Improving Link Prediction in Patent Citation Networks

Yi Liang, Yongtao Mu, Jin Yan, Xiaohu Zhang
Georgia Institute of Technology
yliang389@gatech.edu, ymu42@gatech.edu, jyan83@gatech.edu, xzhang869@gatech.edu

Abstract

Modern patent citation networks are vital for tracking technological innovation, yet predicting new connections (links) within these networks remains a challenge due to their extreme sparsity and dynamic nature. This paper presents an extensive evaluation of state-of-the-art graph-based models, including Structural Deep Network Embedding (SDNE), node2vec, PyTorch-BigGraph (PBG), and Graph Attention Networks (GAT), applied to patent citation and BlogCatalog datasets. We propose enhancements like negative sampling and positional encodings to address dataset-specific challenges, significantly improving link prediction performance. Our results demonstrate that GAT outperforms existing methods, achieving notable improvements in both precision@K and AUC metrics. The findings offer robust tools for technological trend forecasting and emphasize the impact of targeted model adjustments in sparse and dynamic graph environments.

1. Introduction

Patent citation networks are essential for understanding technological innovation by mapping relationships between patents. These networks help researchers and inventors track technological evolution and forecast emerging trends. The objective of this project is to improve the prediction of connections, or links, within these networks. Link prediction involves estimating which earlier patents are likely to be cited by newer patents based on existing data. Accurate link prediction can uncover hidden patterns, forecast future citations, and support innovation tracking.

Currently, link prediction is tackled using either heuristic-based methods or embedding-based approaches. Heuristic methods, such as common neighbors or preferential attachment, rely on predefined rules derived from graph structure but fail to capture complex patterns, especially in sparse networks. Embedding-based approaches, such as node2vec [4], and Structural Deep Network Embedding (SDNE) [11], generate low-dimensional representations of graph nodes for link prediction tasks. While these methods perform well in dense networks, they struggle with challenges like sparsity, temporal dynamics, and scalability in real-world citation networks. Additionally, these models often overfit to dense subgraphs and fail to generalize for sparsely connected nodes.

This project evaluates the performance of state-of-the-art models on two datasets. The first is the BlogCatalog dataset [12], a moderately dense graph, commonly used as a benchmark for link prediction tasks. The second is the Patent Citation dataset [5], a sparse real-world graph. To tackle the inherent challenges of sparsity and vast scale in these networks, our approach utilizes Graph Neural Networks (GNNs). GNNs are adept at learning from large, sparse, and dynamic graphs—qualities typical of patent citation networks. This project explores various graph mining techniques, including established methods like SDNE, node2vec, and PyTorch BigGraph [7] from reference paper [5]. Additionally, we introduce a novel application of GNN, the Graph Attention Networks (GAT) [10], which dynamically allocates attention to nodes, enhancing the prediction of links by focusing on the most relevant features within the graph.

2. Approach

Our goal was to reconstruct missing links and predict future links between nodes by leveraging graph structure and embedding-based techniques. The project addressed two tasks: graph reconstruction and link prediction.

2.1. Dataset Preparation

The BlogCatalog dataset [12] was selected as a benchmark due to its moderate size and density. This dataset has been widely used in prior studies, such as [1], making it an ideal choice for validating our implementations. The degree distribution (Figure 1a) reveals a diverse range of node degrees, offering an opportunity to test methods that are sensitive to structural variability.

The Patent Citation dataset [5] represents a sparse real-world graph with a significantly larger scale. Due to its computational complexity, we focused on a subset of data from 1988–1990. This subset retains the structural sparsity

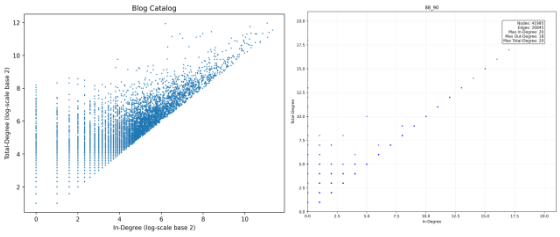

Figure 1. Dataset degree distribution: (a) Blog Catalog; (b) 1988 - 1990 Patent citation subgraph.

and temporal characteristics of the full dataset, providing an efficient yet representative test environment (Figure 1b). Table 1 summarizes the key statistics of the datasets used.

Dataset	nodes	edges
BlogCatalog	10,312	333,983
Patent Citation	2923922	16522438
Patent Citation (1988-1990)	41,995	28,125

Table 1. Statistics of nodes and edges for different datasets.

The preprocessing steps for both datasets aimed to construct reliable training and testing graphs, ensuring an unbiased evaluation of the models' generalization capabilities. For BlogCatalog, the training graph consisted of established connections, where edges represented positive samples. For the Patent Citation dataset, a subset of citation data was used to represent historical relationships effectively.

Negative samples were generated for both datasets to represent nonexistent connections. The ratio of negative to positive samples was kept fixed to maintain balance. To avoid overlap between training and testing sets, all training edges were excluded from the test graphs in both datasets. This ensured that the models were evaluated on their ability to predict unseen links. By preserving these distinctions, the preprocessing framework supported robust testing of graph reconstruction and link prediction tasks, addressing both structural and temporal challenges.

2.2. Proposed Models

We explored multiple models to solve the problem, leveraging their unique strengths for graph representation.

2.2.1 Structural Deep Network Embedding

Structural Deep Network Embedding (SDNE) is a framework for learning graph representations, particularly effective for both sparse and dense networks. It uses deep autoencoders to optimize two objectives: preserving first-order proximity, which ensures that connected nodes in the graph remain close in the embedding space, and second-order proximity, which captures structural similarities between unconnected nodes by reconstructing the adjacency matrix. This dual-objective approach makes SDNE well-suited for capturing the complex relationships network.

2.2.2 Random Walk Based Method: node2vec

Node2vec [4] is a semi-supervised algorithm designed to transform complex network structures into low-dimensional vector representations. It leverages a parametrized random walk strategy to capture both homophily and structural equivalence. The algorithm introduces two key parameters p (return parameter) and q (in-out parameter), which control the walk's exploration behavior, allowing transitions between depth-first and breadth-first network traversals. The random walk mechanism dynamically adjusts node sampling probabilities, enabling the algorithm to generate embeddings that reflect both local neighborhood characteristics and global structural roles.

The random walk sequences generated by node2vec are then passed to the word2vec algorithm [8] to learn node embeddings. word2vec is a widely used natural language processing technique that learns dense vector representations of words by capturing semantic relationships based on their contextual usage. It uses two main architectures: skip-gram and continuous bag-of-words (CBOW) to generate embeddings. In the case of node2vec, word2vec utilizes these architectures to learn meaningful node embeddings from the generated sequences, enabling the representation of complex graph structures in a way that is useful for downstream tasks like edge reconstruction and link prediction.

2.2.3 Pytorch-BigGraph

Pytorch-BigGraph (PBG) [7] is a highly scalable framework designed for learning embeddings from massive graphs. It supports efficient training by partitioning the graph and embedding tables, enabling distributed computation across multiple machines. This makes PBG well-suited for large datasets like the Patent Citation graph.

PBG optimizes embeddings by representing nodes and edges in a vector space, where the embeddings are learned such that relationships between nodes (edges) are preserved. The core idea of PBG is to use a score function to determine the likelihood of an edge existing between two nodes based on their embeddings. Specifically, the score function is:

$$s(u, v) = \phi(\mathbf{u}, \mathbf{v}, \mathbf{R}), \tag{1}$$

where $\mathbf{u}$ and $\mathbf{v}$ are the embeddings of nodes u and v, respectively; $\mathbf{R}$ represents the embedding of the relation type (if applicable, in multi-relational graphs); $\phi(\cdot)$ is a scoring function, such as the *dot product*, *L2 norm*, or a parameterized distance metric.

2.2.4 Graph Attention Networks

GAT [10] builds upon Graph Convolutional Networks and leverages the powerful attention mechanism, which is the foundation of the transformer architecture. The attention mechanism combines the messages from the neighboring nodes with a weighted attention function:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(a^\top \left[\mathbf{W}\mathbf{h}_i \,\|\, \mathbf{W}\mathbf{h}_j\right]\right)\right)}{\sum_{k\in\mathcal{N}_i} \exp\left(\text{LeakyReLU}\left(a^\top \left[\mathbf{W}\mathbf{h}_i \,\|\, \mathbf{W}\mathbf{h}_k\right]\right)\right)}$$

The attention parameter α can be thought of as a feed-forward neural network that combines the embedding vectors at the node level, parameterized by a weighted vector $a^\top$, and then applies the LeakyReLU nonlinearity. Similar to the attention mechanism in transformers, the graph attention network can be extended by introducing multiple attention heads (*multi-head attention*) with distinct sets of attention weights.

GAT Architecture

Fortunately, the implementation of GAT is available in the PyTorch Geometric (PyG) [3] library, which is commonly used for deep learning on graphs and other irregular structures. We use the PyG library to build the GAT model for predicting tasks.

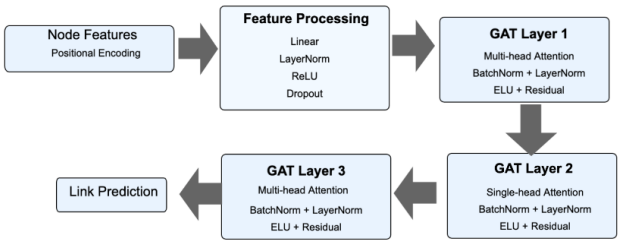

Figure 2. GAT Model Architecture

Figure 2 shows the architecture of the GAT model. The GAT architecture employs three key layers: multiple heads in the first layer aggregate neighborhood features into complementary representations, a single head in the second layer distills this information, and multiple heads in the final layer expand the model's capacity to capture diverse graph structures. While PyG [3] typically suggests two layers for feature-rich graphs, we add a third layer to better process relationships from positional encodings and capture more complex graph patterns. Dropout was applied in every layer to prevent overfitting. BatchNorm and LayerNorm were used during training to stabilize learning and improve convergence. In terms of the activation functions, the GAT model use both the Exponential Linear Unit (ELU) for the hidden layer and the Leaky Rectified Linear Unit (LeakyReLU) activations for the attention layer, which introduce non-linearity to the model. Binary Cross-Entropy Loss was used for predicting the existence of links as a classification problem. Similar to the example provided by PyG [3], we choose Adam Optimizer due to its stability and adaptability in GNNs.

Positional embedding of the GAT

Since BlogCatalog and Patent Citation datasets only provide topological information rather than node features, we enhance the GAT model by implementing positional embeddings, adapted from transformer architectures([9]), to generate meaningful initial node features that encode unique structural positions within the graph.

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

The feature preprocessing layer, consisting of Linear, LayerNorm, ReLU, and Dropout components, standardizes and transforms the positional embeddings before they enter the GAT layers.

2.3. Evaluation Metrics

To evaluate the performance of our models, we perform two key tasks: link reconstruction and link prediction, which correspond to in-sample and out-of-sample testing, respectively, in statistical approaches. The *link reconstruction* task assesses the model's ability to recover existing edges, while the *link prediction* task evaluates the model's capacity to predict new, unseen edges.

2.3.1 Precision@K

Our primary evaluation metric is **Precision@K**, which measures the proportion of correctly predicted edges among the top-K most confidently predicted edge pairs.

Link reconstruction: following the approach from [1] and [6], we first generate edge embeddings by performing element-wise multiplication of the node embeddings. Next, we randomly select 15% of the training edges for evaluation. These edges are temporarily removed from the training graph to create a modified network for testing the model's reconstruction capabilities. We then train a logistic regression classifier on the edge embeddings to predict edge existence between node pairs. The classifier outputs probability scores, which are ranked for precision@K evaluation at various values of K.

Link prediction: Similarly, we test the model's ability to predict edges from the test set, which represent future or held-out connections. As with link reconstruction, edge

embeddings are obtained, and a logistic regression classifier is trained to predict the likelihood of edge formation between node pairs. The predictions are ranked by probability scores, and precision@K metrics are calculated to evaluate the model's performance at different values of K.

2.3.2 AUC

We also use the Area Under Curve (**AUC**) score as a complementary metric for evaluating model performance. The AUC score ranges from 0 to 1, with a score of 1.0 indicating perfect classification, and 0.5 representing random chance performance. While precision@K evaluates the accuracy of predictions within the top-K results, AUC offers a comprehensive measure of how well the model ranks positive links higher than negative ones across all thresholds. This makes AUC particularly valuable for comparing models' overall predictive power.

3. Experiments and Results

3.1. Experimental Setup

3.1.1 Structural Deep Network Embedding

The original SDNE implementation, sourced from a TensorFlow-based repository [11] was rewritten in PyTorch to better align with our project requirements. Initially, the same configuration was applied to both datasets, including an embedding size of 100. While the setup performed adequately for the dense BlogCatalog dataset, it failed on the sparse Patent Citation dataset, where Precision@K scores were uniformly 0. To address this issue, we reduced the embedding size to 40, reasoning that smaller embedding dimensions would prevent overfitting on sparse connections and better capture the limited structural information.

3.1.2 Node2vec

Node2vec and word2vec are integrated in experiments to generate feature representations of nodes. node2vec performs random walks on the graph to create sequences of nodes that encode both local and global structural properties. These sequences are then feed into word2vec to learn low-dimensional embeddings for the nodes, capturing their relationships and similarities in the graph.

To extend the embeddings to edges, the learned node embeddings are combined using operations like element-wise multiplication, addition, or other binary operations to create edge features. These features are used to train a logistic regression model for predicting whether specific node pairs are connected in the graph.

3.1.3 Pytorch-BigGraph

The datasets were preprocessed to adhere to the input requirements of PBG, and embeddings were learned using the configurations described below. The key experimental parameters for Blog Catalog dataset are as follows: embedding size: 1024; training epochs: 30; learning rate: 0.001; loss Function: the default scoring mechanism of PBG, "ranking", was used.These parameters were chosen to efficiently train the model on the relatively dense connectivity of the graph, enabling quick convergence. For the patent citation dataset, same embedding size of 1034 used, but training Epochs increase to 100; learning rate increased to 0.01; and loss function of "Softmax". The increased number of training epochs accounts for the sparse connectivity of the graph, ensuring sufficient updates for all nodes and edges. The decreased learning rate helps stabilize training and prevent overfitting, especially since the sparsity can lead to noisier gradients compared to the dense BlogCatalog dataset. Both datasets utilized a single entity type and a single relation type, with no additional transformations ('operator="none"').

3.1.4 Graph Attention Networks

The GAT model is optimized through three key hyperparameter categories: model architecture parameters (dropout rate, attention heads, etc.), optimization parameters (learning rate, weight decay, learning schedule, etc.), and training parameters (batch size, epochs, early stopping patience, etc.). The model learns node embeddings and computes similarity scores between node pairs, which are transformed through sigmoid activation to predict edge probabilities. During training, the model's performance is monitored using AUC to guide optimization and parameter updates. Initially, a learning rate scheduler was employed, but experimentation revealed that the best model performance in terms of AUC was consistently achieved with a fixed learning rate of 0.0001. Gradient clipping is also implemented to prevent gradient explosion during model training.

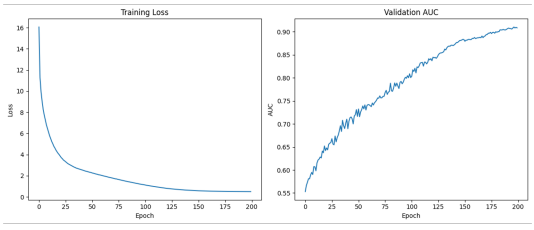

Figure 3. AUC During the training of BlogCatalog

As shown in Figure 3, the AUC performance fluctuated over time during training. To mitigate potential performance deterioration while optimizing computational efficiency, a patience parameter of 15 was introduced to halt

training if performance declined continuously. The epoch size was set to a sufficiently large value of 200, consistent with the methodology described in [1]. However, training often terminated early due to the effectiveness of the patience mechanism and the checkpoint saves the best-performing model during training.

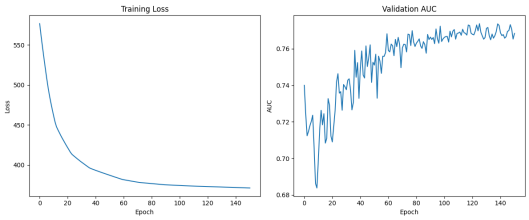

Figure 4. AUC During the training of Patent Citation

For the Patent Citation dataset, we maintained the similar architecture as BlogCatalog for consistency. We increased the dropout rate to 0.3 for stronger regularization, reduced batch size to match the smaller dataset scale, and decreased input feature dimensions to prevent overfitting. The training curves Figure 4 show a steady decrease in loss function and a gradually improving validation AUC from 0.68 to 0.78, with some volatility early in training but stabilizing after epoch 80. The scheduler effectively managed the learning rate decay, allowing the model to reach stable performance over 150 epochs.

3.2. Performance

The performance of different models was evaluated using Precision@K for both graph reconstruction and link prediction tasks (Table 2 through Table 5), as well as the AUC metrics (Figure 5).

3.2.1 Blog Catalog Dataset

For the dense BlogCatalog dataset, GAT achieved the highest Precision@K and AUC scores, outperforming other methods in both reconstruction and link prediction tasks.

SDNE: SDNE achieved reasonable Precision@K values, particularly in reconstruction tasks, but its AUC score indicated limited discriminative power for link prediction compared to other methods.

Node2vec: node2vec provided robust results for reconstruction tasks and demonstrated superior performance compared to prior benchmarks, particularly when negative sampling was employed. The AUC score for node2vec was slightly lower than SDNE, but its performance remained consistent across different values of K. While node2vec performed well in reconstruction tasks, its predictive power for link prediction was more limited in the sparse Patent Citation dataset compared to the denser Blog Catalog dataset.

PBG: PBG displayed consistent performance, achieving a balance between Precision@K and AUC. Its scalability and ability to process dense graphs efficiently made it a strong baseline, although it lagged behind GAT in terms of overall accuracy.

GAT: GAT delivered superior results across all metrics, achieving a perfect Precision@K for $K = 10, 50$, and 100 in both tasks. It also recorded the highest AUC score of 0.95, showcasing its ability to capture intricate relationships within dense graphs effectively.

Precision@K	SDNE	node2vec	PBG	GAT
10	1	1	0.6	1
50	0.9	1	0.72	1
100	0.75	1	0.75	1
500	0.764	0.998	0.856	0.994
1000	0.718	0.996	0.862	0.994

Table 2. Blog Catalog reconstruction results.

Precision@K	SDNE	node2vec	PBG	GAT
10	1	1	0.5	1
50	0.72	1	0.5	1
100	0.7	0.97	0.48	1
500	0.468	0.982	0.47	0.998
1000	0.418	0.973	0.457	0.998

Table 3. Link prediction results for Blog Catalog.

3.2.2 Patent Citation Dataset

The sparse Patent Citation dataset presented significant challenges, where GAT once again emerged as the most effective model.

Precision@K	SDNE	node2vec	PBG	GAT
10	0.2	0.5	0.5	1
50	0.28	0.7	0.44	1
100	0.32	0.61	0.21	0.98
500	0.278	0.61	0.266	0.995
1000	0.262	0.57	0.2750	0.996

Table 4. Reconstruction results for Patent Citation (1988-1990).

Precision@K	SDNE	node2vec	PBG	GAT
10	0	0.4	1	1
50	0.08	0.54	0.98	0.96
100	0.05	0.56	0.96	0.97
500	0.046	0.54	0.802	0.975
1000	0.049	0.553	0.809	0.934

Table 5. Link prediction for Patent Citation (1988-1990).

SDNE: SDNE struggled with the sparse graph, with Precision@K values below 0.3. These results emphasize its limitations in sparse environments, particularly for link prediction tasks. The AUC score for SDNE on this dataset was 0.47, reflecting its limited ability to distinguish positive and negative links in sparse networks.

Node2vec: Node2vec improved upon SDNE with better Precision@K scores, but its AUC score remained low at 0.52, reflecting its challenges in generalizing to the sparse Patent Citation dataset. Despite some improvements, node2vec's reliance on structural sampling strategies could not fully capture the sparse connections in the data.

PBG: PBG displayed high Precision@K values, indicating strong performance in identifying top-ranked positive links, but its AUC score was 0.5, suggesting that it struggled to distinguish between positive and negative links across the entire range. While it effectively reconstructed sparse graphs, its overall predictive accuracy for link prediction was limited by its inability to generalize well to negative links, resulting in a low AUC score despite the high Precision@K.

GAT: GAT achieved the highest AUC score of 0.95, demonstrating its adaptability to sparse graphs. Using positional encodings significantly improved its link prediction metrics, with Precision@K values consistently outperforming other models.

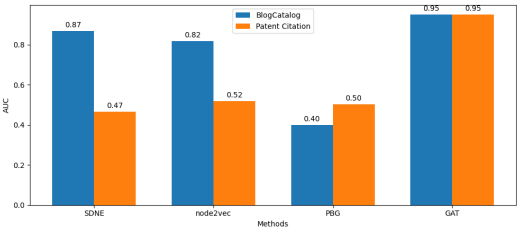

Figure 5. AUC comparison.

The AUC results, summarized in Figure 5, further highlight the strengths of each method across the datasets. While SDNE and node2vec excel in dense graphs, they falter in sparse networks. PBG provides scalable solutions but struggles with overall link prediction accuracy, as indicated by its low AUC score. In contrast, GAT consistently delivers superior results, showcasing its robustness and adaptability to both dense and sparse graph structures.

3.2.3 Negative Sampling Ratio Study

Based on Table 4 and Table 5, GAT model achieved superior precision@k metrics compared to other approaches, but with a near perfect performance across all k values, raising concerns about overfitting in the sparse networks. We hypothesized that the negative sampling ratio is a key driver in this overfitting behavior. Doubling the negative samples to a 2:1 ratio led to more realistic results - while reconstruction performance remained strong (0.9-1.0), prediction metrics decreased to more reasonable levels (0.69-0.79), with AUC dropping from 95% to 90%.

To cross-validate this finding, we conducted sensitivity analysis on negative sampling ratios using node2vec across both the patent and BlogCatalog datasets, choosing node2vec for its lower computational intensity. Results showed the effect of negative sampling ratios varied with graph density. In the dense Blog Catalog graph, Precision@K scores changed minimally, ranging from 0.0–0.1 in reconstruction and 0.0–0.3 in link prediction, showing robustness. In contrast, the sparse Patent Citation graph saw Precision@K drop sharply, from 0.7 to 0.05 in reconstruction and 0.5 to 0.05 in link prediction, as ratios increased from 1 to 25. AUC scores remained stable, averaging around 0.8 for Blog Catalog and 0.5 for Patent Citation. These findings highlight the intricate role of negative sampling.

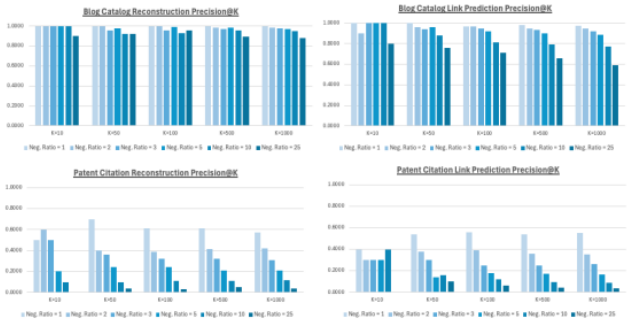

Figure 6. Negative Sample Impact analysis

4. Future work

Scaling these models to manage very large datasets efficiently is a significant challenge due to its complex attention mechanisms and computational demands. Future efforts should focus on optimizing these models through advanced graph partitioning, better parallel processing techniques, or innovative training approaches that reduce computational effort. Adding a broader range of evaluation metrics, such as Normalized Discounted Cumulative Gain (NDCG) [2], would also be beneficial. NDCG is particularly useful for measuring ranking tasks and could provide more detailed insights into how well the models rank important patent citations. Furthermore, improve GAT by developing edge embeddings that directly model interactions between nodes, capturing the unique dynamics of each relationship.

5. Work Division

The delegation of work among team members is provided in Table 6.

Student Name	Contributed Aspects	Details
Yi Liang	Visualization, SDNE	Visualization of the dataset and trained the SDNE on the BlogCatalog and Patent Citation datasets and analyzed the results.
Yongtao Mu	Dataset, node2vec	Prepared negative sampling for training and testing dataset. Trained the node2vec on the BlogCatalog and Patent Citation datasets and analyzed the results.
Jin Yan	Writing, Pytorch Biggraph	Trained the PyTorch Biggraph on the BlogCatalog and Patent Citation datasets and analyzed the results. Organized project and final report writing.
Xiaohu Zhang	Visualization, Graph Attention Network	Trained the Graph Attention Network (GAT) on the BlogCatalog and Patent Citation datasets and analyzed the results. Analyzed effect of evaluation metrics. and visualization

Table 6. Contributions of team members.

References

[1] Samuel Chen, David Dang, Robert Macy, and Chris Rockwell. Link prediction on the patent citation network, 2019. 1, 3, 5

[2] Aparna Dhinakaran. Demystifying NDCG: How to best use this important metric for monitoring ranking models. Online, January 2023. 6

[3] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. https://pyg.org/, 2019. Accessed: 2024-12-07. 3

[4] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 855–864. ACM, Aug. 2016. 1, 2

[5] Bronwyn Hall, Adam Jaffe, and Manuel Trajtenberg. *The NBER Patent Citation Data File: Lessons, Insights and Methodological Tools*. Oct. 2001. 1

[6] Lucas Hu, Thomas Kipf, and Gökçen Eraslan. lucashu1/link-prediction: v0.1: Fb and twitter networks (v0.1). https://github.com/lucashu1/link-prediction?tab=readme-ov-file, 2018. Accessed on 2024-12-07. 3

[7] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. Pytorch-biggraph: A large scale graph embedding system. *Proceedings of Machine Learning and Systems*, 1:120–131, 2019. 1, 2

[8] Tomas Mikolov. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 3781, 2013. 2

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 3

[10] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018. 1, 3

[11] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1225–1234. ACM, Aug. 2016. 1, 4

[12] R. Zafarani and H. Liu. Social computing data repository at ASU, 2009. 1