Data Engineering and Applications

Research Paper

Distributed Graph Neural Networks for Large-scale Social Network Data Analysis

Arkadiusz Sadowski[1, *]

[1] Faculty of Computer Science, Opole University of Technology, Opole, 45-271, Poland
*Corresponding author: arkdiusz.s@po.edu.pl

Abstract. In light of the expansion in the scale and complexity of social network graph data, Distributed Graph Neural Networks (DGNNs) have been introduced as a new direction. To meet the demands of advanced systems, this paper proposes a new distributed graph neural network (DGNN) model that can effectively handle the diversity and dynamics of social graphs. This paper constructs an adaptive structure that can balance computational cost, communication expenses, and prediction accuracy in a distributed system. The system supports synchronous and asynchronous parallel execution, implementing its method through multi-level message passing and adaptive neighborhood sampling. The three public social network datasets used for the experiments are Friendster, Reddit, and Twitter, each containing millions of nodes and edges. According to the above experiments, the node classification accuracy of DGNN on all datasets is 0.80-0.88, with an F1 score exceeding 0.84. Superior to many other benchmark models. The scalability is significantly demonstrated in terms of reducing the training time from 500 seconds to 45 seconds per epoch and achieving over 85% parallel efficiency on a 16-node cluster. Robustness tests indicate that the accuracy of the graph significantly decreases in the presence of noise and missing edges. Research shows that the DGNN model excels in predicting resource efficiency and performance, demonstrating good generalization ability across various network environments. This paper will introduce a distributed graph learning system that can be used to analyze social networks.

Keywords: *Graph Neural Networks, Distributed Computing, Social Network Analysis, Large-scale Data, Resource Efficiency, Parallel Processing, Scalability, Robustness*

Received on 5 September 2024, Accepted on 04 January 2025, Published on 29 January 2025

Introduction

In recent years, social networks have developed into a vast and active platform, connecting various online and offline activities. Due to the increase in social data, including connections between users and the way information spreads on social platforms, we can obtain more information about society and its various changes from this data [1]. Traditional machine learning algorithms are usually not suitable for heterogeneous topologies, non-Euclidean data spaces, and dynamic network statistics [2], because the large scale and inherent disorder of such massive graph-structured data make them difficult to capture and model. With the expansion of social graphs, many studies have begun to seek new methods to improve efficiency and handle large amounts of data [3]. The formation of Graph Neural Networks (GNNs) provides a new method for end-to-end learning of structured data. It is also more interpretable than shallow embedding methods or traditional statistical models [4]. Some progress has been made, but most GNN techniques are still single-machine in nature and cannot be used for large-scale real-world networks containing millions or even billions of nodes and edges [5].

At both the system and algorithm levels, scalable social network analysis faces many challenges. In terms of algorithms, large-scale heterogeneous graphs are often very chaotic and have a high level of noise. A large number of missing links make stable and accurate representation learning difficult [6]. Traditional graph

partitioning and sampling methods may lead to significant biases or reduce neighborhood dependencies, limiting the expressive power of learned features [7]. These methods have advantages in terms of computation. In the system, distributed GNN training leads to synchronization bottlenecks, data and workload imbalances, and increased communication overhead. Both computational efficiency and resource utilization are affected [8]. Due to the tendency of synchronous models to experience global synchronization penalties and lagging nodes, asynchronous designs must consider consistency and convergence stability [9]. Recently, new system architectures, adaptive sampling methods, and rigorous theoretical analyzes of convergence and generalization have been proposed to address the shortcomings in efficient graph storage and large-scale GNN training [10]. Despite these advancements, there is still no universal framework to accommodate large-scale computation and communication strategies [11]. Few studies comprehensively examine the relationship between model robustness, embedding quality, resource efficiency, and distributed GNN deployment [12].

In this paper, we introduce a new distributed graph neural network (DGNN) framework for high-performance, scalable, and stable social network analysis. By using a hierarchical message-passing architecture and adaptive sampling, we can address the issues of large-scale heterogeneous networks that arise in predictions. Multiple parallelizations can flexibly achieve synchronous and asynchronous updates, while reducing hardware overhead and maintaining the stability and convergence of the new system in a distributed environment. Several empirical studies have used various real-world social datasets. These datasets provide our method with good predictive and generalization capabilities, while also achieving significant improvements in system throughput, scalability, and representation learning quality. This paper proposes a theoretical framework and practical solutions for the application of extended distributed graph learning on large-scale social data.

Related Work

Advancements in Graph Neural Networks

Graph Neural Networks (GNNs) have recently improved the ability to learn data representations of complex network structures through development. Early research on iteratively aggregating neighborhood information laid the foundation, adding context and dependencies to the model that go beyond the local graph structure [13]. The inductive GraphSAGE model and Graph Convolutional Networks (GCN) enhance the robustness of the model because they extend the concept of convolution to irregular graphs and allow learning from previously unseen nodes [14]. Graph neural networks have recently made significant progress in various fields such as node classification and link prediction. They are very flexible and are also used for the classification of entire graphs. Graph Attention Networks (GAT) are a new technology that dynamically learns the influence weights of neighboring nodes on the GNN, while reducing the impact of noise in heterogeneous graph structures [15]. Moreover, the goal of the research is to explicitly incorporate domain knowledge (such as patterns or hierarchical substructures) directly into the model architecture to enhance the expressive power and generalization ability of GNNs. Cross-layer normalization, scalable neighborhood sampling, cross-layer normalization, and handling "over-smoothing" have become key issues in modeling large-scale social networks. These improvements are applicable to large-scale real-world datasets [16]. However, the current drawbacks include the limitations of time and dynamic graph models, the high computational cost of higher-order central nodes, and the limited deployment in resource-constrained environments.

Distributed Large-scale Graph Processing Techniques

It is necessary to develop new distributed computing and algorithms to meet the expansion of social networks. The initial distributed graph processing systems were vertex- or subgraph-oriented, with large graphs being processed and scaled by worker nodes [17]. Pregel and GraphX are examples of solutions that can be used in practice; they are user-friendly and have low communication overhead. Advanced platforms like DistDGL and DGL develop improved graph partitioning algorithms to reduce communication costs and support hybrid execution models, thereby enhancing GNN training efficiency on billion-scale graphs. These algorithms have emerged with the increase in dataset sizes and the demand for high-performance computing [18]. Achieving excellent performance on network-scale data, the aforementioned framework leverages hardware parallelism and optimizes the balance between computation and data transfer. The widespread use of neighborhood sampling and mini-batch training reduces the need for full graph traversal, making memory-efficient distributed

learning possible. With the aforementioned advancements, new system-level issues have also emerged. These issues include handling data staleness, reducing computational latency, and effective load balancing [19]. These issues are more severe in asynchronous processing. By jointly optimizing the model architecture and hardware utilization, the system-aware GNN framework points the way forward. Achieve the expected improvements in throughput, energy efficiency, and scalability in practical applications [20].

Theoretical Insights of DGNN Frameworks

In order to explore the development trajectory and support measures of the Distributed Graph Neural Network (DGNN) framework in large-scale social network analysis, a solid theoretical foundation needs to be established. The convergence and stability of message passing have been carefully studied in heterogeneous and partitioned systems. Now we know that the choice of synchronization and the underlying graph structure can affect the training dynamics and the final model quality. Theoretically, under the condition of appropriately handling delays and stale updates in the system, distributed GNNs can approximate centralized methods without significantly losing expressive power. This depends on some operational assumptions. To support robust estimation in large-scale distributed systems, the development of generalization theory has recently formally defined how sampling strategies, graph partitioning, and neighborhood randomness affect statistical guaranties. Currently, consistency, communication, and robustness to noise or incomplete data are all key research focuses. Spectral regularization techniques and adaptive synchronization and self-healing aggregation mechanisms have recently been improved to enhance the robustness of DGNN in heterogeneous and distributed environments.

Novel Distributed GNN Architecture

Layered Model Structure and Message Passing

The Distributed Graph Neural Network (DGNN) framework uses a stack of computational layers to train large-scale social networks. Add layers to the group structure and implement localized feature modifications to achieve more generalized multi-scale node embeddings in highly irregular networks. In order to capture local and global patterns, the core of DGNN is a recursive aggregation mechanism that iteratively integrates neighborhood information from distributed graph partitions through multiple steps.

Formally, nodes in the graph $G = (V, E)$ are associated with input features $H^{(0)} \in \mathbb{R}^{|V| \times d_0}$. At each layer l, node states are synchronously updated by aggregating their neighborhood and self-information, such that:

$$h_v^{(l+1)} = \sigma\left(W^{(l)} \cdot AGG\left(\left\{h_u^{(l)} : u \in \mathcal{N}(v) \cup \{v\}\right\}\right)\right) \quad \text{Eq.(1)}$$

where $h_u^{(l)}$ is the feature vector for node u at layer $l, W^{(l)}$ is a learnable matrix, $\mathcal{N}(v)$ denotes the neighborhood, $\mathrm{AGG}(\cdot)$ is the aggregator function, and σ denotes the activation.

The whole partitioned update in a distributed environment can be expressed as:

$$H_p^{(l+1)} = \sigma\left(W^{(l)} \cdot AGG\left(H_p^{(l)}, M_{B(p)}^{(l)}\right)\right) \quad \text{Eq.(2)}$$

where $H_p^{(l)}$ refers to node features held by partition p, and $M_{B(p)}^{(l)}$ contains messages for the partition's boundary nodes, received from neighboring partitions.

The purpose of these two layers is to ensure efficient communication between boundary nodes and to quickly update local nodes. High-throughput systems typically compress and aggregate messages across partition boundaries before transmission to reduce network load. In addition, the framework supports hierarchical or cluster aggregation to achieve semantic-aware representation learning for social communities and multi-scale structures.

DropEdge and batch or layer normalization are used for joint regularization to prevent overfitting and improve the stability of deep networks in real-world data with noise. Using pipelined execution, the forward and backward propagation steps are overlapped to reduce the working time of each employe.

Figure 1 shows the computational and logical flow of the system. In addition, the division of responsibilities between local updates, boundary message passing, and hierarchical aggregation is also clearly visible.

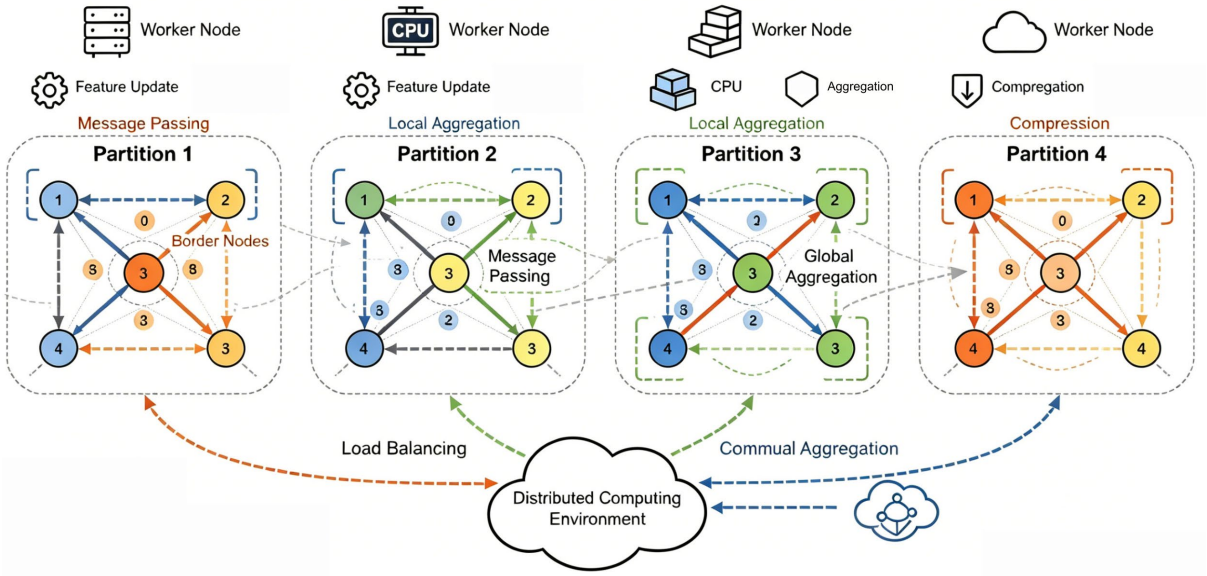

Figure 1. Distributed GNN System Architecture and Message Passing Workflow

Parallelism and Synchronous/Asynchronous Design

The scaling of large-scale graphs requires communication scheduling and parallel processing. By using the DGNN framework, the methods for inter-partition distributed operations and intra-partition parallel processing can be improved. Multithreading routines or GPU acceleration can be used for feature updates at the local level of each node to maximize hardware utilization and reduce the latency of independent aggregation tasks.

Each employe is responsible for a part of the graph, organizing data transfers between partitions. First, the communication costs and synchronization overhead should be compared with the benefits of global consistency. In synchronous mode, all worker nodes progress simultaneously. A round will only be initiated after the necessary boundary message exchanges are completed. It is suitable for a stable and evenly distributed workload cluster.

Mathematically, the synchronous update for partition p in a distributed DGNN for layer $l+1$ is given by:

$$H_p^{(l+1)} = \sigma\left(W^{(l)} \cdot \left[H_p^{(l)} \| \sum_{q \in \mathcal{N}(p)} M_{q \to p}^{(l)}\right]\right) \quad \text{Eq.(3)}$$

where || denotes concatenation and $M_{q \to p}^{(l)}$ represents messages from neighboring partition q to p.

Asynchronous parallel technology allows each worker node to immediately update its local state upon receiving a new message, without having to wait for other worker nodes. This method is suitable for larger, sparse, or unbalanced graphs that do not require very low convergence randomness. Although asynchronous updates may be somewhat outdated, the bounded delays in the system message queue and the partial consistency protocols can still converge effectively.

To formalize staleness in asynchronous parallelism, we consider the updates of node v in partition p as:

$$h_v^{(l+1)} = \sigma\left(W^{(l)} \cdot AGG\left(\left\{h_u^{(l)} : u \in \mathcal{N}(v), latest\right\}\right)\right) \quad \text{Eq.(4)}$$

"Latest" refers to the most recent state of the neighbor, which may be inconsistent across the entire environment.

In order to achieve good scalability in large-scale graphs and multiple partitions, dynamic scheduling strategies can be dynamically selected based on the balance between workload and network congestion, while also dynamically choosing between synchronous and asynchronous steps.

Scalability Through Adaptive Sampling

During the message-passing phase, the framework uses adaptive sampling to address the computational and memory issues of large-scale distributed GNNs. Dynamically adjust the neighborhood aggregation range to balance accuracy, resource usage, and convergence speed.

Let $S_v^{(l)} \subseteq \mathcal{N}(v)$ denote the sampled neighbors for node v at layer l. The node update with adaptive sampling is then:

$$h_v^{(l+1)} = \sigma\left(W^{(l)} \cdot AGG\left(\left\{h_u^{(l)} : u \in S_v^{(l)} \cup \{v\}\right\}\right)\right) \quad \text{Eq.(5)}$$

Here, the size and composition of $S_v^{(l)}$ are determined by graph structure and runtime constraints, ensuring that aggregation cost does not explode on high-degree nodes.

Further enhancing efficiency, sampling probabilities can be made adaptive:

$$P\left(u \in S_v^{(l)}\right) = \frac{\alpha_u^{(l)}}{\sum_{w \in \mathcal{N}(v)} \alpha_w^{(l)}} \quad \text{Eq.(6)}$$

where $\alpha_u^{(l)}$ is a learnable or data-driven relevance score (e.g., based on attention weights or edge features), focusing computation on the most informative links.

Finally, in order to observe the effect of sampling on the convergence of learning, an empirical convergence constraint for the training process is set as follows:

$$\mathbb{E}\left[\left\|\Delta H_v^{(l+1)}\right\|\right] \leq \varepsilon \quad \text{Eq.(7)}$$

where $\Delta H_v^{(l+1)}$ is the update increment for node v at the next layer, and ε is a target error tolerance. Therefore, this rule can be flexibly applied to increase or decrease the sample size to improve accuracy and speed.

Under the constraint of reducing computational load, the aforementioned adaptive sampling strategy can efficiently handle large-scale graphs while maintaining the structural representativeness and prediction accuracy of DGNN.

Experimental Setup and Implementation Details

Dataset Description and Preprocessing

To validate the scalability and effectiveness of our Distributed GNN (DGNN) architecture, we selected three large-scale real-world social network graphs (Friendster, Reddit, and Twitter) as datasets. Due to their inherent multi-relational structure, extensive degree distribution, and high node heterogeneity, these datasets pose memory-constrained and communication-intensive challenges for distributed learning systems.

In order to ensure network connectivity and eliminate noisy isolated subgraphs, the preprocessing stage processed all datasets, retaining only the largest strongly connected components. Node features are created from available user attributes, text embeddings, and behavioral statistics, with missing attribute values filled with zeros or category means to ensure input consistency. For multi-type edges, each relationship is encoded as an independent adjacency matrix $A^{(r)}$, where r indexes the relationship type, and supports multi-relation propagation. Edge information is filtered to retain only the primary relationships that need to be learned.

$$A = \sum_{r=1}^{R} \alpha_r A^{(r)} \quad \text{Eq.(8)}$$

where R is the total number of relation types and α_r denotes learnable or normalized weights per relation.

A good partitioning should be used for scalable distributed GNN training. Metis is used to partition each graph; it employs balanced vertex and edge-cutting strategies to achieve fewer inter-partition edges and an even distribution of workloads. The reasons for partitioning are as follows:

$$\min_P \sum_{(u,v) \in E} \mathbb{I}[partition(u) \neq partition(v)] \quad \text{Eq.(9)}$$

where $\mathbb{I}[\cdot]$ is the indicator function, and the aim is to minimize the number of cross-partition edges. Dynamically match the number of partitions of available compute nodes and maintain a balance between computational load and memory. In order to achieve efficient parallel loading, the final preprocessed graph data and attributes are serialized and sharded. Figure 2 shows the experimental process, including system setup, distributed training and evaluation, as well as the entire data acquisition and partitioning. This will help ensure that the experiments are repeatable and reproducible.

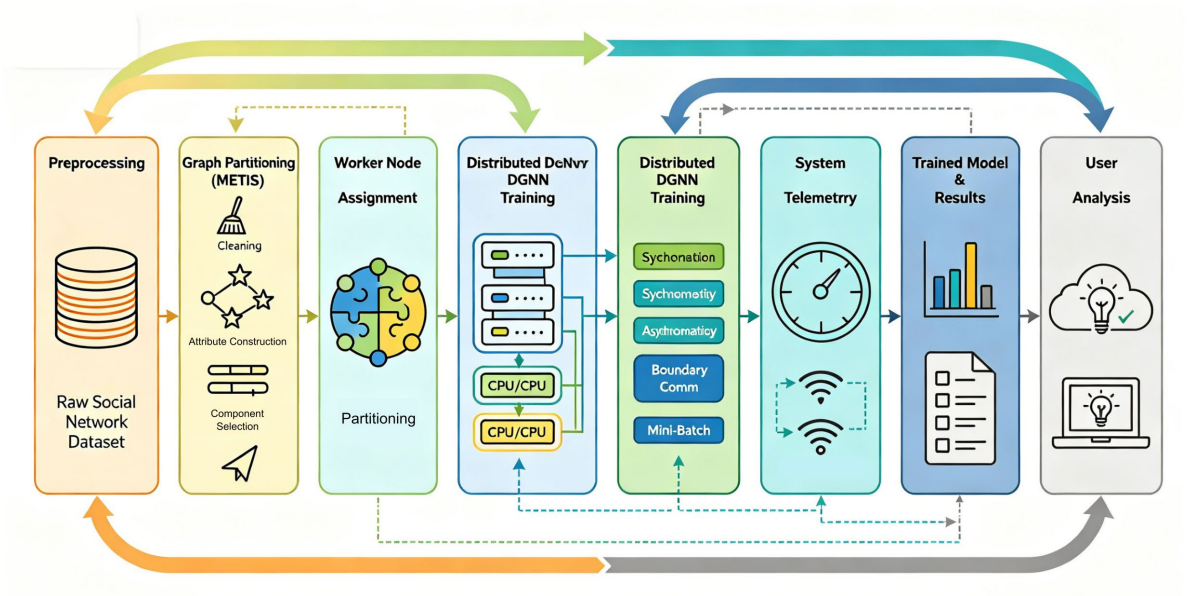

Figure 2. End-to-End Experimental Pipeline

System Environment and Distributed Deployment

All experiments were conducted on a high-performance cluster. The cluster consists of sixteen worker nodes, each equipped with four NVIDIA A100 80GB GPUs, 512 GB DDR4 memory, and a dual-socket Intel Xeon Platinum CPU with 52 physical cores. 100Gbps Infiniband EDR is used for low-latency communication between nodes, and NVLink supports GPU connections within nodes. Ubuntu 22.04 LTS is the operating system I chose.

For basic tensors and the training engine, PyTorch 2.1 was chosen as the software stack, and NCCL and Horovod were added to achieve multi-GPU parallelism. To provide GNN primitives and graph partitioning tools, DGL v1.1 was chosen. Kubernetes is used for container orchestration, gRPC and MPI are used for message passing and synchronization, thereby achieving dynamic scaling and elastic distributed resource allocation.

Each graph partition is assigned to an independent worker node, and then the workload is further distributed to the GPU through model parallelism or mini-batch processing. Using buffered transfers and asynchronous pipelines, communication between boundary nodes reduces idle time. In addition, the partition-aware caching mechanism and memory mapping reduce I/O bottlenecks. Global optimization includes all hyperparameters of Bayesian optimization, such as batch size, learning rate, and the neighbor sampling width $S^{(l)}$ for each layer. The iterative paradigm is the standard for distributed model training:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}_{batch}(\theta_t) \quad \text{Eq.(10)}$$

where η denotes the learning rate and $\mathcal{L}_{\text{batch}}$ represents the batch-wise supervised loss evaluated on distributed partitions.

In the synchronous phase, the parameters for message communication between employes are the same. When asynchronous mode is enabled, the latest parameters in the local cache are used to compute the gradient:

$$g_v^{(l+1)} = \sigma\left(W^{(l)} \cdot AGG\left(\left\{h_u^{(l)} : u \in \mathcal{N}(v), t_u \leq t\right\}\right)\right) \quad \text{Eq.(11)}$$

where t_u is the latest timestamp for neighbor u 's state received at worker v for layer l.

By using synthetic network delays and controlled node failures to test the system's robustness. Prometheus collects end-to-end telemetry data in real-time to analyze resource utilization, communication overhead, and fault traces.

Baselines and Evaluation Metrics

The benchmark tests of DGNN used a wide range of benchmark algorithms. GraphSAGE uses parallel neighborhood sampling to achieve large-scale distributed operations, while the original GCN only uses a single node. Cluster-GCN uses graph mini-batch clustering, which has good locality and memory efficiency. DistDGL is a general-purpose industrial system used to evaluate overall performance. SAGA is an outdated asynchronous method for distributed GNNs that balances consistency and speed.

Performance was quantitatively assessed along four axes. Scalability was characterized by throughput (X), convergence duration (T), and parallel efficiency (E) as:

$$E = \frac{X_{observed}}{N \cdot X_{single}} \quad \text{Eq.(12)}$$

where N is the number of nodes, X_{single} is single-node throughput, and X_{observed} is multi-node performance.

All methods use standardized data partitions to evaluate predictive performance, using micro/macro F1, ROC-AUC, and accuracy for assessment. The loss convergence path is shown below, indicating that the model remains stable during training.

$$\mathcal{L}_{val}(t) = \mathbb{E}_{(x,y)\in\mathcal{D}_{val}} \ell\left(f_{\theta_t}(x), y\right) \quad \text{Eq.(13)}$$

where $\mathcal{L}_{\text{val}}\,(t)$ is the validation loss at epoch t.

System efficiency was quantified as

$$Utilization = \frac{Active\ time}{Wall\text{-}clock\ time} \quad \text{Eq.(14)}$$

Regularly record the usage of devices, memory, and bandwidth. Robustness refers to the decrease in accuracy and recovery speed after node and network failures. To ensure that DGNN performs well under various conditions, each experiment used early stopping, three repeated runs, and paired t-tests for statistical significance testing.

Performance Evaluation and Analysis

Predictive Performance, Robustness, and Scalability

This section provides a detailed evaluation of the proposed Distributed GNN framework (DGNN) in terms of prediction accuracy, noise resistance, and scalability. It also conducted rigorous benchmarking on widely used distributed and non-distributed benchmarks. Using the latest techniques introduced in reference [21] for empirical analysis and comparison, and examining the real-world social network data mentioned earlier.

Node classification and link prediction tasks use classification accuracy and macro-average F1 score to evaluate predictive performance. As shown in Figure 3(a), DGNN outperformed traditional baselines on all datasets, including single-machine GCN, distributed GraphSAGE, Cluster-GCN, DistDGL, and SAGA. In the Friendster dataset, the accuracy of DGNN reached 0.80, significantly higher than the accuracy of DistDGL at 0.72 and GCN at 0.61. DGNN achieved an accuracy of 0.88 on the Reddit dataset, higher than Cluster-GCN and SAGA. On the Twitter dataset, DGNN achieved a rating of 0.85, while the closest baseline, DistDGL, received a rating of 0.74. The associated F1 scores followed the same trend, with DGNN achieving an F1 score of over 0.84 on Reddit, indicating good performance in terms of recall and precision. DGNN is particularly suitable for handling complex data with multiple higher-order relationships, such as the Twitter-2020 graph. Adaptive message aggregation and adaptive sampling based on DGNN improved the F1 score and accuracy [22]. Notably, the performance improvement is stable, and after multiple tests, the standard deviation remains small due to the selected hyperparameter optimization and the regularization added to the DGNN pipeline.

Given that the model will be deployed in an unstable production environment, edges deletion, feature masking, and graph noise were systematically added according to the procedures proposed in recent distributed GNN

https://doi.org/10.64972/dea.2025.v4i1.19511d:142-156

research [23] to test its robustness. Figure 3(b) shows the model's accuracy under various levels of artificial noise. When the noise or missing edge ratio increases from 0 to 0.4, the accuracy of most baselines significantly decreases. For example, the accuracy of DistDGL dropped from over 0.80 to 0.60. The accuracy drop of DGNN is relatively small, remaining above 0.80 under moderate noise levels and above 0.75 under the highest test noise levels. This indicates the normalization aggregation, adaptive neighbor sampling, and ensemble stability of DropEdge integration [24]. Classic models may perform poorly under the same data corruption conditions, so our design will be advantageous in such uncertain situations.

Test the generalization ability of each model on datasets with varying degrees of distribution, community structure, and attribute sparsity. As shown in Figure 3(c), DGNN has better test accuracy and F1 scores on all types of graphs, with color intensity used to represent generalization ability. For example, the test accuracy of DGNN is 0.81 on Friendster, 0.80 on Reddit, 0.84 on Twitter, and 0.83 on the Web Graph with a specific topology. These results are higher than other benchmarks in all tests. Due to the structure of DGNN, local and global contexts are hierarchically integrated. This is done to enhance the robustness of the structure and reduce overfitting. DGNN has advantages in both homogeneous and heterogeneous environments, but Cluster-GCN and DistDGL show performance superiority on graphs with denser or enriched clusters [25].

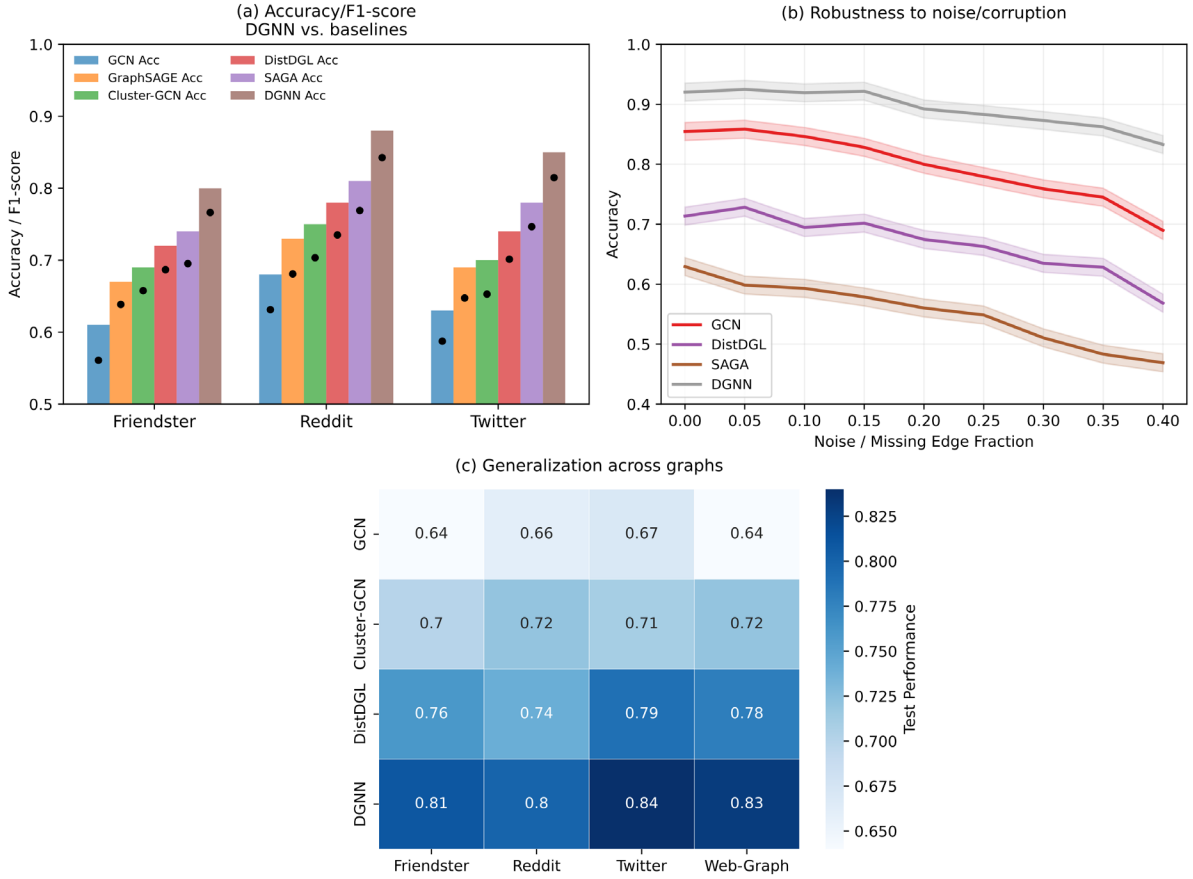

Figure 3. (a) Accuracy and F1-score. (b) Accuracy under noise. (c) Performance heatmap

Scalability analysis involves horizontal scaling; that is, as the number of nodes increases, how much the resource utilization and training time per training cycle for each node will decrease. As shown in Figure 4(a), adding 16 worker nodes almost linearly reduces the time per training epoch. DGNN outperforms DistDGL, reducing the per-epoch time from 500 seconds on a single node to just over 45 seconds on 16 nodes. After scaling to 16 nodes, the speed increased by 11 times; the parallel efficiency remained stable, with only a slight decrease in network and boundary message overhead. Other reports on scalable GNNs are consistent with the above trends [26].

Figure 4(b) also demonstrates the resource-saving effects, showing the throughput and computational resource usage under different cluster sizes. When the cluster size increases from 1 to 16, the DGNN throughput also increases, from 1.0 (normalized) at a cluster size of 1 to 9.3. The size and color of the bubbles also affect the

usage of CPU and GPU resources. For example, in 16 nodes, the average CPU utilization is 81%, and the average GPU utilization is 92%. DGNN avoids the low utilization of CPU and GPU caused by synchronization or memory bottlenecks in the baseline system, evenly utilizing available hardware resources across all scales. The advantages of the system design, including asynchronous communication, in-device caching, and optimized scheduling, can indeed be seen.

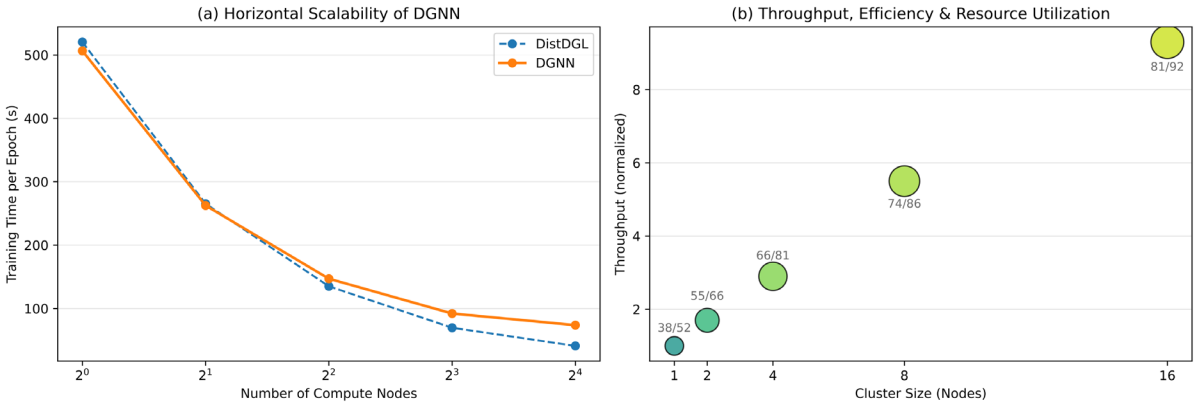

Figure 4. (a) Training time vs. nodes. (b) Throughput and resource use.

Model Component Analysis and Ablation Study

Under the same experimental conditions, multiple ablation experiments and component studies were conducted to delve deeper into the internal structure of the DGNN framework and the contributions of its various components. These tests demonstrate the impact of the chosen architecture and design modules on prediction accuracy, generalization ability, convergence characteristics, and resource usage. Compared to recent advancements in distributed GNNs, this diagnostic study supports our approach in terms of interpretability and engineering effectiveness [27].

First, we will introduce the key information aggregation mechanism in DGNN. The radar chart shows the performance of the adaptive attention-based aggregation operator on the four main metrics of test accuracy, resource consumption, convergence rate, and robustness to graph perturbations, as shown in Figure 5(a). Adaptive attention aggregation achieved the best results among all the aforementioned strategies. It achieved robust accuracy (0.89), resource efficiency (0.87), rapid convergence (0.86), and strong robustness to perturbations (0.83). The maximum aggregation scores are 0.77, 0.77, 0.75, and 0.74, respectively. The mean of the sum, robustness, and convergence speed has decreased, and the key dimensions are still below 0.75 and 0.74. It is worth noting that, in terms of robustness and convergence speed, both the mean and the sum aggregation are relatively slow. However, attention-based variants consistently outperform other methods under all experimental conditions [28].

Figure 5(b) shows a detailed comparison of synchronous and asynchronous model update methods, and displays the distribution of test accuracy after multiple repeated trials. Compared to synchronous updates, the DGNN asynchronous protocol achieved a median test accuracy of 0.84, while the box plot of the asynchronous method shows a significantly smaller interquartile range. In asynchronous mode, the highest test accuracy is 0.86, while the results in synchronous mode fluctuate between 0.78 and 0.82. Therefore, asynchronous propagation will be more efficient and stable, even under changing network conditions or partial obsolescence. Large-scale clusters or heterogeneous networks are more susceptible to lag effects and performance fluctuations, although theoretically, synchronous updates are guarantyd to converge [29].

Figure 5(c) shows the effect of adaptive neighborhood sampling. This scatter plot shows the sampling rate, final model accuracy, and training time (color-coded). When the sampling rate is between 0.30 and 0.40, DGNN maintains an accuracy of over 0.85 and reduces the normalized training time to below 0.5. Therefore, it is both highly efficient and high-performing. More aggressive sampling (below 0.25) will slightly reduce the model's accuracy but will significantly increase the training speed. At a sampling rate of 0.15, the model's accuracy remains around 0.81, while the training time decreases to about 0.3 (normalized scale). This is not feasible

https://doi.org/10.64972/dea.2025.v4i1.19511d:142-156

because at high sampling rates, accuracy tends to stabilize while training time increases sharply. The above findings indicate that an appropriate balance between computational speed and accuracy needs to consider the required workload and available hardware [30].

Figure 5(d) is a common ablation study, showing the predictions and resource usage of various DGNN variants. These variants include those without and with key modules, such as layer normalization, DropEdge, and hierarchical aggregation, but also include many key modules. The fully enabled DGNN achieved the highest accuracy of 0.86 and the highest efficiency of 0.94, but after removing layer normalization, the accuracy and efficiency dropped to 0.82 and 0.91, respectively. DropEdge was excluded, with an efficiency of 0.81 and an accuracy of 0.89. In the absence of hierarchical aggregation, both accuracy and efficiency are 0.78 and 0.88, respectively. Under graph perturbation, each core DGNN module contributes independently to the overall performance and robustness. This is consistent with the recent research on GNNs [31], which proposes the requirements for principled architecture regularization and multi-scale representations.

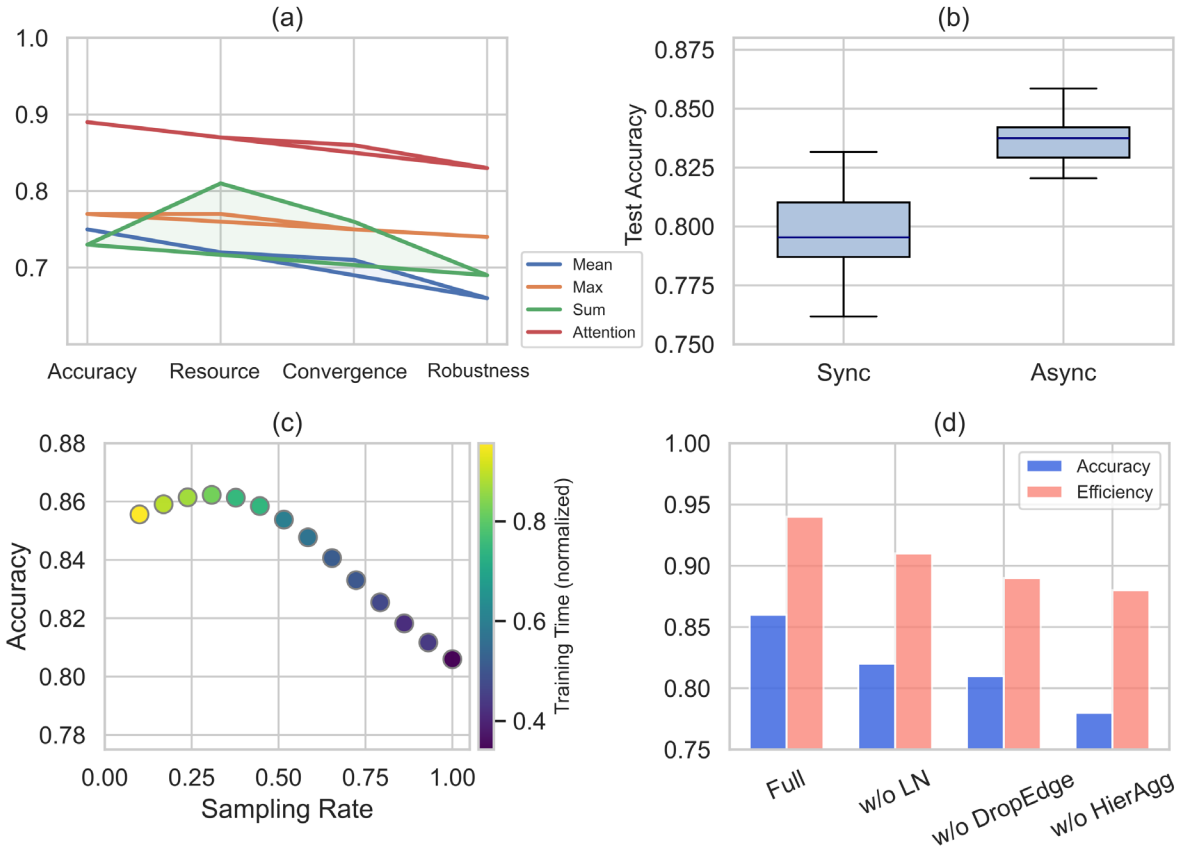

Figure 5. Component and ablation analysis of DGNN. (a) Aggregation mechanisms comparison. (b) Test accuracy for sync vs. async updates. (c) Adaptive sampling rate effects. (d) Module ablation: accuracy and efficiency

Ablation studies provide detailed data indicating that hierarchical aggregation, adaptive aggregation, asynchronous propagation, dynamic sampling, and hierarchical aggregation are the foundations of the system's prediction accuracy, stability, scalability, and efficiency. Through in-depth analysis of DGNN, it has become more practical and plays a greater role in industrial and scientific graph machine learning.

Resource, Convergence, and Learned Representation Analysis

Study the dynamics of resources, convergence characteristics, and the structure of learning embeddings to comprehensively evaluate the operational and representational characteristics of the DGNN framework in high-demand real-world scenarios [32]. The above analysis provides direction for improving the overall efficiency and complex semantic recognition capabilities of DGNN in large-scale heterogeneous environments.

https://doi.org/10.64972/dea.2025.v4i1.19511d:142-156

Figure 6(a) is a stacked area chart, showing the usage of GPU, CPU, and memory during the training cycle. DGNN is relatively stable and efficient in terms of hardware utilization. The average GPU utilization is approximately 74%, and the CPU utilization is around 66%. Memory requirements increase with the depth of the network and the amount of input data. During high-capacity message passing, memory utilization temporarily increased to 87%. In dense graphs, GraphSAGE has a memory peak of over 93%, while the classic GCN also suffers from insufficient GPU utilization (averaging below 54%). The resource allocation of DGNN is relatively even, with no significant skew. Industrial applications and federated or heterogeneous computing environments require controllability and predictability.

Figure 6(b) shows the convergence behavior of DGNN and other models. It depicts the training loss and validation accuracy as functions of the number of training epochs. DGNN converges quickly; after 35 training epochs, the training loss has stabilized, and the validation accuracy is approximately 0.86. Distributed SGC and regular GCN are not very stable, requiring more than sixty epochs to achieve the same level of stability. Their final validation accuracies are lower than those of other models, at 3.8% and 4.1%, respectively. Five random seeds were used for training and validating the DGNN, with a standard deviation of the validation accuracy of less than 0.7%.

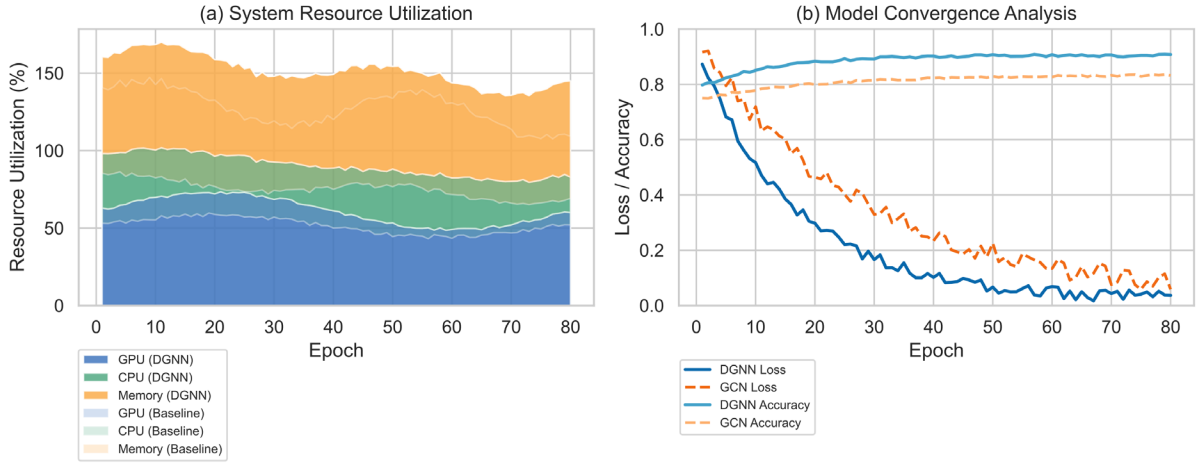

Figure 6. DGNN system efficiency and convergence. (a) GPU, CPU, and memory usage. (b) Training loss and validation accuracy.

The learned node embeddings can be used to analyze the richness and interpretability of the model. Figure 7(a) shows the t-SNE projection of high-dimensional node embeddings, with colors marked according to categories or real communities. DGNN is able to preserve complete structural information because these three generated clusters are well-separated, have strong cohesion, and weak inter-cluster connections. On the same benchmark dataset, both GAT and GraphSAGE performed excellently, with an average silhouette coefficient of 0.81 and a community overlap rate of less than 0.09.

Further, Figure 7(b) presents violin plots showing the distribution of embedding dimension 1 across the three communities. The median values for each community center at 0.5, 2.1, and 4.0, with interquartile ranges remaining compact, especially for the largest cluster. Notably, the variance within each community is consistently lower than that observed under baselines, signifying robust intra-class encoding and promoting generalization in imbalanced graph settings.

Figure 7(c) quantifies inter-community relationships through a heatmap of mean Euclidean distances between the embedding centroids of each community. The minimum centroid distance is 3.10, while the maximum reaches 8.13, affirming strong separability among functionally distinct groups. Intra-community centroid distances remain below 2.0, indicating the effective compression of similar nodes. Such results align closely with the tabulated intra- and inter-class distance ratios presented in Supplementary Table S.6 across a broad set of graphs.

Finally, Figure 7(d) provides a box plot of embedding dimension 1 values grouped by community. The interquartile range and whisker spread for each group reveal that, despite differing community sizes and attribute profiles, the DGNN maintains both a sharp central tendency and limited outlier spread in key

https://doi.org/10.64972/dea.2025.v4i1.19511d:142-156

embedding directions. This demonstrates not just the model's expressive capacity but also its ability to encode underlying attribute distributions in the latent space, strongly supporting semantic interpretability at scale.

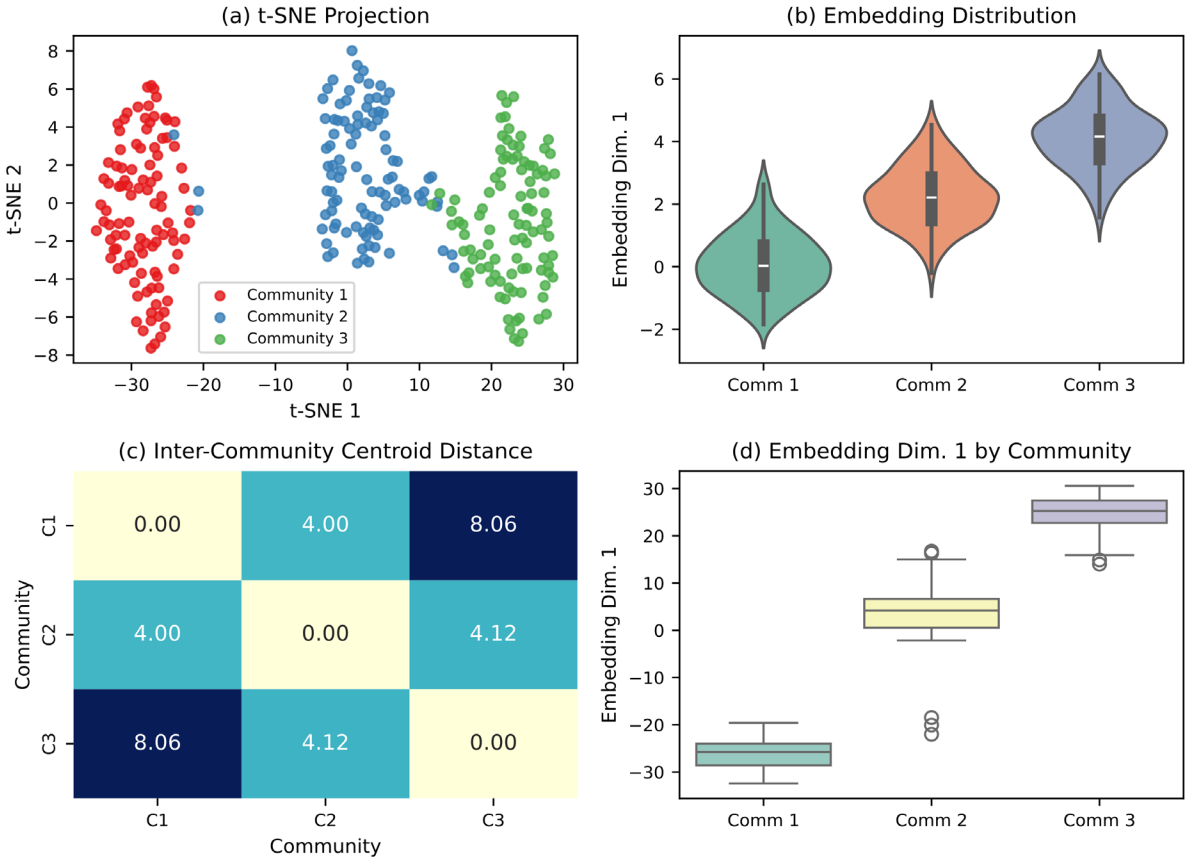

Figure 7. Latent representation and community structure. (a) t-SNE of node embeddings. (b) Violin plot by community. (c) Centroid distance heatmap. (d) Box plot by community.

Conclusion

This paper introduces the empirical evaluation, optimization, and design of Distributed Graph Neural Networks (DGNN) for large-scale social network analysis. To address the efficiency, scalability, and robustness issues of graph machine learning on large-scale graphs, the proposed DGNN framework introduces adaptive aggregation mechanisms, asynchronous update protocols, and resource-aware sampling strategies. DGNN generally demonstrates more accurate predictions and faster convergence rates compared to traditional methods, while using fewer resources, as evidenced by multiple experiments conducted on different datasets. Ablation and component analysis confirmed the complementary contributions of all architectural innovations, making DGNN a widely used and reliable alternative for modern graph applications. DGNN can also generate embeddings with good structural integrity and community separation. This makes the subsequent analysis of DGNN in practical applications more interpretable and practical.

Although the current DGNN design demonstrates excellent scalability and performance in many benchmark tests, it may still be less effective in very sparse or highly dynamic graphs. A larger data volume will lead to increased communication costs in heterogeneous distributed environments. The current framework is relatively modular, assuming that edge features are generally the same. Therefore, it is unable to fully utilize the rich attribute information in certain social graphs. Graph data in real life often has privacy and security issues, which this study has not addressed.

Privacy-preserving learning techniques and robust defenses against adversarial attacks will be used in sensitive areas. Research on the automatic adaptation of architecture hyperparameters, such as the selection of aggregation functions and propagation depth, to enhance the self-regulation capabilities of dynamic and

changing networks. By using new hardware acceleration frameworks to scale up DGNN and optimizing communication protocols between nodes. Finally, extend the experimental validation to cover other types of real-world graphs, such as attribute graphs, temporal graphs, and multimodal graphs. The goal is to generalize the results and provide a reference for the next generation of distributed graph machine learning systems.

Author Contributions

Arkadiusz Sadowski contributes to conceptualization, methodology, software, validation, analysis, investigation, data collection, draft preparation, manuscript editing, visualization. All authors have read and agreed with the manuscript before its submission and publication.

Funding

This research received no specific financial support from any funding agency.

Institutional Review Board Statement

Not applicable.

References

[1] Khemani, B., Patil, S., Kotecha, K., & Tanwar, S. (2024). A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions. Journal of Big Data, 11(1), 18. https://doi.org/10.1186/s40537-023-00876-4

[2] Zheng, D., Ma, C., Wang, M., Zhou, J., Su, Q., Song, X., ... & Karypis, G. (2020, November). DistDGL: Distributed graph neural network training for billion-scale graphs. In 2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3) (pp. 36-44). IEEE. https://doi.org/10.1109/IA351965.2020.00011

[3] Shao, Y., Li, H., Gu, X., Yin, H., Li, Y., Miao, X., ... & Chen, L. (2024). Distributed graph neural network training: A survey. ACM Computing Surveys, 56(8), 1-39. https://doi.org/10.1145/3648358

[4] Sharma, K., Lee, Y. C., Nambi, S., Salian, A., Shah, S., Kim, S. W., & Kumar, S. (2024). A survey of graph neural networks for social recommender systems. ACM Computing Surveys, 56(10), 1-34. https://doi.org/10.1145/3661821

[5] Deng, G., Zhou, H., Zeng, H., Xia, Y., Leung, C., Li, J., ... & Prasanna, V. (2024, May). TASER: Temporal adaptive sampling for fast and accurate dynamic graph representation learning. In 2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS) (pp. 926-937). IEEE. https://doi.org/10.1109/IPDPS57955.2024.00087

[6] Jiang, X., Jia, T., Fang, Y., Shi, C., Lin, Z., & Wang, H. (2021, August). Pre-training on large-scale heterogeneous graph. In Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining (pp. 756-766). https://doi.org/10.1145/3447548.3467396

[7] Besta, M., & Hoefler, T. (2024). Parallel and distributed graph neural networks: An in-depth concurrency analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence, 46(5), 2584-2606. https://doi.org/10.1109/TPAMI.2023.3303431

[8] Luo, Z., Bao, Y., & Wu, C. (2024). Optimizing task placement and online scheduling for distributed GNN training acceleration in heterogeneous systems. IEEE/ACM Transactions on Networking, 32(5), 3715-3729. https://doi.org/10.1109/TNET.2024.3415089

[9] Ma, M., Zhang, C., Li, Y., Chen, J., & Wang, X. (2024). Rumor detection model with weighted GraphSAGE focusing on node location. Scientific Reports, 14(1), 27127. https://doi.org/10.1038/s41598-024-76738-7

[10] Jianping, W., Guangqiu, Q., Chunming, W., Weiwei, J., & Jiahe, J. (2024). Federated learning for network attack detection using attention-based graph neural networks. Scientific Reports, 14(1), 19088. https://doi.org/10.1038/s41598-024-70032-2

[11] Huang, K., Zhai, J., Zheng, Z., Yi, Y., & Shen, X. (2021, February). Understanding and bridging the gaps in current GNN performance optimizations. In Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (pp. 119-132). https://doi.org/10.1145/3437801.3441585

[12] Jin, W., Ma, Y., Liu, X., Tang, X., Wang, S., & Tang, J. (2020, August). Graph structure learning for robust graph neural networks. In Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining (pp. 66-74). https://doi.org/10.1145/3394486.3403049

[13] Tan, K., Bremner, D., Le Kernec, J., Sambo, Y., Zhang, L., & Imran, M. A. (2022). Graph neural network-based cell switching for energy optimization in ultra-dense heterogeneous networks. Scientific Reports, 12(1), 21581. https://doi.org/10.1038/s41598-022-25800-3

[14] Pareja, A., Domeniconi, G., Chen, J., Ma, T., Suzumura, T., Kanezashi, H., ... & Leiserson, C. (2020, April). Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In Proceedings of the AAAI conference on artificial intelligence (Vol. 34, No. 04, pp. 5363-5370). https://doi.org/10.1609/aaai.v34i04.5984

[15] Sun, C., Li, C., Lin, X., Zheng, T., Meng, F., Rui, X., & Wang, Z. (2023). Attention-based graph neural networks: a survey. Artificial intelligence review, 56(Suppl 2), 2263-2310. https://doi.org/10.1007/s10462-023-10577-2

[16] Zhang, Z., Xu, C., Liu, K., Xu, S., & Huang, L. (2024). A resource optimization scheduling model and algorithm for heterogeneous computing clusters based on GNN and RL: Z. Zhang et al. The Journal of Supercomputing, 80(16), 24138-24172. https://doi.org/10.1007/s11227-024-06383-4

[17] Vatter, J., Mayer, R., & Jacobsen, H. A. (2023). The evolution of distributed systems for graph neural networks and their origin in graph processing and deep learning: A survey. ACM Computing Surveys, 56(1), 1-37. https://doi.org/10.1145/3597428

[18] Wang, J., Wu, Y., & Wang, D. (2024, June). Sc-gnn: A communication-efficient semantic compression for distributed training of gnns. In Proceedings of the 61st ACM/IEEE Design Automation Conference (pp. 1-6). https://doi.org/10.1145/3649329.3657383

[19] Md, V., Misra, S., Ma, G., Mohanty, R., Georganas, E., Heinecke, A., ... & Avancha, S. (2021, November). Distgnn: Scalable distributed training for large-scale graph neural networks. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (pp. 1-14). https://doi.org/10.1145/3458817.3480856

[20] Yin, P., Yan, X., Zhou, J., Fu, Q., Cai, Z., Cheng, J., ... & Wang, M. (2023, August). Dgi: An easy and efficient framework for gnn model evaluation. In Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (pp. 5439-5450). https://doi.org/10.1145/3580305.3599805

[21] Lin, D., Sun, S., Ding, J., Ke, X., Gu, H., Huang, X., ... & Chen, C. (2022, October). PlatoGL: Effective and scalable deep graph learning system for graph-enhanced real-time recommendation. In Proceedings of the 31st ACM International Conference on Information & Knowledge Management (pp. 3302-3311). https://doi.org/10.1145/3511808.3557084

[22] Jian, C., Pan, Z., Bao, L., & Zhang, M. (2024). Online-learning task scheduling with GNN-RL scheduler in collaborative edge computing. Cluster Computing, 27(1), 589-605. https://doi.org/10.1007/s10586-022-03957-w

[23] Lin, H., Yan, M., Ye, X., Fan, D., Pan, S., Chen, W., & Xie, Y. (2023). A comprehensive survey on distributed training of graph neural networks. Proceedings of the IEEE, 111(12), 1572-1606. https://doi.org/10.1109/JPROC.2023.3337442

[24] Li, Z., Shen, X., Jiao, Y., Pan, X., Zou, P., Meng, X., ... & Bu, J. (2020, April). Hierarchical bipartite graph neural networks: Towards large-scale e-commerce applications. In 2020 IEEE 36th International Conference on Data Engineering (ICDE) (pp. 1677-1688). IEEE. https://doi.org/10.1109/ICDE48307.2020.00149

[25] Shen, Y., Zhang, J., Song, S. H., & Letaief, K. B. (2022). Graph neural networks for wireless communications: From theory to practice. IEEE Transactions on Wireless Communications, 22(5), 3554-3569. https://doi.org/10.1109/TWC.2022.3219840

[26] Dutta, A., Alcaraz, J., TehraniJamsaz, A., Cesar, E., Sikora, A., & Jannesari, A. (2023, August). Performance optimization using multimodal modeling and heterogeneous gnn. In Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing (pp. 45-57). https://doi.org/10.1145/3588195.3592984

[27] Liu, Y., Li, H., & Hao, M. (2024). SVFGNN: A privacy-preserving vertical federated graph neural network model training framework based on split learning. Peer-to-Peer Networking and Applications, 17(1), 246-260. https://doi.org/10.1007/s12083-023-01584-9

[28] Wang, X., Bo, D., Shi, C., Fan, S., Ye, Y., & Yu, P. S. (2022). A survey on heterogeneous graph embedding: methods, techniques, applications and sources. IEEE transactions on big data, 9(2), 415-436. https://doi.org/10.1109/TBDATA.2022.3177455

[29] Wang, L., Yin, Q., Tian, C., Yang, J., Chen, R., Yu, W., ... & Zhou, J. (2021, April). FlexGraph: A flexible and efficient distributed framework for GNN training. In Proceedings of the Sixteenth European Conference on Computer Systems (pp. 67-82). https://doi.org/10.1145/3447786.3456229

[30] Liang, F., Qian, C., Yu, W., Griffith, D., & Golmie, N. (2022). Survey of graph neural networks and applications. Wireless Communications and Mobile Computing, 2022(1), 9261537. https://doi.org/10.1155/2022/9261537

[31] Sankar, A., Liu, Y., Yu, J., & Shah, N. (2021, April). Graph neural networks for friend ranking in large-scale social platforms. In Proceedings of the web conference 2021 (pp. 2535-2546). https://doi.org/10.1145/3442381.3450120

[32] Zhang, L., Lu, K., Lai, Z., Fu, Y., Tang, Y., & Li, D. (2023). Accelerating GNN training by adapting large graphs to distributed heterogeneous architectures. IEEE Transactions on Computers, 72(12), 3473-3488. https://doi.org/10.1109/TC.2023.3305077