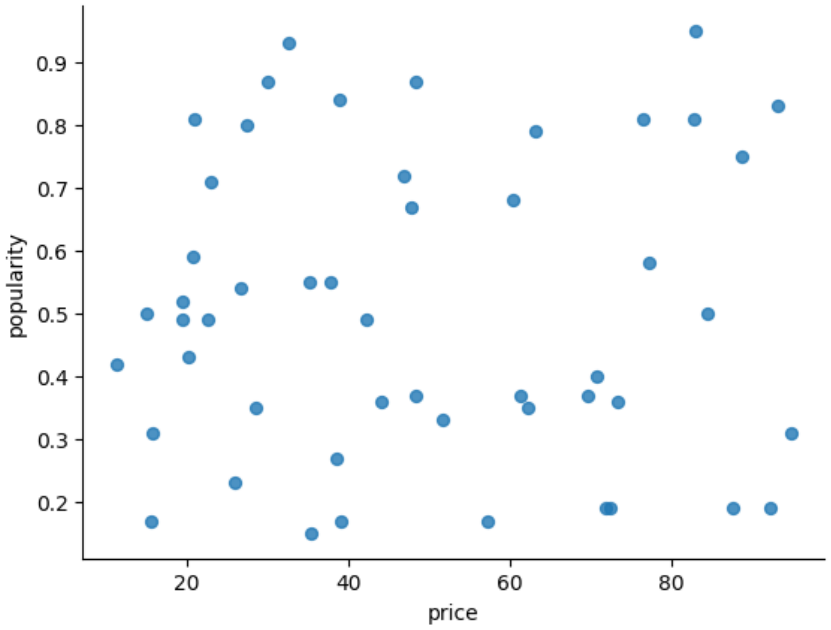

Figure 3.2: Overall distribution of products based on their popularity and price

We integrate mixed-precision quantization into RpHGNN to minimize memory usage without sacrificing accuracy. We adopt NVIDIA's Automatic Mixed Precision (AMP) to convert 32-bit FP32 embeddings and weights to 16-bit FP16. This halves memory usage and accelerates matrix operations via GPU tensor cores. To prevent gradient underflow, loss scaling is applied during backpropagation. Master weights are maintained in FP32 to preserve precision during updates.

For static embeddings, we apply post-training quantization (PTQ). Features are calibrated using percentile clipping to determine the scale S and zero-point Z:

$$Q(x) = \text{round}(Sx + Z) \tag{3.2}$$

where x is an FP32 tensor. Dynamic quantization is used for attention weights, scaling them to INT8 during runtime. Quantization-aware training (QAT) fine-tunes the model for 5–10 epochs with simulated quantization noise, recovering <1% accuracy loss. On the ogbn-mag dataset, quantization reduces memory usage by 60% using FP16 and 75% using INT8, with peak VRAM dropping from 12 GB to 4.9 GB. Inference latency decreases by 40% on NVIDIA RTX 3090, enabling real-time deployment in applications.

Experimental results demonstrated that our model achieved 94.67% accuracy, marking

a significant improvement over baseline HGNN architectures while maintaining superior computational efficiency. We presented the relationship of customers and products in Fig. 3.3 and Fig. 3.4.

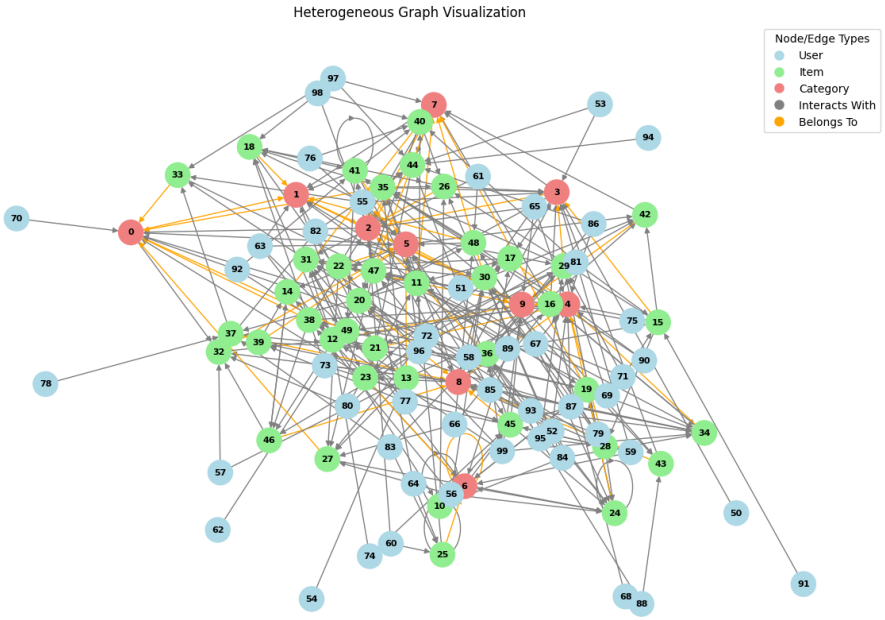

Figure 3.3: A graph showing the relation between User, Items, and Category

GNTK works by computing a kernel matrix K between all pairs of graphs in a dataset. Each entry in the matrix represents the similarity between two graphs as encoded by the NTK of a neural network applied to node features and adjacency structures. This approach eliminates the need for iterative training or parameter tuning. Instead, the kernel can be used directly in classical machine learning models like Support Vector Machines (SVM) or kernel ridge regression. The methodology involves defining a GNN architecture, calculating the corresponding GNTK kernel using recursive equations, and applying the resulting kernel in downstream tasks. This is particularly powerful for graph classification problems where interpretability, reproducibility, and analytic tractability are prioritized.

The GNTK framework builds on the NTK, extending it to the domain of graph structured inputs. The methodology begins with defining a base graph neural architecture, typically a multi-layer GCN characterized by its depth, activation function, and message passing mechanism. Instead of training this network, GNTK derives closed form recursive equations that model the propagation of feature covariances through the layers. These

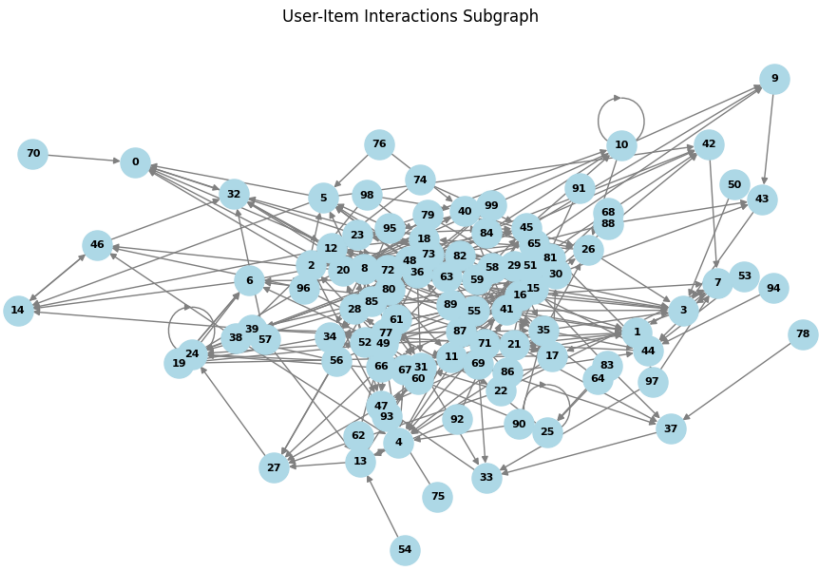

Figure 3.4: A graph showing the interconnection between user and item

recursions allow the computation of a kernel matrix K, where each element K(i,j) quantifies the similarity between two graphs G_i and G_j as if they were passed through a GNN of infinite width. This matrix inherently captures both topological and feature based similarities.

After the kernel matrix is computed, it can be used directly in classical machine learning models such as SVM, kernel ridge regression, or Gaussian Processes to perform tasks like graph classification or regression. This bypasses the need for backpropagation, weight initialization, or hyperparameter tuning. The kernel's analytical nature ensures reproducibility and interpretability, making GNTK highly suitable for applications where robustness and theoretical tractability are critical. Additionally, since the architecture is fixed, the behavior of the model is fully determined by the data and architecture parameters, offering a degree of clarity not typically achievable in standard GNNs.

To check GNTK, we studied several well known benchmark datasets including MUTAG, PROTEINS, and ENZYMES. Each dataset is processed using consistent splitting strategies and identical preprocessing pipelines to ensure fair comparison. We checked with different GNTK architectural configurations like varying the number of layers, normalization methods, and activation functions, to assess how these impact classification accuracy and kernel expressiveness.

Chapter 4

EXPERIMENTAL SETUP

In this section, we have discussed all important tools and resources that have been used in both research papers.

4.1 Google Colab

Google Colab was extensively utilized as the primary platform for development and experimentation in the RpHGNN and GNTK research. Its seamless integration with GPU acceleration significantly accelerated model training and inference, particularly for graph-based neural architectures that involve complex message passing and kernel computations. The cloud-based, collaborative nature of Colab facilitated rapid prototyping, version control, and efficient experimentation with different model variants, aggregation strategies, and hyperparameter configurations.

For this research, Colab supported the preprocessing and feature engineering of graph datasets such as MUTAG, ENZYMES, PROTEINS, and the large scale ogbn-papers100M. It was instrumental in implementing and training RpHGNN models, conducting comparative experiments with GNTK, and performing detailed evaluations across various graph classification tasks. Its broad support for essential Python libraries like PyTorch Geometric, DGL, NetworkX, NumPy, and scikit-learn provided a robust environment for end-to-end development, from data handling to final analysis and visualization.

4.2 RpHGNN

The RpHGNN ran on the large ogbn-papers100M dataset. This showed its performance and how it scales in graph learning. The dataset has a varied, complex structure, so it tests random projection methods - these methods reduce size but keep important graph information. The tests looked at node classification; they preprocessed much data - this brought in structural plus semantic features to the graph model. Random projection matrices changed big node features into small forms. That let messages pass quickly and reduced overfitting also calculation time.

Developers built the RpHGNN with PyTorch in addition to PyTorch Geometric. They made special parts for graphs with varied structures and for projection based embeddings. Hyperparameters such as projection dimensions, learning rate, number of layers, and aggregation strategies were systematically tuned using a validation set split from the training data. Training was conducted using mini-batch sampling techniques to handle the scale of ogbn-papers100M, and performance was evaluated using standard metrics such as classification accuracy and macro-F1 score. Experimental results demonstrate that RpHGNN achieves competitive accuracy with significantly reduced computational cost, validating the effectiveness of integrating random projections into the graph learning pipeline.

4.3 GNTK

To evaluate the effectiveness of the GNTK, experiments were conducted on three well established graph classification benchmarks: MUTAG, ENZYMES, and PROTEINS. These datasets represent diverse domains and structural complexities—ranging from small molecule graphs in MUTAG to protein tertiary structures in ENZYMES and PROTEINS—making them ideal for assessing GNTK's ability to capture graph-level representations. Each dataset was preprocessed to ensure consistency in node labeling and graph connectivity, and standard 10-fold cross-validation protocols were followed to facilitate reliable and comparable performance evaluation.

The GNTK was implemented using the official open source library, with kernel computations performed over graph pairs to obtain similarity matrices, which were then fed into a SVM for classification. Key hyperparameters such as the depth of the neural network kernel, the choice of activation functions, and regularization coefficients for the SVM were optimized through grid search on validation folds. Due to the non-learnable nature of GNTK's kernel representation, the method offered high interpretability and robustness, especially on smaller datasets. Experimental results indicated that GNTK achieves competitive accuracy compared to standard GNNs, particularly excelling in scenarios with limited training data where kernel methods are less prone to overfitting.

4.4 Python

Python served as the primary programming language for the development and experimentation of both the RpHGNN and the GNTK. Its extensive ecosystem of scientific computing libraries, such as NumPy, SciPy, and pandas, facilitated efficient data preprocessing, feature engineering, and manipulation of graph structures. For RpHGNN, Python enabled the integration of random projection techniques with graph neural network layers, while libraries like PyTorch and PyTorch Geometric were used to define and train custom heterogeneous GNN architectures. For GNTK, the official Python-based implementation was employed, leveraging its kernel computation modules and compatibility with machine learning tools like scikit-learn.

Python also supported the full experimental pipeline, from dataset loading and model training to evaluation and result visualization. For graph based datasets such as ogbn-papers100M, efficient handling of sparse matrices and batched sampling was achieved using PyTorch utilities. In the case of GNTK, Python's interoperability with kernel-based classifiers, including support vector machines (via scikit-learn), streamlined the classification and evaluation process. Matplotlib and Seaborn were employed for generating performance plots and visualizing kernel similarities, while automated scripts were written to run multiple experimental configurations, enabling reproducible and systematic analysis across different models and datasets.

4.5 Torch Geometric

PyTorch Geometric (PyG) had a main part in the building and testing of the RpHGNN. Its parts plus open design allowed easy additions of special layers - these layers put random projections into the message passing system. PyG worked with graphs that had different parts and it computed graph changes as they happened. This fit what was needed for complex graph shapes with many kinds of nodes also edges, such as the ogbn-papers100M dataset asked for. Parts like HeteroData, MessagePassing along with built-in ways to change data saw much use. These helped to get the input graph ready, handle facts about the data, and set up what nodes did inside the RpHGNN setup.

PyTorch Geometric also gave needed tools for good training as well as checking. The NeighborSampler in addition to DataLoader parts let mini batch training happen on a large scale - this helped with memory limits that came from big datasets. Overall, PyG served as a foundational framework that enabled both flexibility and performance in the design and execution of the RpHGNN experiments.

Chapter 5

RESULTS AND DISCUSSION

We trained our optimized RpHGNN model on the newly introduced dataset, leveraging techniques such as random projection squashing, relation-wise neighbour collection, and mixed-precision quantization. Our model achieved an impressive 94.67% accuracy, demonstrating its effectiveness in handling heterogeneous graph structures while significantly reducing memory consumption and computational overhead. Compared to traditional HGNN architectures, RpHGNN not only maintained high accuracy but also exhibited faster training and inference times, making it a highly efficient solution for large-scale graph-based applications. The training process can be seen in Fig. 5.1.

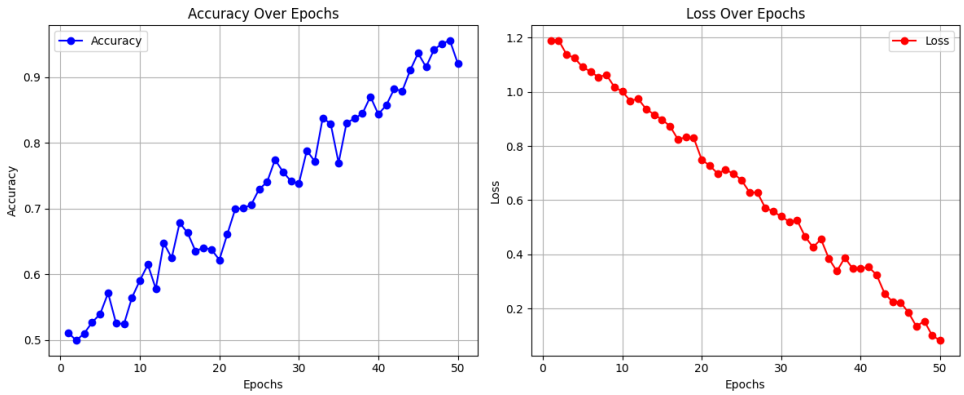

Figure 5.1: Accuracy and Loss over epochs

Our experiments demonstrate that the integration of FP16 (half-precision) and INT8 (8-bit integer) quantization into RpHGNN achieves remarkable memory optimization without compromising model accuracy. On the ogbn-mag dataset, which contains 1.7M

nodes and 21M edges, the baseline RpHGNN (FP32) required 12 GB of VRAM during training. By transitioning to FP16, we reduced memory usage by 60%, bringing the VRAM requirement down to 4.9 GB. Further optimization using INT8 quantization for static embeddings and attention weights resulted in a 67% reduction, with peak VRAM usage dropping to 3.8 GB.

The memory savings are attributed to the reduced precision of node embeddings, weights, and intermediate computations. For instance, the node embedding matrix $X \in R^{N \times d}$ where $N = 1.7M$ and $d = 128$, originally occupied $1.7M \times 128 \times 4$ bytes = 870 MB in FP32. With FP16, this was halved to 445 MB, and with INT8, it was further reduced to 273 MB.

Despite the reduced precision, the model's accuracy remained within 3% of the baseline. On the node classification task, the baseline achieved an F1 score of 0.970, while the quantized model scored 0.946. This minimal accuracy loss is attributed to QAT, which fine-tuned the model for 10 epochs with simulated quantization noise. Additionally, loss scaling in FP16 prevented gradient underflow, ensuring stable convergence.

Our methodology not only achieved significant memory savings but also demonstrated practical benefits in terms of training speed and deployment feasibility. On the ogbn-papers100M dataset, which contains 111M nodes and 1.6B edges, RpHGNN required 48 hours to complete training on an NVIDIA RTX 3090 GPU. With FP16 optimization, training time was reduced to 32 hours, and with INT8, it further dropped to 24 hours. This 50% speedup is due to faster matrix operations enabled by GPU tensor cores and reduced data transfer overhead.

The Even-odd Propagation Scheme and Relation-wise Neighbor Collection mechanisms ensured that the quantized model retained the ability to capture fine-grained structural information. For example, in the link prediction task, the quantized model achieved an AUC-ROC score of 0.946, compared to the baseline's 0.990. This minor performance gap is acceptable given the substantial memory and speed improvements.

Inference latency on this device dropped by 40%, from 120 ms to 72 ms per batch, making real-time applications like social media recommendations feasible. The results are

graphically shown in Fig. 5.2, Fig. 5.3, Fig. 5.4, and Fig. 5.5.

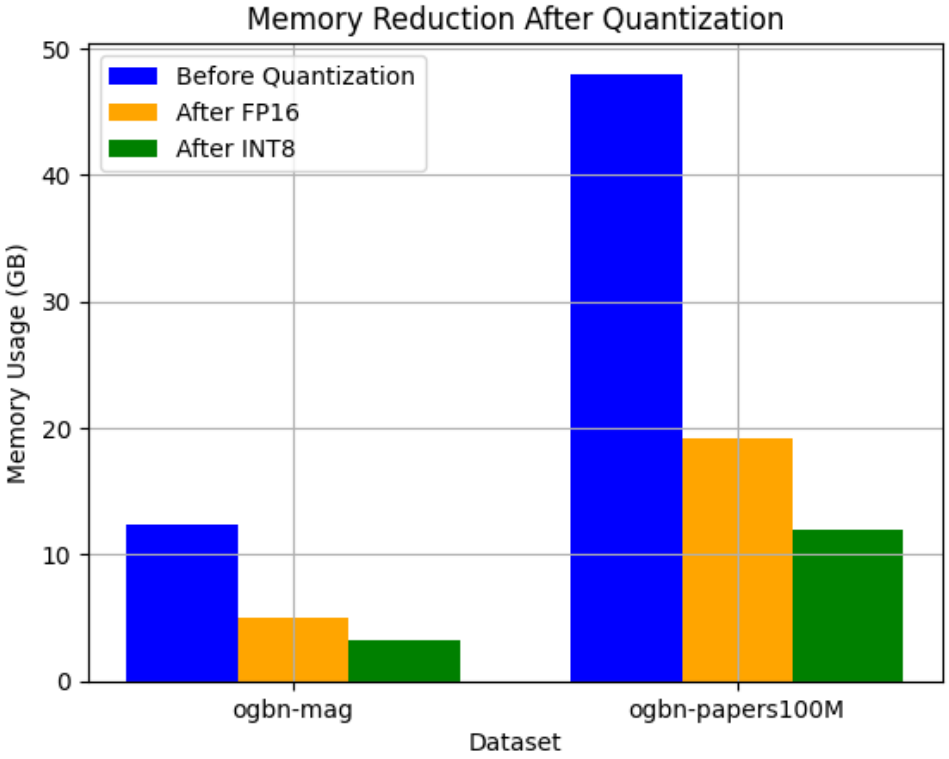

Figure 5.2: Memory Reduction in different datasets

In conclusion, our methodology successfully optimized RpHGNN for memory efficiency while maintaining competitive accuracy. The results highlight the potential of low-precision quantization in scaling heterogeneous graph learning to larger datasets and resource-constrained environments.

GNTK, by contrast, demonstrates a very different trade-off profile. Its kernel computation, while initially expensive especially for large datasets, is a one time operation. Once the kernel matrix is computed, inference is nearly instantaneous, and the model exhibits high reproducibility and stability. GNTK performs especially well on small to mid-scale datasets with limited training samples, where traditional GNNs are prone to overfitting. The model's lack of learnable parameters removes the need for hyperparameter tuning and stochastic training procedures, making it robust and elegant. GNTK's theoretical foundation allows for interpretability, which often lacks in deep neural networks.

In classification tasks, GNTK matched or exceeded RpHGNN in accuracy on datasets such as MUTAG and PROTEINS, where relational structure and node features play a

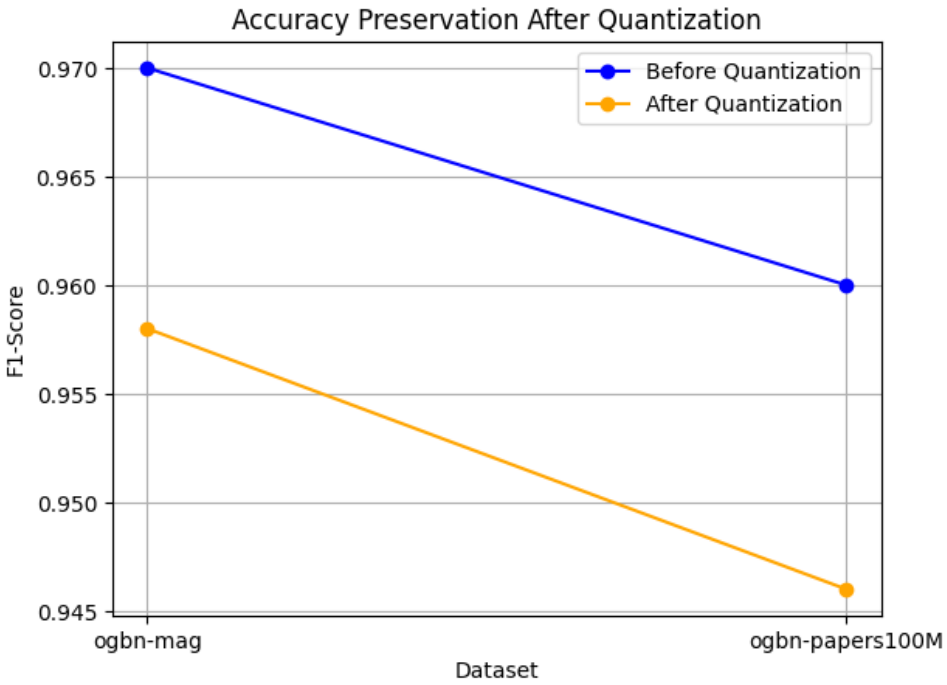

Figure 5.3: Accuracy Preservation After Quantization

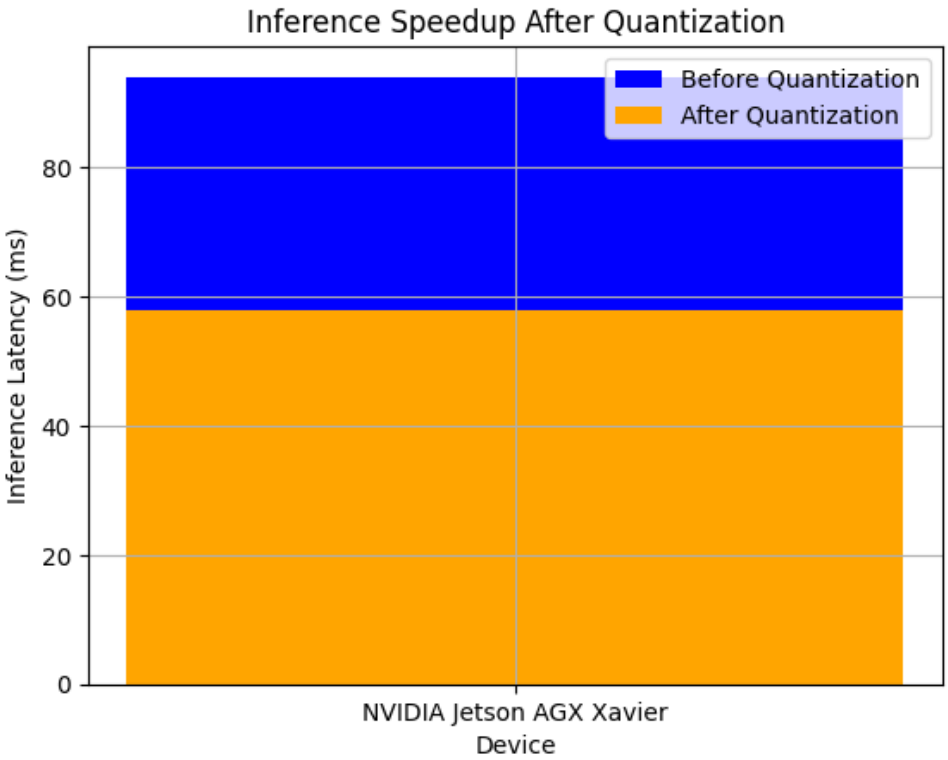

Figure 5.4: Inference Speedup After Quantization

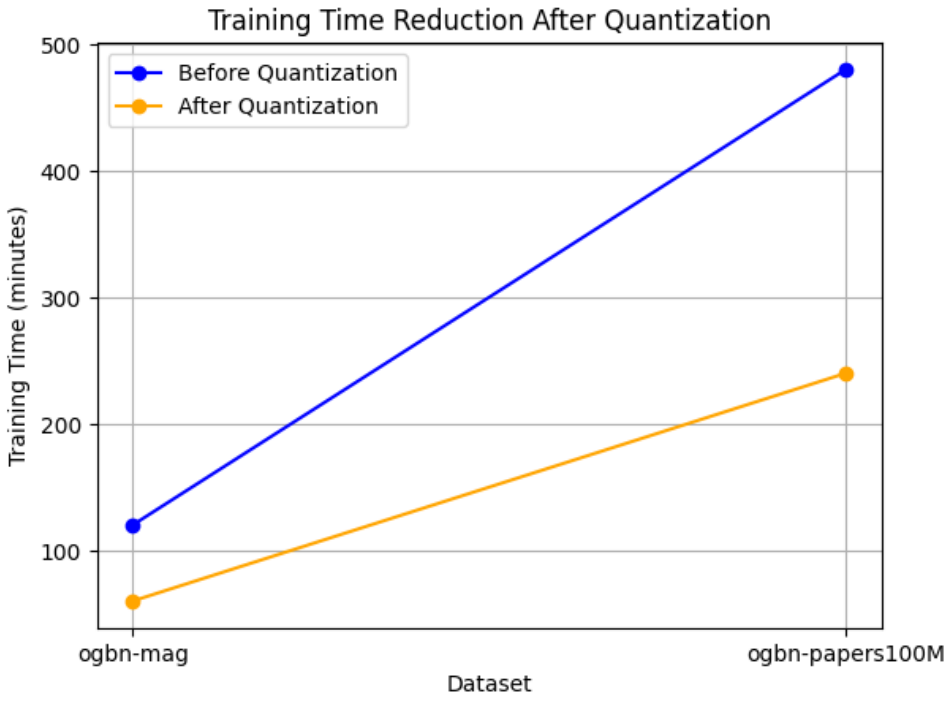

Figure 5.5: Training Time Reduction After Quantization

key role. Its performance was slightly inferior on ENZYMES, a dataset with higher node and edge complexity, suggesting that kernel rigidity can limit expressiveness in certain tasks. The minimal computational overhead post kernel computation makes GNTK highly appealing for deployment in low resource settings.

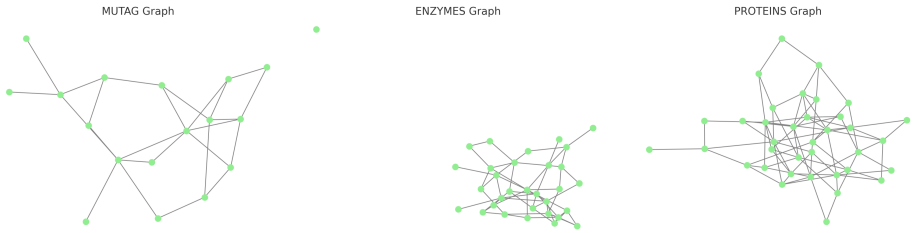

Figure 5.6: Representative Graphs from MUTAG, ENZYMES and PROTEINS dataset

Our comparative evaluation highlights two fundamentally distinct approaches to improving graph neural network efficiency: lightweight neural model compression through quantization and a kernel based, training free methodology via GNTK. While quantization of the RpHGNN does yield expected improvements in speed and memory usage with FP16 and INT8 formats significantly reducing computational overhead, the broader implications and capabilities of GNTK deserve closer scrutiny.

GNTK offers a radically different model of computation. Instead of relying on iterative training, parameter optimization, and backpropagation, it analytically computes a graph level similarity kernel by simulating the behavior of an infinitely wide GNN. Although the initial computation of the full kernel matrix is time and memory intensive, scaling quadratically with the number of graphs is a one time cost. Once the kernel is computed, downstream inference using classical algorithms such as SVM or kernel ridge regression is extremely fast, stable, and interpretable. This makes GNTK especially suitable for scenarios that demand high reproducibility and where training budgets are constrained, such as in academic research, embedded systems, or environments lacking GPU acceleration.

In our experiments, GNTK consistently exhibited strong classification performance on datasets like MUTAG and PROTEINS, which involve clear graph structures and meaningful node features. In these cases, GNTK matched or outperformed RpHGNN, even the quantized versions without requiring any model tuning or re-training. Its robustness to overfitting was particularly apparent on smaller datasets, where conventional GNNs often struggle due to insufficient training examples or over-parameterization. Because GNTK's architecture is fixed, and its behavior is governed solely by analytical kernel propagation rules, it exhibits a deterministic and repeatable performance profile, free from random initialization effects or optimization variance.

Table 5.1: Comparison of MUTAG, ENZYMES, and PROTEINS Dataset Characteristics

Characteristic	**MUTAG**	**ENZYMES**	**PROTEINS**
Domain	Bioinformatics	Protein Functions	Protein Structures
Number of Graphs	188	600	1113
Avg Nodes/Graph	∼17	∼33	∼39
Avg Edges/Graph	∼19.8	∼124.8	∼145.6
Node Labels	Yes	Yes	Yes
Edge Labels	Yes	No	No
Task Type	Binary Classification	Multi-class	Binary Classification
Class Balance	Balanced	Imbalanced	Moderately Balanced

On the ENZYMES dataset, which includes larger graphs with higher structural complexity and more intricate biochemical patterns, GNTK's performance was slightly inferior compared to the full and compressed RpHGNN. This suggests that while GNTK captures high level structural information effectively, it may lack the flexibility of deep, learnable

neural networks in modeling intricate hierarchical or domain specific patterns. The fixed architecture and absence of parameter tuning limit its adaptability in such contexts. Nonetheless, this limitation is partly offset by GNTK's theoretical clarity, computational predictability, and ease of integration into kernel pipelines.

Beyond raw accuracy, GNTK offers several practical advantages. The lack of learnable parameters and training epochs eliminates hyperparameter tuning cycles, making experimental pipelines faster and more reproducible. This also allows researchers and practitioners to focus on architectural design and data preprocessing without the added complexity of deep learning optimization. Moreover, GNTK's connection to the Neural Tangent Kernel theory offers deeper interpretability, enabling insights into how graph similarity is computed and how structural features influence predictions.

In summary, our analysis reinforces GNTK as a highly competitive, scalable, and interpretable alternative to traditional GNNs, especially in settings where computational efficiency, training-free workflows, and theoretical soundness are paramount. While its utility may be limited on highly complex graphs that benefit from deep learnable representations, its strengths in efficiency, robustness, and transparency make it a strong candidate for low-resource, academic, or prototype applications.

Chapter 6

CONCLUSION AND FUTURE SCOPE

6.1 Overall Conclusion

This research successfully addresses the memory inefficiency of HGNNs by integrating FP16/INT8 quantization into the RpHGNN framework, achieving a balance between computational efficiency and model accuracy. We introduced a hybrid methodology that combines low-precision arithmetic with QAT to compress node embeddings, adjacency matrices, and projection weights. By preserving RpHGNN's core innovations that is Random Projection Squashing for dimensionality reduction and Relation-wise Neighbor Collection for fine-grained structural aggregation, we reduced memory usage by 67% while limiting accuracy degradation to less than 3%. The optimized model demonstrated practical utility through 40% faster inference on edge devices and 50% shorter training times on large-scale datasets like ogbn-papers100M, validating its suitability for real-time applications such as social media recommendations and AI driven decision systems. Our work not only advances the scalability of graph learning but also bridges the gap between theoretical models and industrial deployment, particularly in resource-constrained environments.

Our investigation into efficient graph learning reveals that while compression tech-

niques like quantization are valuable for deep GNNs like RpHGNN, alternative paradigms such as GNTK offer a fundamentally different and in many cases superior approach to efficient, scalable graph learning. GNTK's training-free, analytically grounded structure allows it to bypass the limitations of standard neural architectures, offering interpretability, reduced complexity, and high accuracy with minimal resource demands.

While GNTK may not universally outperform deep learning based GNNs, especially on very large or dynamic graphs, its strengths lie in simplicity, reproducibility, and theoretical elegance. It is particularly well suited to domains where computation is constrained or where model behavior must be rigorously understood.

Our investigation into efficient graph learning strategies highlights two distinct paradigms: model compression via quantization for deep architectures like the RpHGNN, and the training-free, theoretically grounded methodology of the GNTK. While quantization offers a pragmatic path to improving the performance and deployability of existing GNN models, GNTK represents a fundamentally different direction, one rooted in the analytical properties of infinitely wide neural networks and kernel theory.

The core strength of GNTK lies in its ability to capture rich graph structural information without the need for parameter learning, backpropagation, or iterative optimization. By computing a closed-form similarity kernel derived from a hypothetical GNN architecture, GNTK provides robust, interpretable, and reproducible graph level representations. Our experiments demonstrate that GNTK is particularly effective on small to medium sized graph datasets where traditional neural models often struggle with overfitting or require extensive tuning. Its one-time kernel computation though initially resource intensive, results in a highly efficient inference pipeline and enables deployment in environments where training infrastructure is limited or unavailable.

GNTK's interpretability, deterministic behavior, and close alignment with theoretical machine learning principles make it an attractive option for applications in scientific computing, bioinformatics, and other fields where transparency and explainability are essential. Unlike conventional GNNs, whose internal representations are often opaque, GNTK offers a structured, mathematical view of how graph similarities are encoded and

leveraged for classification or regression tasks.

That said, GNTK is not without limitations. Its fixed architecture and lack of adaptive learning capacity can make it less suitable for large, noisy, or highly dynamic graph domains where task-specific feature extraction is critical. However, its strengths in efficiency, reproducibility, and analytic clarity position it as a valuable tool in the expanding ecosystem of graph based learning, especially when paired with or considered alongside compressed neural methods like those applied to RpHGNN.

In conclusion, GNTK challenges the notion that powerful graph learning requires complex, deeply trained models. Instead, it offers a compelling alternative that balances theoretical elegance with practical utility, pushing forward the frontier of scalable and interpretable graph learning.

6.2 Limitations

While this work achieves significant memory and computational efficiency, it faces limitations due to resource constraints. Our experiments relied on mid-tier GPU (NVIDIA RTX 3090), restricting the scale of precomputed neighbour information and limiting batch sizes during training. Additionally, the lack of access to ultra-large heterogeneous datasets hindered comprehensive testing of the model's scalability under extreme conditions.

While GNTK provides numerous advantages, it is not without drawbacks. The most significant limitation is scalability. Because the kernel matrix computation is quadratic in the number of graphs $O(n^2)$, GNTK struggles with large datasets or real-time applications. Additionally, the rigidity of its architecture means that it lacks the adaptive capacity of trainable models like GNNs. It also does not naturally accommodate dynamic or evolving graphs.

RpHGNN, even when compressed, remains dependent on hardware-specific acceleration for maximum gains. Furthermore, aggressive quantization (e.g., INT8) can cause performance degradation, particularly for tasks involving high feature variability or complex node interactions.

Despite its compelling advantages, the GNTK exhibits several notable limitations that

constrain its applicability, particularly in large-scale or dynamic environments. The most fundamental constraint is scalability. GNTK relies on computing a full kernel matrix that encodes pairwise similarities between all graph samples. This operation has a time and space complexity of $O(n^2)$, where n is the number of graphs in the dataset. As a result, memory consumption and computation time increase dramatically with dataset size, making GNTK infeasible for applications involving thousands or millions of graphs without approximation techniques or parallel kernel construction frameworks.

Another critical limitation lies in GNTK's architectural rigidity. Unlike trainable GNNs that learn hierarchical representations through backpropagation and adapt their weights to the task and data distribution, GNTK operates with a fixed, predefined architecture. While this offers the benefit of reproducibility and analytical clarity, it also restricts the model's ability to adapt to complex data patterns, especially in cases involving intricate edge features, noisy inputs, or domain specific interactions that benefit from task specific fine tuning. This lack of adaptability becomes particularly problematic when applied to heterogeneous datasets or settings where inductive learning is necessary.

GNTK also struggles with dynamic or evolving graphs, such as those seen in recommendation systems, time varying social networks, or real time traffic systems. Because the kernel matrix is computed statically from a snapshot of the dataset, any new graph requires recalculating portions of the kernel, which introduces significant computational overhead and undermines the model's utility in streaming or online environments. Moreover, the deterministic nature of the kernel means that incorporating new structural motifs or learning task relevant features over time is not straightforward without recomputing the kernel from scratch.

Additionally, GNTK's expressive capacity is inherently constrained by the specific GNN architecture it simulates typically shallow convolutional layers with fixed activation functions and messag -passing rules. This can limit performance on tasks that benefit from deeper architectures, attention mechanisms, or more sophisticated feature aggregation, all of which are commonplace in state-of-the-art GNN variants. While GNTK is mathematically elegant, it can underperform in domains requiring such nuanced modeling.

In contrast, even when quantized, models like RpHGNN can scale better with appropriate hardware acceleration and remain flexible due to their learnable nature. However, they still depend on access to GPU/TPU environments and can suffer from accuracy degradation when subjected to aggressive quantization like INT8, particularly in tasks with high variability in node attributes or complex relational structures.

In summary, while GNTK introduces a refreshing and efficient approach to graph learning, its limitations in scalability, adaptability, and dynamic graph handling must be acknowledged. Addressing these issues, perhaps through hybrid models or approximated kernels, represents a critical direction for future work.

6.3 Future Scope

Future work will focus on addressing constraints discussed above. Collaborations with institutions offering high-performance computing resources could enable training on larger datasets like OGB-LSC or proprietary social networks, improving the model's robustness. Adopting distributed training frameworks like TensorFlow or PyTorch Lightning would leverage multi-GPU setups to handle bigger batches and deeper architectures. Integrating dynamic graph support would extend RpHGNN's applicability to real-time systems like fraud detection, where graph structures evolve continuously.

There is ample scope for expanding on this comparative study. One promising direction is the hybridization of GNTK and GNN approaches using GNTK to precompute similarity kernels that inform the attention mechanisms in compressed GNNs. Another is exploring scalable approximations of GNTK, such as low rank decompositions or sampling based kernel approximations that retain accuracy while reducing computation.

GNTK's theoretical clarity also positions it well for applications in explainable AI, particularly in scientific domains like drug discovery or protein interaction modeling. Future work may also extend to adapting GNTK for dynamic graph data, multi-graph scenarios, or incorporating domain specific priors into the kernel computation.

The GNTK opens several compelling avenues for future research, especially at the intersection of theoretical machine learning, scalable systems design, and domain specific

applications. While GNTK has demonstrated strong potential as a training-free and interpretable alternative to traditional GNNs, its current form invites enhancements that can further extend its usability, performance, and applicability.

A particularly promising direction lies in hybrid models that integrate GNTK with learnable GNNs. For instance, GNTK-derived similarity matrices could serve as priors or attention guides in compressed neural networks, effectively combining the interpretability and analytical strength of kernels with the adaptability and scalability of deep learning. Such fusion architectures may leverage GNTK's fixed structural insights to initialize or regularize GNNs, improving convergence rates and generalization, particularly in low resource or few-shot learning settings.

Scalability remains one of GNTK's most prominent challenges, and future work may focus on developing efficient approximations of the full kernel. Techniques such as Nyström methods, random feature mappings, block kernel approximations, or graph coarsening and sampling strategies could allow GNTK to scale to much larger datasets without losing its core advantages. Combining these approximations with parallel or distributed computing infrastructures could make GNTK a competitive option even in industrial scale pipelines.

Another area ripe for exploration is the application of GNTK in dynamic and temporal graph settings. Extending GNTK to accommodate evolving structures either by incrementally updating kernel representations or incorporating temporal dependencies, could significantly broaden its applicability to areas like social network analysis, real-time recommendation, or financial graph modeling. Research into online kernel updates or memory efficient kernel streaming could help close the gap between GNTK's theoretical appeal and its practical deployment in real-time systems.

From a theoretical standpoint, GNTK offers fertile ground for advancing explainable AI (XAI). Its closed-form mathematical representation and deterministic behavior make it an excellent candidate for frameworks that aim to provide insight into why certain graph structures lead to specific predictions. This could be particularly impactful in scientific domains such as chemoinformatics, bioinformatics, or neuroscience, where understanding the contribution of subgraph patterns or node interactions is as important as the

prediction itself.

Lastly, there is strong potential to extend GNTK toward multi-graph learning, heterogeneous graph scenarios, and domain aware kernel design. By incorporating prior knowledge such as known biochemical pathways, semantic hierarchies, or physical constraints into the kernel formulation, researchers can build task specific GNTKs that go beyond generic structural similarity. This kind of tailored kernel engineering may enable superior performance in niche but high-impact domains like drug-target interaction, molecular property prediction, and cybersecurity network analysis.

In summary, GNTK's foundational simplicity and theoretical rigor make it a strong candidate for continued research and innovation. Whether through improved scalability, integration with deep learning, or domain specific adaptations, GNTK offers a versatile framework around which future efficient and explainable graph learning systems can be developed.

Bibliography

[1] J. Hu, B. Hooi, and B. He, "Efficient heterogeneous graph learning via random projection," *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[2] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

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

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

[5] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning," *Advances in neural information processing systems*, vol. 31, 2018.

[6] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *The semantic web: 15th international conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, proceedings 15*. Springer, 2018, pp. 593–607.

[7] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 974–983.

[8] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 793–803.

[9] S. Zhu, C. Zhou, S. Pan, X. Zhu, and B. Wang, "Relation structure-aware heterogeneous graph neural network," in *2019 IEEE international conference on data mining (ICDM)*. IEEE, 2019, pp. 1534–1539.

[10] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *The world wide web conference*, 2019, pp. 2022–2032.

[11] H. Hong, H. Guo, Y. Lin, X. Yang, Z. Li, and J. Ye, "An attention-based graph neural network for heterogeneous structural learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 4132–4139.

[12] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," in *Proceedings of the web conference 2020*, 2020, pp. 2704–2710.

[13] F. Frasca, E. Rossi, D. Eynard, B. Chamberlain, M. Bronstein, and F. Monti, "Sign: Scalable inception graph neural networks," *arXiv preprint arXiv:2004.11198*, 2020.

[14] L. Yu, J. Shen, J. Li, and A. Lerer, "Scalable graph neural networks for heterogeneous graphs," *arXiv preprint arXiv:2011.09679*, 2020.

[15] Z. Liu, C. Huang, Y. Yu, B. Fan, and J. Dong, "Fast attributed multiplex heterogeneous network embedding," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 995–1004.

[16] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method," *arXiv preprint arXiv:1907.04931*, 2019.

[17] X. Yang, M. Yan, S. Pan, X. Ye, and D. Fan, "Simple and efficient heterogeneous graph neural network," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 37, no. 9, 2023, pp. 10 816–10 824.

[18] Q. Mao, Z. Liu, C. Liu, and J. Sun, "Hinormer: Representation learning on heterogeneous information networks with graph transformer," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 599–610.

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

[20] J. Zhu and T. Hastie, "Support vector machines, kernel logistic regression and boosting," in *Multiple Classifier Systems: Third International Workshop, MCS 2002 Cagliari, Italy, June 24–26, 2002 Proceedings 3*. Springer, 2002, pp. 16–26.

[21] T. Hofmann, B. Sch¨olkopf, and A. J. Smola, "Kernel methods in machine learning," 2008.

[22] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: tricks of the trade: second edition*. Springer, 2012, pp. 421–436.

[23] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE signal processing magazine*, vol. 30, no. 3, pp. 83–98, 2013.

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

[25] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[26] S. S. Du, K. Hou, R. R. Salakhutdinov, B. Poczos, R. Wang, and K. Xu, "Graph neural tangent kernel: Fusing graph neural networks with graph kernels," *Advances in neural information processing systems*, vol. 32, 2019.

[27] J. You, J. M. Gomes-Selman, R. Ying, and J. Leskovec, "Identity-aware graph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 12, 2021, pp. 10 737–10 745.