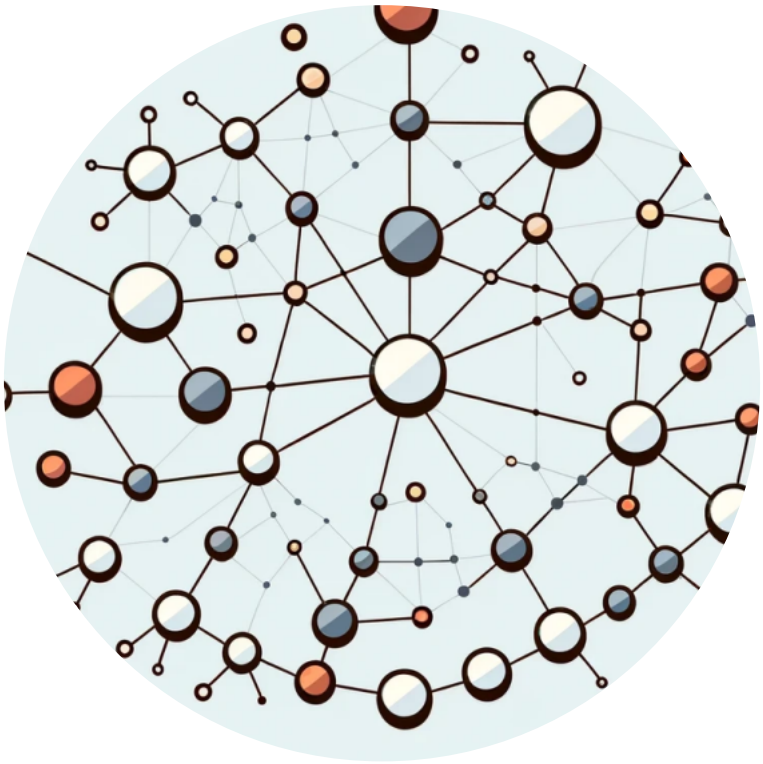
University of Applied Sciences and Arts Northwestern Switzerland
School of Engineering

Link Prediction in Graphs with Graph Neural Networks

Investigating Transfer Learning

Bachelor's Thesis

Brugg-Windisch, August 2024



Students	Thomas Mandelz (FHNW) Jan Zwicky (FHNW)
External Expert	Prof. Dr. Frank-Peter Schilling (ZHAW)
Professional Experts	Stephan Heule (FHNW) Prof. Dr. Daniel Perruchoud (FHNW)
Project Number	24FS_I4DS26

University of Applied Sciences and Arts Northwestern Switzerland, School of Engineering

Abstract

This thesis explores the applications of transfer learning in link prediction using Graph Neural Networks, focusing on enhancing model performance by pre-training Graph Neural Network models and fine-tuning them afterwards. The thesis employs a robust methodology including data preprocessing, exploratory data analysis and model evaluation. Training model architectures such as GCN and NGCN, we investigate the benefits of transfer learning by pre-training models on the ogbn-papers100M dataset and fine-tuning on the ogbn-arxiv dataset.

Our findings reveal significant improvements in model performance, with enhanced mean reciprocal rank compared to reference models, highlighting the potential of transfer learning in predicting links within graph structures. This approach not only accelerates training times but also enhances model performance for the GCN architecture. While these findings could not be fully validated for the NGCN architecture, model training remains faster.

This thesis contributes to a deeper understanding of GNN capabilities and transfer learning's impact, offering insights into their practical applications and optimisation.

Keywords

Graph Neural Networks, Graph Theory, Link Prediction, Transfer Learning, Deep Learning

Acknowledgments

We would like to thank Prof. Dr. Daniel Perruchoud and Stephan Heule, for their guidance and support throughout this research project. Their advice has been helpful in navigating the key aspects of our thesis.

We also appreciate the assistance of Prof. Michael Henninger and Dr. Cédric Huwyler, who kindly answered our questions and gave us a jumpstart in graph theory and graph neural networks, significantly advancing our work.

Lastly, we are grateful to our friends, Chantal, Joseph, Martin and Urs for their help in proofreading this document and offering feedback that improved its readability.

Table of contents

List of Figures

List of Tables

Acronyms

AUROC Area under the Receiver Operating Characteristic curve. 23, 24, 29, 47, 48, 50

CN Common Neighbor algorithm. 4, 15, 18, 19, 48, 51, 53, 64, 65
CV Computer Vision. 8

FC Fully Connected Layer. 21, 22

GC Graph Convolutional Layer. 20
GCN Graph Convolutional Network. 15, 20–22, 28, 30, 33, 34, 36, 37, 50–54
GNN Graph Neural Network. 1, 3, 4, 7–9, 18–20, 26, 27, 48, 49, 51, 54, 71

MAG Microsoft Academic Graph. 10, 11
MRR Mean Reciprocal Rank. 23, 24, 26, 28, 29, 31, 33–35, 37–41, 44–46, 48–51, 54, 61

NGCN Nested Graph Convolutional Network. 22, 28, 50, 52, 53, 61
NLP Natural Language Processing. 8

OGB Open Graph Benchmark. 10, 14, 48, 50

RF Random Forest. 19, 26, 27, 33, 35, 39, 41, 52
ROC Receiver Operating Characteristic. 23

Glossary

fine-tuned model A GNN model which is fine-tuned, meaning it was pretrained on the arXiv w/o CS dataset (3.3) and then fine-tuned on our arXiv CS dataset (3.3). 1, 10, 11, 28, 30, 42, 46, 51, 52, 54, 61

pre-trained model A GNN model which is pre-trained, meaning it was trained on our arXiv w/o CS dataset (3.3). These models weights are in the next step loaded and fine-tuned. 1, 7, 8, 11, 18, 23, 28, 37, 45, 51

reference model A GNN model which is used as a reference to compare a fine-tuned model against. 1, 11, 25, 28, 30, 34, 36, 37, 42–47, 50, 51, 53, 61

1 Introduction

Graph data is characterised by entities (nodes) that are interconnected by relationships (edges or links), forming a complex and non-euclidean structure. A key task in the analysis of graph data is link prediction, which involves predicting the existence of a link between two nodes. This task is critical in numerous real-world applications, including social network analysis, recommendation systems, and biological network inference. Link prediction on such data requires models that can effectively capture the intricate patterns inherent within the graph. Traditional methods often fall short in capturing these complexities, leading to the exploration of more sophisticated approaches such as Graph Neural Networks (GNNs).

GNNs have emerged as powerful tools in the field of machine learning for graph-structured data. Their ability to learn representations that encapsulate both node features and the topology of the graph has led to significant advances in various tasks, including link prediction. However, training GNNs often requires large amounts of labeled data, which may not always be available. This challenge can be addressed by leveraging transfer learning, a technique that involves transferring knowledge from one domain to another to enhance performance on tasks where data is scarce.

This thesis investigates the application of transfer learning to improve link prediction in GNNs. While GNNs and transfer learning have been extensively studied separately, their intersection, particularly in the context of link prediction, remains underexplored. By examining how pre-trained models can be fine-tuned for specific link prediction tasks, this research aims to bridge this gap and provide insights into the potential benefits of combining these approaches.

The scope of this thesis is deliberately focused on the feasibility and effectiveness of transfer learning for link prediction using GNNs, rather than pursuing a broader foundation model approach. The research is grounded in practical applications, specifically utilising citation networks as a case study for link prediction. Two datasets are employed for this purpose: ogbn-arxiv, a smaller dataset consisting of computer science papers, is used for fine-tuning the models, while ogbn-papers100M, a larger dataset encompassing the entire arXiv archive, is used for pre-training. These datasets simulate a realistic scenario where the model predicts whether a paper cites another paper.

This thesis aims to address the following questions:

- Can a GNN achieve a higher mean reciprocal rank (MRR) than a heuristic model?
- Can the benefits of transfer learning be utilised with a pre-trained model to achieve an increase in jumpstart, asymptotic performance or transfer ratio compared to a GNN reference model?
- Are there edges of specific graph structures that a GNN cannot predict?
- Can a GNN trained with transfer learning predict these edges?
- Does an increase in model capacity increase the model performance for transfer learning?

To address these research questions, we begin by conducting a thorough literature review. This review covers the fundamentals of graph data, explores link prediction techniques including heuristic and GNN-based methods, and examines the principles of transfer learning with a particular focus on its application to GNNs. A non-exhaustive summary for this thesis is found in section 2. In section 3 we describe our datasets, preprocessing and data split methodology.

Based on this understanding of the data and the fundamentals we describe our approach in section 4 by establishing used methods and model architectures. The section 5 comprises of quantitative and qualitative results of heuristic based models, reference models and fine-tuned models. They are also compared with one another, and their transferability is demonstrated

using transfer metrics. The comprehensive discussion of the research questions in the context of our results is presented in section 6, while section 7 offers a conclusion of the entire thesis and provides an outlook for furthering our research.

2 Fundamentals

This section serves as an introduction to the fundamental theories and methods that are important for the reader to grasp the subsequent work of this thesis. By providing a foundation, we aim to ensure that readers of all backgrounds can follow the thesis.

Graph theory is a mathematical framework used to model pairwise relations between objects (West, 2001 and Diestel, 2006). At its core, a graph G is defined by a set of nodes N and a set of edges E (also known as links) that connect pairs of nodes. In directed graphs, edges have a direction, indicating a one-way relationship between nodes. In undirected graphs, edges do not have a direction, representing a mutual relationship between nodes.

In graph theory, several key concepts need to be explained for our thesis: Link prediction, Degrees, Self-Loops, Connected components, Clustering coefficient and Reachability. These concepts are detailed in section 2.1.

The general functionality of GNNs is discussed in the following sections, with detailed explanations of the specific architectures we used provided in section 4.4 and section 4.4.

To contextualise our research, we review existing literature on transfer learning and its applications to graph-related tasks. This overview will provide insights into the current state of the research and highlights the gap that our thesis aims to address.

2.1 Graph Theory

A directed graph G is defined as a pair (N, E), where N represents the set of nodes and E represents the set of edges connecting these nodes. Let $N = \{n_1, n_2, n_3, n_4, n_5\}$ be the set of nodes, with n_1, n_2, n_3, n_4, and n_5 being distinct nodes.

Similarly, let $E = \{(n_1, n_2), (n_2, n_3), (n_3, n_1), (n_4, n_5), (n_5, n_4)\}$ be the set of edges, where each edge is represented as an unordered pair of nodes. Then the graph G can be visualised as seen in Figure 2.1.

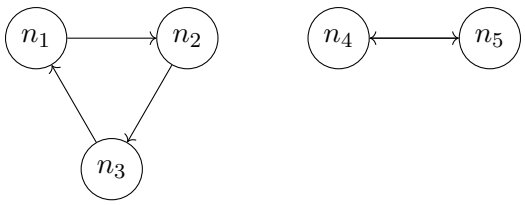


Figure 2.1: A simple directed graph G with five nodes n_1, n_2, n_3, n_4, and n_5. Each node is represented by a circle, and each edge is represented by a line with an arrow between the corresponding nodes.

The adjacency matrix A of a graph G is an $n \times n$ matrix, where n is the number of nodes in the graph. The adjacency matrix of a graph G, denoted as $A = [a_{ij}]$, indicates whether an edge exists from node i to node j (typically with $a_{ij} = 1$ for existing edges and $a_{ij} = 0$ for non-existing edges). This matrix provides a compact representation of the graph's structure and forms the basis for many graph-based analyses and prediction methods.

For the given graph G from Figure 2.1, the adjacency matrix A is as follows:

$$A = \begin{array}{c|ccccc} & n_1 & n_2 & n_3 & n_4 & n_5 \\ \hline n_1 & 0 & 1 & 0 & 0 & 0 \\ n_2 & 0 & 0 & 1 & 0 & 0 \\ n_3 & 1 & 0 & 0 & 0 & 0 \\ n_4 & 0 & 0 & 0 & 0 & 1 \\ n_5 & 0 & 0 & 0 & 1 & 0 \end{array}$$

Link Prediction

In graph theory, there are various prediction tasks such as graph classification, node classification, and link prediction, see Table 2.1 for detailed examples. This thesis focuses exclusively on link prediction, as the combination of transfer learning and link prediction is novel. The concept of link prediction involves predicting the probability of a potential connection between a source and a target node in a graph. A schematic is shown in Figure 2.2.

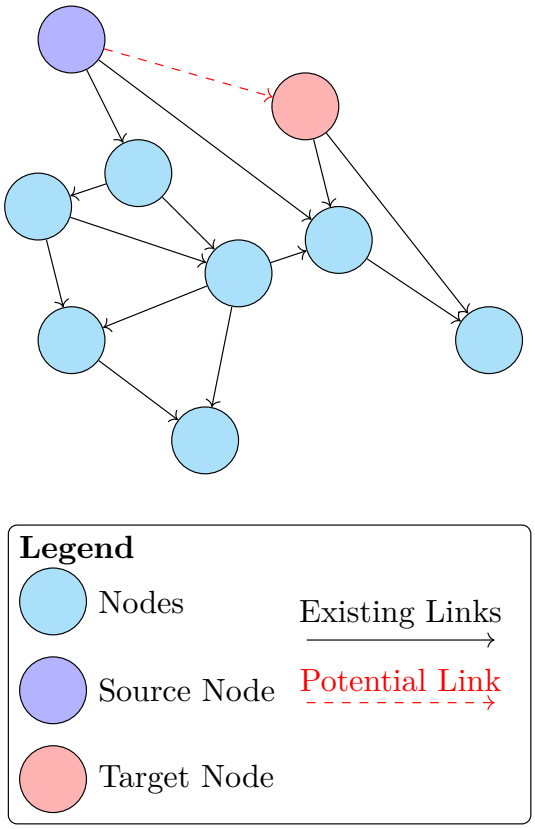


Figure 2.2: Schematic of Link Prediction in a directed graph, The node in blue is a source node, the node in red is a target node, all other nodes are in cyan. The red dashed line indicates a potential link (link to predict). A model predicts the probability of this connection.

Link prediction can be performed using various heuristics such as Common Neighbor algorithm (CN), further explained in section 4.4 or more advanced models such as GNNs.

Degree

In graph theory, the degree of a node is a concept that indicates the number of edges connected to that node. The indegree of a node is the number of edges that enter this node, while the outdegree is the number of edges that leave this node.

Self-Loop

A self-loop, is an edge where starting point and end point are at the same node. In the adjacency matrix $\tilde{A}$ ($\tilde{A}$ is used for the adjacency matrix where self-loops are included) of a graph, self-loops are represented with $a_{ii} = 1$ in the diagonal entries.

Connected Components

In graph theory, the concept of connected components refers to the division of a graph into subsets of nodes.

A directed graph consists of nodes and directed edges indicating a direction from one node to another. In such a graph, we can distinguish between two types of connected components: strong and weak.

Strongly Connected Components

A strongly connected component in a directed graph is a subset of nodes where each node is connected to every other node through directed paths (Nuutila and Soisalon-Soininen, 1994). In other words, there is a directed path from every node to every other node in the same component. An example Graph is shown in Figure 2.3.

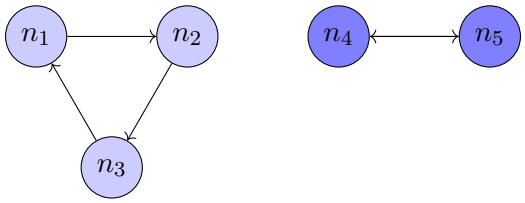


Figure 2.3: The two instances of strongly connected components of graph G highlighted in different shades of blue. $Component_1 = \{n_1, n_2, n_3\}$, $Component_2 = \{n_4, n_5\}$

Weakly Connected Components

A weakly connected component in a directed graph is a subset of nodes where there is an undirected path between every pair of nodes. In other words, if we ignore the directions of the edges, the graph is connected. An example Graph is shown in Figure 2.4.

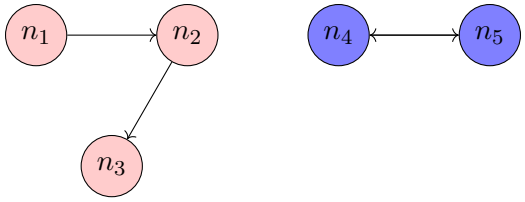


Figure 2.4: Example for a graph with a weakly connected component. The weakly component is highlighted in pink. Here, the edge $e = (n_3, n_1)$ was removed from the graph G. Consequently, not all nodes are reachable along the direction of edges anymore, leading to the $Component_1$ becoming a weakly connected component. $Component_2$ is still a strongly connected component because of the bidirectional edge.

Cluster Coefficent

The cluster coefficient is a measure that quantifies the local cohesion of a graph (Fagiolo, 2007). It represents how strongly the neighbors of a node are connected to each other. The cluster coefficient $C(n)$ of a node n is defined as the ratio of the number of directed triangles T_n through node n to the maximum possible triangles.

For a node n with $k(n)$ neighbors (sum of in-degree and out-degree), there are at most M triangles when all possible combinations of incoming and outgoing edges are present.

$$M = k(n)(k(n) - 1) - 2k_{\leftrightarrow}(n) \tag{2.1}$$

The links denoted by $k_{\leftrightarrow}(n)$ are those that are bidirectional. The cluster coefficient $C(n)$ is then calculated as follows:

$$C(v) = \frac{T_n}{M} \tag{2.2}$$

For the entire graph, the average cluster coefficient C across all nodes is calculated as follows:

$$C = \frac{1}{|N|} \sum_{n \in N} C(n) \tag{2.3}$$

The cluster coefficient is an important metric in graph theory because it provides insights into how strongly nodes in a graph form local groups or clusters. A high cluster coefficient indicates strong local cohesion, while a low cluster coefficient may indicate a more distributed or less interconnected structure.

Reachability

In graph theory, reachability refers to the ability to reach one node from another through a sequence of edges (John and Edmund, 2008). Formally, the reachability of two nodes in a graph is defined by a path, which is a sequence of nodes where each node is directly connected to the next through an edge. The reachability between two nodes u and n is described by a function $R_k(u, n)$, which indicates whether there is a path from u to n using exactly k hops (k edges used to reach from u to n). Mathematically, this function is defined as:

$$R_k(u, n) = \begin{cases} 1 & \text{if there exists a path from } u \text{ to } n \text{ with exactly } k \text{ hops,} \\ 0 & \text{otherwise.} \end{cases} \tag{2.4}$$

A cumulative function representing the number of nodes reachable through increasing neighborhoods can be defined to quantify the reachability of a node u. Specifically, this function,

denoted as $Q_k(u)$, counts the number of nodes n that are reachable from u using exactly k hops. Formally, $Q_k(u)$ is given by:

$$Q_k(u) = |\{n \in N : R_k(u, n) = \text{True}\}| \tag{2.5}$$

Graph Neural Networks

Graph Neural Networks (GNNs) provide an effective method for processing graphs. They enable learning information about the structure and properties of the graph and using it for tasks such as predicting edges.

The most common approach is to use a GNN to generate an embedding representation for each node in the graph (Kipf and Welling, 2017, Hamilton et al., 2018, Xu et al., 2019). These embeddings capture the structural and attribute information of each node. Subsequently, various methods, such as neural networks or classical machine learning algorithms, can be used to predict the probability of a connection between two nodes based on their embeddings.

Mathematically expressed, the prediction of the probability of an edge between two nodes u and n using a GNN can be formulated as follows:

$$P(u, n) = f(E(u), E(n)) \tag{2.6}$$

where $E(u) \in \mathbb{R}^d$ and $E(n) \in \mathbb{R}^d$ represent the embedding vectors of nodes u and n, respectively. Here, $\mathbb{R}^d$ denotes the d-dimensional vector space in which these embeddings are situated. The function f combines the embeddings of the two nodes to predict the probability of the link between them. For further details on graph theory in the context of GNNs, refer to Prince (2023) and for a general overview on GNNs refer to Wu et al. (2021).

2.2 Transfer Learning

Transfer Learning is a concept in machine learning where knowledge learned by solving a specific task (source task) is transferred to a related task (target task). Formally defined by Yang et al. (2020b):

> Given a source domain D_s and learning task T_s, a target domain D_t and learning task T_t, transfer learning aims to help improve the learning of the target predictive function $f_t(\cdot)$ for the target domain using the knowledge in D_s and T_s , where $D_s \neq D_t$ or $T_s \neq T_t$.

Yang et al. (2020b) define various ways to transfer knowledge during transfer learning, including:

- instance-based algorithms
- feature-based algorithms
- **model-based algorithms**
- relation-based algorithms

This thesis primarily explores model-based transfer learning, focusing on fine-tuning pre-trained models. The goal is to leverage knowledge from a source domain model to improve learning in a target domain, typically by sharing components like weights or hyperparameters. Fine-tuning pre-trained weights is preferred due to its efficiency and effectiveness, allowing models to adapt to new tasks while retaining generalised knowledge. Fine-tuning with deep learning models is further described by Yang et al. (2020a).

2.3 Related Work

This thesis investigates the application of transfer learning in link prediction through a series of experiments. To the best of our knowledge, this is the first thesis to evaluate the feasibility of transfer learning specifically in the context of link prediction using GNNs.

While transfer learning has become a state-of-the-art technique in various data modalities and tasks, such as image classification, its potential in link prediction remains unexplored. The following section aims to provide an overview of existing transfer learning research and methodologies in related fields.

We also draw inspiration from recent studies, like Kooverjee et al. (2022), on transfer learning in node and graph classification, which have significantly contributed to the foundation of our thesis. By building upon these efforts, we seek to extend the application of transfer learning to link prediction.

Transfer Learning in Computer Vision and Natural Language Processing

Transfer learning is extensively utilised in both Computer Vision (CV) and Natural Language Processing (NLP). In CV, a common approach involves leveraging a pre-trained model with generalised embedding representations of images, such as ResNet (He et al., 2015), which is initially trained on a large, labelled image classification dataset like ImageNet (Deng et al., 2009). This pre-trained model is then fine-tuned for the specific target task (X. Li et al., 2020, Gupta, 2021, Dąbrowski and Michalik, 2017).

In NLP, the state-of-the-art has evolved significantly. The focus has shifted from learning word embeddings based on the occurrence of words in a corpus, for example GloVe (Brochier et al., 2019), to learning contextualised word embeddings using advanced models such as BERT (Devlin et al., 2019) and GPT (Brown et al., 2020). These models provide deeper understanding and better performance in various downstream NLP tasks by capturing the context in which words appear. These models are then fine-tuned for specific target tasks (Howard and Ruder, 2018, Raffel et al., 2023, Ruder et al., 2019).

The previous examples of transfer learning are not exhaustive and serve as common illustrations. Numerous other techniques, models, and transfer methods also exist and are widely used, an overview can be found in Zhuang et al. (2021).

Transfer Learning in Graph Prediction Tasks

In contrast to the extensive research on transfer learning with pre-trained models in CV and NLP, the field of transfer learning in graph prediction tasks has seen comparatively little exploration. Graph prediction tasks encompass node-level, graph-level, and edge-level predictions. Within these categories, a multitude of tasks can be solved using GNNs, as detailed by Wu et al. (2021). This variety underscores the potential for diverse transfer learning approaches. A non-exhaustive list of these tasks is presented in Table 2.1.

Several studies have assessed transferability of GNNs in node-level and graph-level prediction tasks (Kooverjee et al., 2022, Zhu et al., 2021, W. Hu et al., 2020, Z. Hu et al., 2019, Z. Hu et al., 2020) while edge-level tasks are underexplored, especially for link-prediction there is to our best knowledge no research done in the context of transfer learning with GNNs.

Zhu et al. (2021) presents a framework for transferring knowledge in GNNs, particularly focusing on link-classification as the graph prediction task. It introduces Ego-Graph Information maximisation to capture essential graph information for transferable GNN training, with theoretical

Level	Task
node-level	node-classification
node-level	node-regression
graph-level	graph-classification
graph-level	graph-regression
graph-level	graph-generation
edge-level	**link-prediction**
edge-level	link-classification

Table 2.1: Non-comprehensive list of graph prediction tasks, the most relevant ones per level are mentioned.

analysis and synthetic experiments validating its effectiveness. W. Hu et al. (2020) address the challenge of effectively pre-training GNNs for tasks with distributional differences and scarce labels. They introduce a novel strategy and self-supervised methods for pre-training GNNs at both node and graph levels, enabling simultaneous learning of local and global representations.

Z. Hu et al. (2019) present a pre-training framework for GNNs, emphasising classification tasks for nodes, graphs, and links. This framework, employing denoising link reconstruction, centrality score ranking, and cluster preserving tasks on synthetic graphs, effectively captures transferable graph structural information. By requiring less labeled data and domain-specific features, the pre-trained GNN achieves high performance across various downstream classification tasks. Z. Hu et al. (2020) follow up with the GPT-GNN framework, which leverages generative pre-training to initialise GNNs with self-supervised learning. GPT-GNN introduces a novel self-supervised attributed graph generation task to pre-train GNNs, enabling them to capture both structural and node feature properties of graphs.

Kooverjee et al. (2022) explore transfer learning in the context of graph and node classification. The research demonstrates that transfer learning is indeed effective with GNNs for their tasks. They highlight the importance of GNN architecture selection for transferability. Real-world and synthetic data are used to evaluate node and graph classification tasks, with a proposed methodology for transfer learning experimentation and a novel algorithm for synthetic task generation. Their findings indicate that GNNs significantly improve transfer, particularly when source and target tasks share community structure.

3 Data & Data Analysis

In this section, we detail the datasets used in our research, along with the methods employed for data preprocessing and the specific methodologies for data splitting to facilitate accurate model evaluation. Additionally, we show our explorative analyses for the datasets.

3.1 Datasets

For our thesis, we are using the Open Graph Benchmark (OGB) provided by W. Hu et al. (2021). A simple overview is found in Table 3.1, with further details provided in the subsequent sections.

Dataset	Domain	Nodes	Edges	Source
ogbl-citation2	Papers (MAG subset)	2'927'963	30'561'187	Link
ogbn-arxiv	Papers (arXiv CS)	169'343	1'166'243	Link
ogbn-papers100M	Papers (MAG subset)	111'059'956	1'615'685'872	Link

Table 3.1: Dataset overview for the original unprocessed datasets, detailed information is provided in OGB datasets. Microsoft Academic Graph (MAG) is later described in section 3.1. arXiv CS stands for the arXiv Computer Science category.

Open Graph Benchmarks

OGB is a collection of diverse and large-scale graph datasets designed to facilitate research in the field of machine learning with graphs. These datasets encompass a wide range of domains (citation networks, blood vessels, etc.) and tasks (link prediction as well as node and graph classification), offering researchers standardised benchmarks for evaluating and comparing model performance. This is particularly useful for our thesis, as it allows us to obtain model architectures and their metrics to validate our trained models and their performance. Additionally, the varying sizes of the datasets allow for the utilisation of large datasets for pre-training and smaller ones for fine-tuning within the same domain, thereby enabling straightforward implementation of transfer learning strategies.

Microsoft Academic Graph

The OGB datasets are based on subsets of the Microsoft Academic Graph (MAG), which presents unique challenges to our thesis. One significant challenge is data leakage, particularly in the context of transfer learning. The main way data leakage occurs for our research is when information from the pre-training phase inadvertently influences the performance of the fine-tuned model, potentially compromising its performance by overestimating it.

ogbl-citation2

The ogbl-citation2 dataset is a directed graph representing paper citations. Each node is a paper with 128 dimensional Word2Vec features from its title and abstract (Mikolov et al., 2013). Additionally, the publication year is included as a feature, but no other features or identification attributes (title, citation identifiers like DOI numbers) are available, making it challenging to identify a paper. Directed edges denote citations. Papers are split by publication year for citation recommendation simulation, with 2019 papers as sources. Two papers from each source's references are omitted for validation and testing, and the rest are for training.

The ogbl-citation2 dataset serves as a validation tool for our pipeline. Notably, this dataset includes a benchmark leaderboard[1] for link prediction, where our chosen architectures have already achieved rankings. Reproducing these benchmark results validates the integrity of our training pipeline and model implementation.

ogbn-arxiv

The ogbn-arxiv dataset is a directed graph of Computer Science arXiv paper citations from MAG. Each node represents a paper, with edges indicating citations. The papers have 128 dimensional Word2Vec feature vectors (Mikolov et al., 2013) derived from title and abstract. Additionally, the publication year is included as a feature, but no other features or identification attributes (title, citation identifiers like DOI numbers) are available, making it challenging to identify a paper. The benchmark leaderboard and dataset is set up for node classification. The dataset splitting follows publication dates, adding all edges of papers published until 2017 for training, those from 2018 for validation, and those from 2019 onward for testing. To ensure maximal utilisation of available training data, the data splitting approach from ogbl-citation2 is implemented. For detailed insights into the data splitting methodology, please refer to the dedicated section 3.2.

This dataset serves as our target domain. The fine-tuning and reference models will be trained on this dataset.

ogbn-papers100M

The ogbn-papers100M dataset is a vast directed citation graph containing 111 million papers, indexed by MAG. It shares its graph structure and node features construction method with ogbn-arxiv. Approximately 1.5 million of its nodes represent arXiv papers, many of them labeled with a research field from arXiv.

Original dataset splitting follows a time-based strategy akin to ogbn-arxiv. The data splitting approach adopted from ogbl-citation2 is implemented. For detailed insights into the data splitting methodology, please refer to the dedicated section 3.2.

This dataset serves as our source domain. The pre-train models will be trained on this dataset.

3.2 Preprocessing

The following section describes the preprocessing of our datasets, including the reduction and the detailed implementation of the datasplit.

Reduction

To fully ensure the prevention of data leakage from our pre-trained models to the fine-tuned models, we have taken steps to refine the ogbn-papers100M dataset by removing nodes that could potentially cause information leakage.

Firstly, we eliminated all nodes categorised as 'arXiv Computer Science'[2], as these nodes are highly likely to overlap with those in the ogbn-arxiv dataset. This reduction removed about 0.2% of nodes and 0.05% of edges. Secondly, we removed all papers that were not categorised under any arXiv label. This additional measure was implemented to minimise the risk of data

[1] Leaderboard for ogbl-citation2 dataset benchmark, including models and their implementation.
[2] arXiv computer science category

leakage, given that we cannot fully categorise these instances and, therefore, cannot guarantee they are not already present in the arXiv CS category. As a beneficial side effect, this also significantly reduced the dataset size, which was advantageous given our constraints on GPU memory. This second reduction removed about 98.81% of nodes and 99.21% of edges, ensuring that the pre-training dataset consists exclusively of arXiv papers, with the arXiv CS category fully excluded.

The number of nodes decreased from 111'059'956 to 1'324'684, marking a reduction of approximately 99%. Similarly, the number of edges dropped from 1'615'685'872 to 12'823'767, also reflecting a 99% decrease. Details of these reductions are outlined in Table 3.2.

Reductions	**Node count**	**change**	**Edge count**	**change**
ogbn-papers100M	111'059'956	-	1'615'685'872	-
Remove arXiv CS category	110'837'858	-0.2 %	1'614'808'211	-0.05 %
Remove no arXiv category	1'324'684	-98.81 %	12'823'767	-99.21 %
ogbn-papers100M reduced	1'324'684	-	12'823'767	-

Table 3.2: Dataset filters for reductions, each stage shows a new node and edge count as well as the change in % in relation to the previous line. The last line is only a renaming, we call the dataset that we reduced in two steps 'ogbn-papers100M reduced'.

Datasplit

We followed the splitting methodology of the ogbl-citation2 for our other datasets. This approach was applied to both datasets, the ogbn-arxiv and ogbn-papers100M (after the reduction).

To simulate a practical scenario in citation recommendation, as described by ogbl-citation2[3] , we split the edges based on publication time. Imagine a user writing a new paper who has already cited various existing papers but seeks recommendations for additional references. The most recent papers (published in 2019 for ogbn-arxiv and 2018 respectively for ogbn-papers100M) are designated as the source papers from which we aim to suggest references. For each source paper, we omit two papers from its list of references. These omitted papers form two edges, one for validation and one for testing, pointing from the source paper to the omitted ones. The remaining edges constitute the training data. If a paper has less than two edges they are put into the trainset. A schematic of the splitting methodology is provided in Figure 3.1.

[3]see Dataset splitting in ogbl-citation2 Methodology

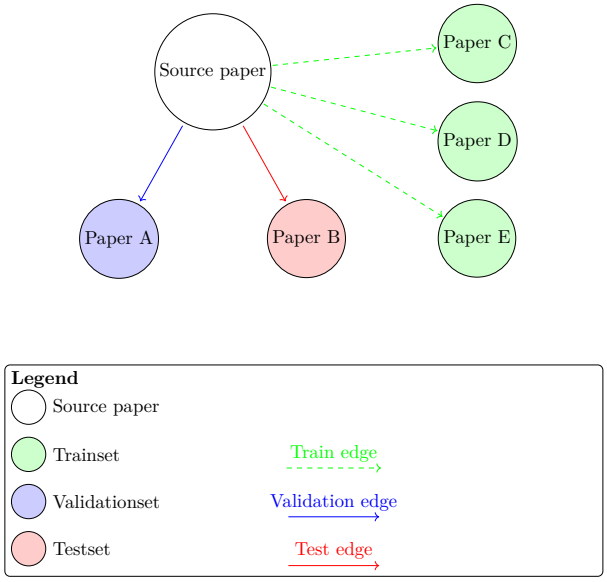


Figure 3.1: Schematic of datasplit, for all papers in the evaluation years (e.g. 2019 for ogbn-arxiv). Per paper, one edge is put into the validation set (blue) and a second edge is put into the test set (red). All remaining edges are put into the trainset (green).

Additionally, 1000 negative samples are drawn for the validation and test splits, details are found in section 4.7. A schematic is shown in Figure 3.2.

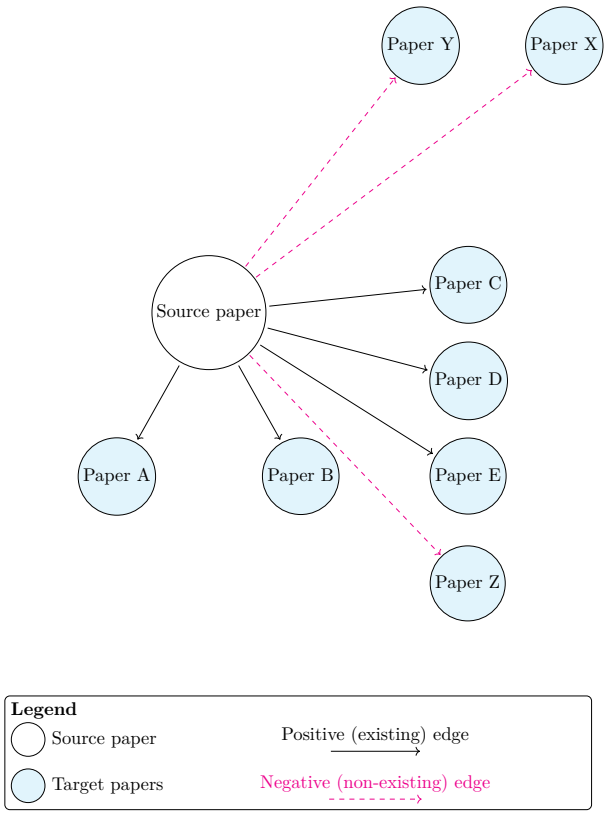


Figure 3.2: Schematic of negative edge sampling, 1000 negative edges per paper are sampled which do not exist. In the schematic these are magenta-dashed edges, the existing edges are in black.

The data split resulted in the distributions presented in Table 3.3 and Table 3.4 for both datasets, ogbn-arxiv and ogbn-papers100M.

Split	Node count	Edge count	Negative Sampling
Training	158'191	1'143'939	At training time
Training Evaluation	11'152	11'152	$11'152 \times 1000$
Validation	11'152	11'152	$11'152 \times 1000$
Test	11'152	11'152	$11'152 \times 1000$

Table 3.3: Final arXiv CS (ogbn-arxiv) dataset splits for the training, training evaluation (metrics calculation only), validation and test set. Training evaluation, validation and test nodes are the same, from each node an edge per split was sampled. Negative sampling for training is done at training time. Note that the edges in the training evaluation are also included in the training edges.

Split	Node count	Edge count	Negative Sampling
Training	1'248'020	12'747'103	At Training time
Training Evaluation	76'664	76'664	$76'664 \times 1000$
Validation	76'664	76'664	$76'664 \times 1000$

Table 3.4: Final arXiv w/o CS (ogbn-papers100M) dataset splits for the training, training evaluation (metrics calculation only) and validation. For the ogbn-papers100M dataset, no test set was sampled because it is used for pre-training. Negative sampling for training is done at training time. Note that the edges in the training evaluation are also included in the training edges.

3.3 Summary of Datasets

While our datasets are based upon OGB datasets, we are subsequently naming them arXiv CS (ogbn-arxiv) and arXiv w/o CS (ogbn-papers100M reduced) after their preprocessing. In Table 3.5 the final datasets are summarised with their respective node and edge counts. The different datasets are assigned to different uses e.g. for the pre-training, fine-tuning and reproduction of models.

Dataset	Usage	Nodes	Edges
ogbl-citation2	Reproduction	2'927'963	30'561'187
arXiv CS	Reference & Fine-tuning	169'343	1'166'243
arXiv w/o CS	Pre-training	1'324'684	12'823'767

Table 3.5: Final dataset overview with our own naming convention, their use in our thesis and the node and edge counts.

3.4 Exploratory Data Analysis

Exploratory data analysis (EDA) is conducted to provide an initial assessment of the number of layers (3.4). Additionally, the dataset is validated, and potential difficulties for transfer learning are estimated. In this section, the focus has been placed on the analysis of reachability (3.4) and strongly connected components (3.4).

The detailed EDA is presented in Appendix C. A comprehensive investigation of the dataset's structure and the distributional characteristics of the baseline heuristics has been conducted. This analysis has revealed significant discrepancies between the datasets that could potentially influence the performance of the models. Below are the most critical findings.

Key Differences in Dataset Structures

Discrepancies between the datasets have been identified by examining strongly connected components and clustering coefficients. Specifically, it was found that the arXiv CS dataset has a higher average clustering coefficient (0.11) compared to the arXiv w/o CS dataset (0.08). This suggests that papers within the Computer Science field are more closely interconnected. This could be because arXiv w/o CS covers a variety of topics that are rarely linked to each other, whereas arXiv CS is more thematically cohesive, leading to a higher degree of interconnection. Additionally, the arXiv w/o CS dataset includes over 150'000 nodes with no links, indicating the presence of noise, which could impact model training. Since these nodes can be sampled as target nodes but are never actually cited.

Further analysis of the indegree distribution supports these observations. While the outdegree remains consistently low in both datasets, differences in the indegree distribution show how certain papers have a greater influence on others within each dataset. These structural differences could present challenges when applying transfer learning between the datasets.

Baseline Model Verification

To establish a heuristic baseline for subsequent model evaluations, we applied the Common Neighbor algorithm and analysed cosine similarity between papers. The results reveal significant differences in the distributions of common neighbors (C.4) and cosine similarities (C.5) between randomly sampled links and actual links. These findings confirm that both methods are effective baseline models, as they exhibit distinct differences in the distributions. However, the large proportion of pairs with very few common neighbors suggests that the Common Neighbor algorithm model might underperform.

Number of Layers & Reachability

This analysis examines node reachability, as detailed in section 2.1, to understand the structure of the underlying networks and to determine the optimal number of layers for a Graph Convolutional Network (GCN) (4.4).

The number of layers in a GCN is important for prediction accuracy. More layers enable the propagation of information over greater distances within the network. Identifying the optimal number of convolutional layers (k in k-hops is equivalent to number of layers) is important for achieving the best performance, and this choice depends on node reachability. The influence of each individual paper diminishes as more papers are reached, raising the question of how many cited papers actually influence the process of a citation and therefore should be considered for the models.

To the best of our knowledge, there is no empirical evidence for choosing an optimal neighborhood (k-hops) for a GCN. Consequently, we adopted an arbitrary threshold of a maximum of 500 papers to be reached by a prediction. This value can be adjusted based on the specific context or could be seen as a hyperparameter. Our goal is to provide an initial estimate of the number of hops to use in a GCN, under the assumption that the majority of papers are influenced by fewer than 500 other papers.

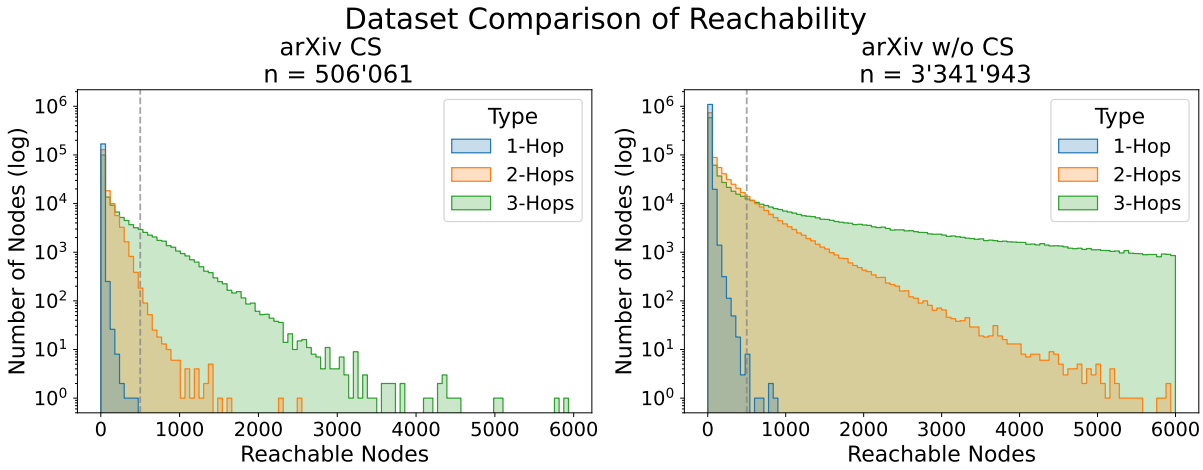


Figure 3.3: Reachability in the arXiv datasets. The colors represent different hop distances: blue for 1-hop, orange for 2-hops, and green for 3-hops. Nodes are counted only once for each k-hop. Our arbitrary threshold of 500 papers is marked with a dashed gray line. Be advised the y-axis is logarithmic scaled. The right side of the plot is cut of at 6000 on the x-axis to make the visual comparison simpler. The uncut plot can be found in Figure C.6.

The absolute reachability of the two graphs differs significantly, as shown in Figure 3.3.

We use the 95% quantile for comparison, as it includes the majority of nodes while excluding outliers that are further away.

In the arXiv CS graph, a 1-hop reach extends to only 24 nodes, while in the arXiv w/o CS graph, it reaches 42 nodes. With a 2-hop, this expands to 214 and 902 nodes, respectively. For the 3-hop, both graphs exceed the threshold of 500 nodes, with the arXiv CS graph reaching 708 nodes and the arXiv w/o CS graph reaching 5,946 nodes. The complete values are shown in Table 3.6.

k-hop	95% quantile (arXiv CS)	95% quantile (arXiv w/o CS)
1-hop	24	42
2-hop	214	902
3-hop	708	5946

Table 3.6: Reachability Comparison Across Different k-hops and for arXiv CS and arXiv w/o CS dataset. 95% quantiles are shown for comparison. The quantiles are calculated from the distributions in Figure 3.3.

Based on a threshold of 500 nodes, the optimal number of hops in the arXiv CS dataset is two, as a 1-hop reach is too limited, and a 3-hop reach includes too many nodes. A 2-hop reach, covering 214 nodes, offers a good compromise.

In contrast, in the arXiv w/o CS dataset, the optimal number of hops is only one, as a 2-hop reach already extends to about 902 nodes, which is far over the threshold. This discrepancy should be considered in model development to ensure optimal performance.

Connected Components

In this analysis, we examine the size of the strongly connected components as described in section 2.1. Due to the temporal nature of paper publications, where a new paper can only

cite previous works and not vice versa, we anticipate that there will be relatively few strongly connected components.

However, the presence of a large number of strongly connected components in both datasets is particularly noteworthy, as illustrated in Figure 3.4. The characteristics of the connected components in the arXiv CS dataset will be investigated further in the next step.

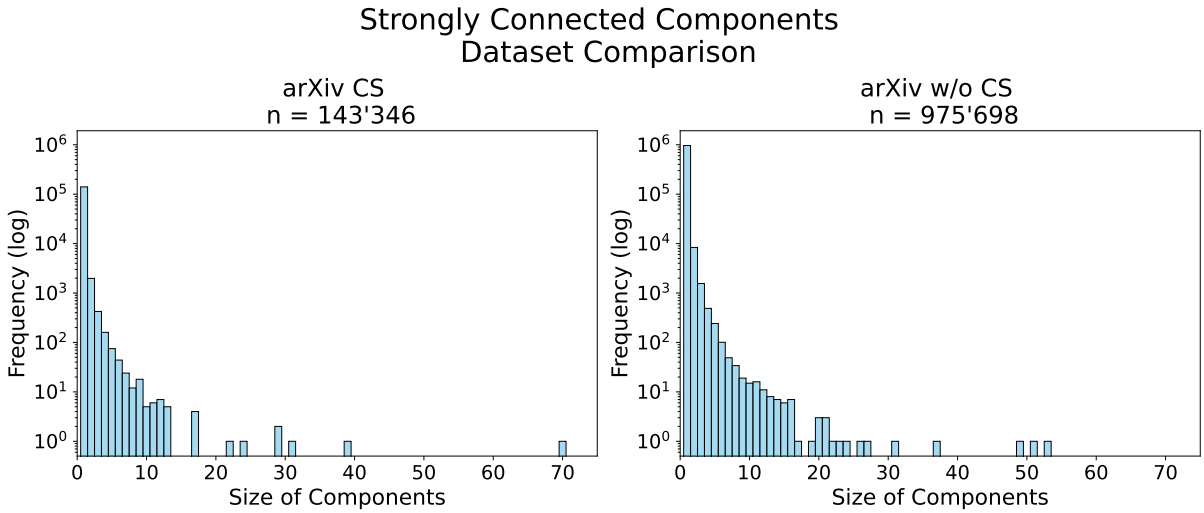


Figure 3.4: Visualisation of strongly connected components in the arXiv datasets. Note: For better readability we removed the largest strongly connected component in arXiv CS with size 21'506 and the largest strongly connected component arXiv w/o CS with size 332'616.

The existence of multiple strongly connected components is plausible due to papers sharing the same publication year, often published in the same conference or through collaboration. However, as shown in Figure 3.5, there are also strongly connected components that appear implausible due to the significant difference in publication years. Another plausible explanation may be that papers which were revised at a certain point may include more references while keeping their original publication year. The bidirectional citation gets more implausible the bigger the difference in publication years. Since this is a benchmark dataset, it is assumed to be correctly constructed. Nevertheless, for a complete and formal understanding further investigation is required. At the very least, it should be verified that the model performs similarly well when a node is within a strongly connected component, to ensure that there is no impact on the modeling.

Figure 3.5: Sketch of a subgraph visualisation of an implausible strongly connected component. The node with year 2013 may cite the node with year 1999, but vice versa is implausible because of the temporal nature of citations.

4 Methods

In this section, we present the methodologies employed throughout our research. This includes an explanation of the infrastructure used, the framework for training and evaluating GNNs, the models implemented, hyperparameter tuning strategies, transfer learning approaches, and evaluation metrics. By systematically addressing each aspect, we aim to provide a comprehensive understanding of the processes involved in our thesis.

4.1 Infrastructure

We utilised multiple GPU infrastructures due to the varying sizes of graph datasets, as well as the availability and cost of GPUs. Detailed descriptions of each setup can be found in the tables (A.1, A.2, A.3) in the appendix.

4.2 Libraries

In this section, the most important libraries for training, evaluating models, and managing data are presented.

PyTorch serves as the backbone of the framework, offering flexibility and efficiency in building and training neural networks (Paszke et al., 2019) . Additionally, model saving and loading are critical for enabling transfer learning, where a pre-trained model is fine-tuned on a new dataset. This framework includes functionalities for saving model states after training and loading them for further training or evaluation.

NetworkX serves as a key component of the framework, providing robust tools for exploratory data analysis, processing of graph datasets and network metrics for the evaluation (NetworkX developers, 2023).

For the implementation of the Common Neighbor algorithm, the repository by Muhan Zhang et al. (2024) was adapted. The GNN implementations were based on the repository by Weihua Hu and Mathias Fey (2024). Both repositories were adapted to fit the evaluation methods and datasets used in this thesis. Additionally, Comet-ML was used for tracking experiments (Comet ML, 2024).

4.3 GNN Model Training

In this section, the structure of training for the GNN models is explained.

The input consists of the adjacency matrix A $(n \times n)$, a matrix containing the features of the nodes X $(p \times n)$ where p is the number of features, vectors E_{source} $(b \times w)$ with the selected source nodes and their embeddings where b is the batch size and w is the activation layer size from the convolutional layers, target nodes E_{target} $(b \times w)$ that exist, and 1000 randomly sampled negative nodes per actual node in the batch $(1000 \times b \times w)$. The fundamentals and feature descriptions are found in section 2.1 and section 3.1 respectively.

The negative samples are generated by randomly selecting nodes, which helps the model learn to distinguish between actual connections and random node pairs. This method improves the robustness and generalisation of the model by exposing it to a wide variety of potential non-links.

These inputs are fed into a GNN and the output consists of the probabilities for both the positive and random samples, indicating the likelihood that a link actually exists.

For the loss function, Negative Log Likelihood (NLL) loss is chosen instead of the traditional Cross-Entropy loss. The choice of NLL loss is motivated by its suitability for binary classification problems where probabilities are predicted directly. The NLL loss is calculated as follows:

$$\text{NLL Loss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)]$$

where N is the number of nodes, y_i is the true label (1 for positive links and 0 for negative links), and p_i is the predicted probability for the corresponding sample.

4.4 Heuristics and Models

This section provides a detailed description of heuristics and model architectures used. Each architecture will be examined in detail in the subsequent sections. The heuristics serve as baselines for comparison with the GNN models.

Cosine Similarity

Cosine similarity is a metric used to measure the similarity between two vectors. It calculates the cosine of the angle between two vectors, providing a value between -1 and 1. The cosine similarity between nodes b and c is defined as:

$$\text{Cosine Similarity}(b, c) = \frac{b \cdot c}{\|b\|\|c\|} \tag{4.1}$$

We use it as a feature-based heuristic. To calculate the ranking metrics, the cosine similarities are ranked in order of magnitude.

Common Neighbors

Common Neighbor algorithm (CN) is a simple graph structure based heuristic used to measure the similarity between two nodes in a network based on the number of common target nodes they share. The common neighbors between nodes b and c is defined as:

$$\text{CN}(b, c) = |N(b) \cap N(c)| \tag{4.2}$$

In this variant of the common neighbors heuristic for directed graphs, $N(b)$ is the set of successors of node b and $N(c)$ is the set of successors of node c. To calculate the ranking metrics, the count of neighbors are ranked in order of magnitude.

Random Forest

Random Forest (RF) stands as a versatile ensemble learning model widely applied in classification tasks (Louppe, 2015; Breiman, 2001). The Random Forest model is used to enhance predictions through graph features, helping to identify which graph features are missing from the GNNs and could improve their predictive performance. A detailed explanation of how it is applied and which features are used can be found in section 5.3.

Graph Convolutional Network (GCN)

A GCN, first introduced by Kipf and Welling (2017), consists of several components, as illustrated in Figure 4.1.

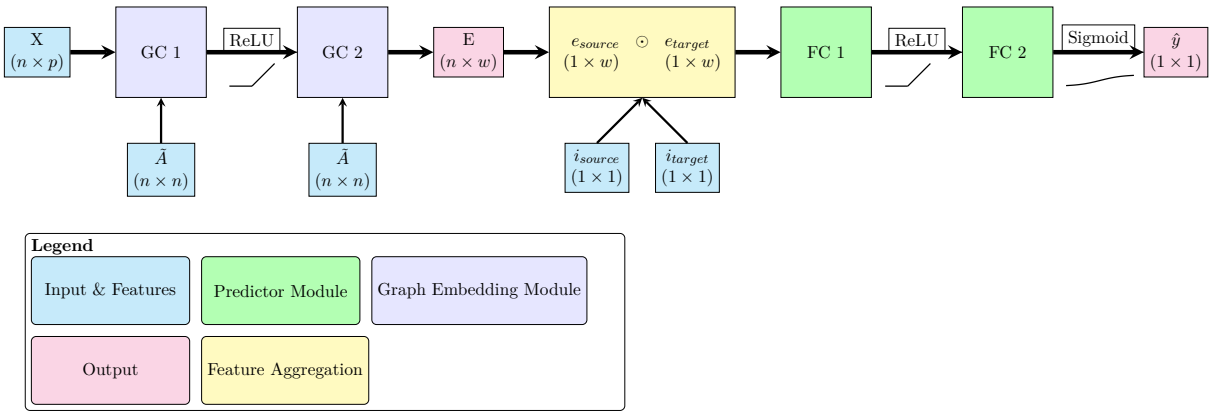


Figure 4.1: Schematic of a GCN architecture with two GC layers, $\tilde{A}$ $(n \times n)$ represents the adjacency matrix of the nodes with added self-loops, X $(n \times p)$ represents the node feature matrix, $E(n \times w)$ is the trained embedding matrix of the Graph Convolutional Layers where w stands for the activation layer size of the graph convolution, i_{source} and i_{target} stands for the indexes of the source and target nodes which filter the embeddings to use as input for the prediction module, e_{source} $(1 \times w)$ and e_{target} $(1 \times w)$ are the embeddings of the selected target and source nodes, $\odot$ is a piece-wise multiplication of the source and target embeddings and $\hat{y}$ is the prediction probability for a link between i_{source} and i_{target} of the predictor module.

In the first step, the node's representation is updated with a Graph Convolutional Layer (GC) which is shown in a schematic in Figure 4.2.

The key idea of a GC is to aggregate feature information from a node's neighbors to update the node's embedding representation. All nodes and edges in the graph are used at once to perform forward and backward passes.

The GCN was chosen because it is a relatively simple model within the realm of GNNs. This simplicity allows for more detailed analysis and easier implementation, making it a suitable choice for the purposes of this thesis.

The input layer GC 1 takes the node features X and the adjacency matrix A of the graph. Hidden layers consist of one or more graph convolutional layers. In our implementation, the hidden channel size is kept consistent across all layers, including the fully connected layers. Each layer performs the aggregation and transformation of node features. The output of each layer l is passed to the next layer $l+1$. The output layer GC 2 produces the final node representations.

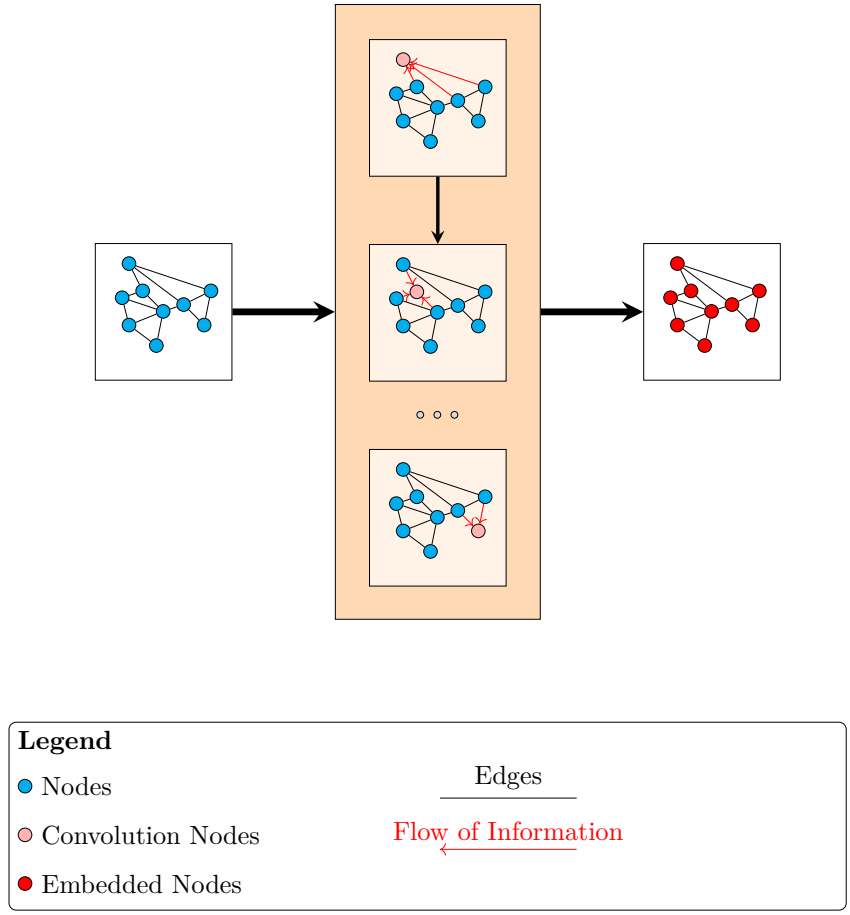


Figure 4.2: Schematic of a GC, each node is embedded separately, red arrows show the flow of feature information into the models convolution from the outgoing links.

The formula for a Graph Convolutional Layer is given by:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)} + b^{(l)}\right) \tag{4.3}$$

where:

- $H^{(l)}$ is the matrix of activations in the l-th layer or the node features X for the first layer,
- $\tilde{A}$ is the adjacency matrix of the graph with added self-loops,
- $\tilde{D}$ is the degree matrix corresponding to $\tilde{A}$,
- $W^{(l)}$ is the weight matrix for the l-th layer,
- $b^{(l)}$ is the bias vector for layer l,
- σ is an activation function (e.g., ReLU).

GCNs can be computationally intensive and require significant memory, especially for large graphs. This is because the entire graph's adjacency matrix and feature matrix must be loaded into memory.

The final node representations are calculated for all nodes in the graph. In the next step, the representations of the source and target node is selected. These are two vectors that are then piece-wise multiplied and fed into a classifier (e.g., a Fully Connected Layer (FC)) for the actual link prediction.

Nested Graph Convolutional Network (NGCN)

The key difference between a Nested Graph Convolutional Network (NGCN) (Zhang and Li, 2021) and a standard GCN is the inclusion of FC to compute the node embedding representations. This structure is illustrated in Figure 4.3.

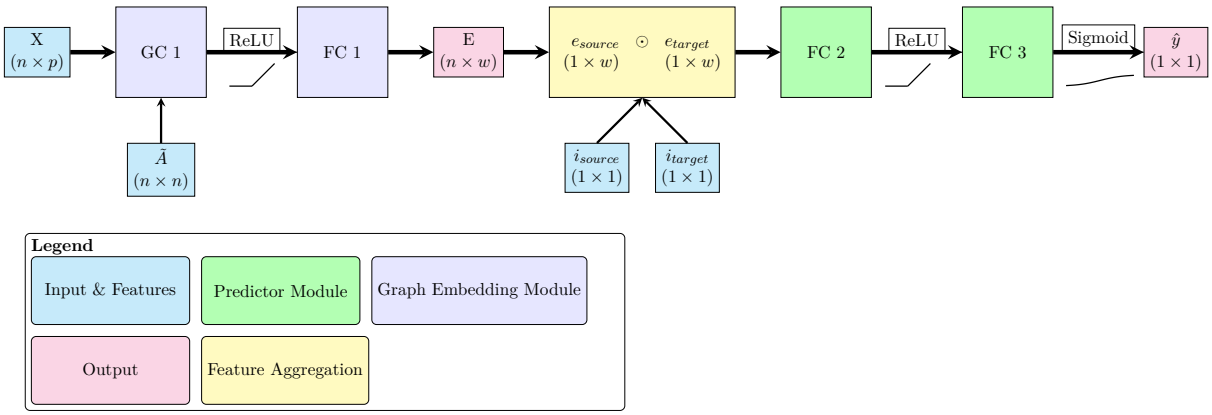


Figure 4.3: Schematic of a NGCN architecture with one GC layer and a fully connected layer after it, only one Graph embedding block (GC 1 and FC 1) is visualised for simplicity, normally two or more blocks are used. $\tilde{A}$ $(n \times n)$ represents the adjacency matrix of the nodes with added self-loops, X $(n \times p)$ represents the node feature matrix, $E(n \times w)$ is the trained embedding matrix of the Graph Convolutional Layers where w stands for the activation layer size of the graph convolution, i_{source} and i_{target} stands for the indexes of the source and target nodes which filter the embeddings to use as input for the prediction module, e_{source} $(1 \times w)$ and e_{target} $(1 \times w)$ are the embeddings of the selected target and source nodes, $\odot$ is a piece-wise multiplication of the source and target embeddings and $\hat{y}$ is the prediction probability for a link between i_{source} and i_{target} of the predictor module.

With the addition of Fully Connected Layers (FC 1) in the graph embedding module, this architecture facilitates more complex transformations and combinations of node features, thereby enhancing the model's capacity. This improvement is achieved without needing access to additional nodes that would otherwise be reached through more hops.

4.5 Hyperparameter Tuning

For hyperparameter tuning, Random Search is selected (Bergstra and Bengio, 2012). The goal is to identify the hyperparameters that enable the model to achieve the best possible value on the selected metric using the validation set.

We utilised the hyperparameter optimisation implementation provided by comet-ml (Pykes, 2022).

For the learning rate and batch size, we applied the linear scaling parameter, which corresponds to sampling from a uniform distribution between the specified minimum and maximum values.

Regarding the hidden channel size and the number of layers, we employed the categorical parameter. This method restricts the search to predefined discrete values, such as 512, 384, 256, 128 for hidden channel sizes.

Random Search was chosen over Grid Search due to the extensive hyperparameter space and the computational intensity of each model evaluation (Bergstra and Bengio, 2012). Due to constraints on GPU availability, we were unable to use a fixed number of iterations. Instead, we varied the number of iterations depending on the computational time required for different

models. Some models underwent more iterations while others underwent fewer, based on the available computing resources and model complexity.

4.6 Transfer Learning

In this thesis, the model is first pre-trained on a larger dataset, the arXiv w/o CS. After this, transfer learning is applied by initialising the model with the parameters from the pre-trained model. Subsequently, the model was optimised for the target task. In some experiments, the feature embedding layers of the model were frozen, meaning they were not optimised further for the target task, allowing us to preserve the pre-trained knowledge in those layers while fine-tuning the rest of the model. The feature embedding layers are frozen because they have already learned to recognise certain patterns, which do not require further targeted learning.

4.7 Evaluation

This section outlines the methods and metrics used to evaluate model performance and transfer learning effectiveness.

Negative Sampling

For the evaluation each of the positive links is evaluated against 1000 randomly selected links. To conserve computing resources, no constraints are imposed on the randomly selected links, except that actual existing links cannot be selected as negative samples. Consequently, these negative samples may include self-loops.

Metrics

To evaluate the performance of a model, two primary metrics are utilised: Area under the Receiver Operating Characteristic curve (AUROC) and Mean Reciprocal Rank (MRR). To evaluate transfer learning abilities, jump start, asymptotic performance and transfer ratio are utilised as presented by Taylor and Stone (2009).

AUROC: Area under the Receiver Operating Characteristic curve (AUROC) is a key metric derived from the Receiver Operating Characteristic (ROC) curve, which is a graphical plot illustrating the ability of a binary classifier system (Bradley, 1997). The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various thresholds. The true positive rate (TPR) measures the proportion of actual positives correctly identified by the model, while the false positive rate (FPR) measures the proportion of actual negatives incorrectly identified as positives. These rates are defined mathematically as follows:

$$\text{True Positive Rate (TPR)} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}} \tag{4.4}$$

$$\text{False Positive Rate (FPR)} = \frac{\text{False Positives (FP)}}{\text{False Positives (FP)} + \text{True Negatives (TN)}} \tag{4.5}$$

The AUROC quantifies the overall ability of the model to distinguish between positive and negative classes. An AUROC of 1 indicates perfect classification, while an AUROC of 0.5 suggests no predictive power. This metric can be used to evaluate whether a certain node (e.g., a paper) is of interest or definitively not (binary classification).

MRR: The MRR is used to rank the actual link (ground truth) against the 1000 negative links. This metric helps evaluate which additional nodes (e.g. papers) might also be of interest if there are several papers of interest. MRR is defined as:

$$\text{MRR} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\text{rank}_i} \tag{4.6}$$

where:

- N is the number of links evaluated.
- rank_i is the rank position of the actual link for the i-th link.

When cross-validation is performed or a model is initialised multiple times with random initialisations, the MRR is reported as $\mu = \textit{mean of multiple MRRs} \pm \textit{one standard deviation.}$

Transfer Learning Metrics: The transfer learning metrics, as illustrated in Figure 4.4, selected for this thesis are aligned with those used by Kooverjee et al. (2022). According to Kooverjee et al. (2022):

> The Transfer Ratio is the ratio of the total cumulative performance of the transfer learner to the base learner, while jumpstart is the initial performance improvement by the transfer over the base learner. Asymptotic Performance refers to the improvement made in the final learnt performance in the target task.

The value ranges for these metrics are as follows:

- Jumpstart: [-1,1] (For AUROC or MRR as calculation metric)
- Asymptotic Performance: [-1,1] (For AUROC or MRR as calculation metric)
- Transfer Ratio: $[-\infty, \infty]$

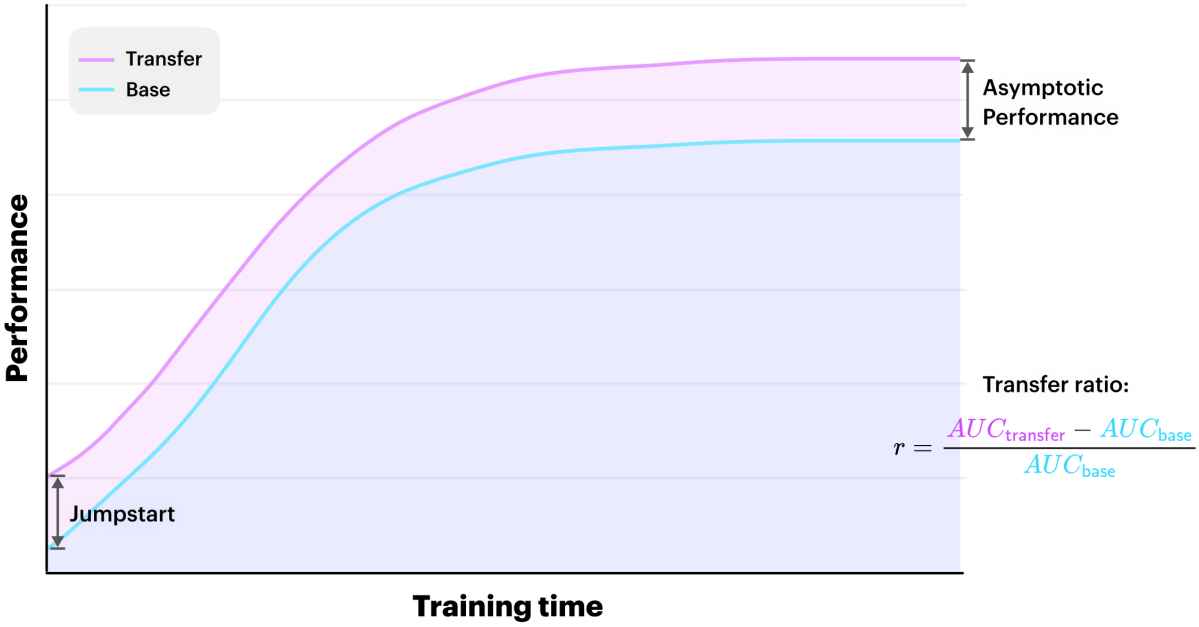


Figure 4.4: Transfer learning metrics schematic, describing transfer metrics in a hypothetical training process. Schematic was created by Kooverjee et al. (2022). AUC refers to the AUROC metric. 'Base' refers to the reference model in our thesis.

These metrics provide a comprehensive evaluation of the benefits of transfer learning, encompassing both initial gains and final improvements. In our thesis, the base model is named reference model.

Qualitative Evaluation

This section elaborates on the most important figures utilised for qualitative evaluation in this thesis, detailing their application and interpretation. All the figures in this section are example visualisations using generic data and do not convey any specific findings or conclusions. Furthermore, additional figures used for qualitative evaluation are provided and explained in Appendix D for completeness.

The figures serve to evaluate the model's performance across different areas and feature spaces, helping to pinpoint strengths and weaknesses.

As shown in section 3.4, it is important to examine how well the model predicts links within strongly connected components. The reciprocal ranks are depicted in Figure 4.5, indicating whether a link is within a connected component.

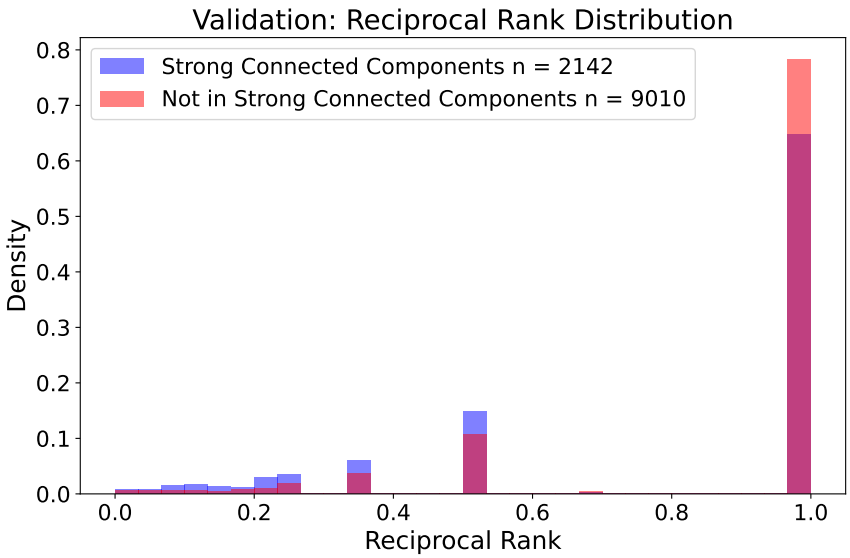


Figure 4.5: Example figure of reciprocal rank of connected components distribution evaluation. It shows how the reciprocal ranks of the predicted links are distributed, separating strongly and not in a strong connected components. A difference in the distributions shows different model performance for the groups.

Additionally, Figure 4.6 examines the cosine similarity of the link with the highest predicted probability. The cosine similarity is calculated between the features (Word2Vec) of the source and target nodes. Cosine similarity instead of euclidean distance is used, because it focuses on the direction of the vectors rather than their magnitude. This is important because in the Word2Vec (Mikolov et al., 2013) features, the vectors are normalised and their length is less informative compared to their direction. It helps in identifying papers that are similar in context, irrespective of their length or magnitude (Jatnika et al., 2019). The goal is to investigate feature areas where the positive link ranked first.

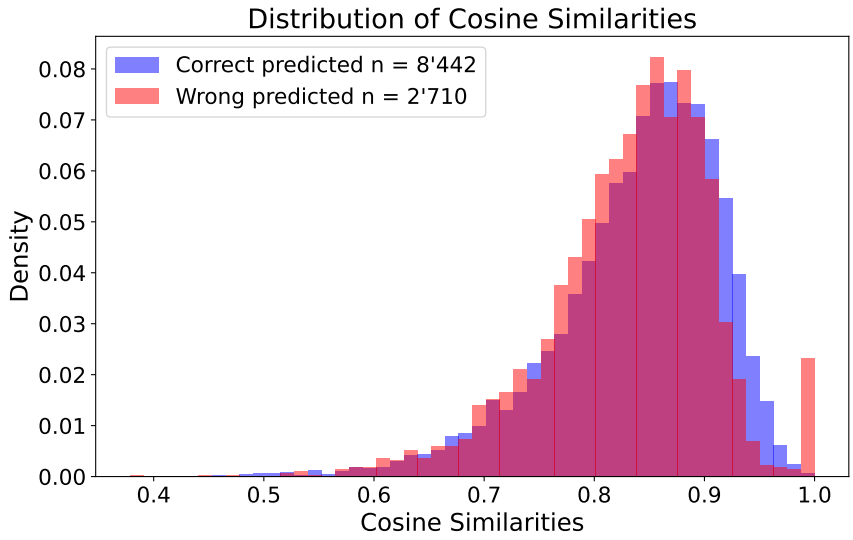


Figure 4.6: Example figure of cosine similarity evaluation for correct (blue) and incorrect (red) predictions in validation set. A correct prediction occurs when the positive edge appears first in the ranking, whereas an incorrect prediction occurs when a negative edge is ranked first.

4.8 Explainability

Permutation importance is a model-agnostic technique that can be applied to various machine learning models to assess the importance of features. For each feature, the model is trained on the original dataset, and the performance metric (e.g. MRR) is recorded. Subsequently, the values of the selected feature are randomly shuffled, disrupting any existing relationship between the feature and the target variable.

The model is then re-evaluated on the dataset with the shuffled feature, and the change in performance is calculated. If shuffling a particular feature significantly impacts the model's performance negatively, it indicates that the model relies heavily on that feature for accurate predictions. The extent of the drop in performance is indicative of the feature's importance. To derive a robust importance score, the described process is iterated multiple times, and the average change in performance is calculated. This iterative approach contributes to a more stable estimate of the feature's importance (Altmann et al., 2010).

4.9 Modelling Results

After training a GNN, it is essential to determine which additional features could enhance the model's performance. This approach allows for the identification of areas where additional features could be beneficial, providing a more accurate assessment of feature importance and model performance. To evaluate this, model predictions were initially trained on a RF (4.4) model using the following features, many of which are explained in section 2:

- **Prediction model:** Prediction scores of the GNN model.
- **Cosine similarity:** Measure of cosine similarity between the source and target nodes.
- **Node year target:** Publication year of the target node.
- **Common neighbor all:** Count of common neighbors when the graph is converted to an undirected graph.
- **Indegree target:** Number of incoming edges to the target node.
- **Outdegree target:** Number of outgoing edges from the target node.

- **Cluster coefficient target:** Measures the clustering tendency of the target node.
- **arXiv category target:** Category of the target node within the arXiv dataset.
- **Node in strong target:** Indicator if the target node is part of a strongly connected component.
- **Indegree source:** Number of incoming edges to the source node.
- **Outdegree source:** Number of outgoing edges from the source node.
- **Cluster coefficient source:** Measures the clustering tendency of the source node.
- **arXiv category source:** Category of the source node within the arXiv dataset.
- **Node in strong source:** Indicator if the source node is part of a strongly connected component.
- **Common neighbor successor:** Count of common neighbors for outgoing links.

These features were used for the RF models, but other features could also be considered. To more accurately assess which features positively influence the model, Permutation Importance (see 4.8) was utilised.

For this evaluation, a 5-fold cross-validation was performed using the validation set. The validation set was chosen because the trained GNN model is unlikely to have overfitted on these data.

To further verify the plausibility of the results, a model was trained on the entire validation set and then tested on the test set. This step ensures that the findings are robust and can generalise well to unseen data, providing confidence in the model's performance and the effectiveness of the selected features.

5 Results

This section presents the results derived from the models and methods detailed in section 4. We begin by reporting the baselines and heuristics, followed by an analysis of the reproduction of the OGB GNN implementation (snap-stanford, 2020). Next, we present the results of reference models, pre-trained models, and fine-tuned models. Additionally, we examine the transfer metrics between reference models and fine-tuned models. These results are provided for both GCN and NGCN architectures. For the model architecture, which demonstrated better transfer performance, we also conduct a qualitative evaluation of the models weaknesses. To further analyse these weaknesses, we employ a random forest model and permutation importance analysis. A comprehensive overview and comparison of all results is shown in Table 5.19. In Appendix E all hyperparameters used are listed.

5.1 Baselines

The cosine similarity method, described in section 4.4, was used to evaluate the performance of a feature based heuristic. The common neighbor baseline, described in section 4.4, was used as a heuristic based on the graph structure.

Graph structure-based baselines outperform feature-based methods on the arXiv CS dataset, achieving a Test MRR of **0.3118** compared to the feature-based method's Test MRR of **0.1827**. The baseline results are summarised in Table 5.1.

Method	Dataset	Valid MRR	Test MRR
cosine similarity	arXiv CS	0.1833	**0.1827**
common neighbor	arXiv CS	0.3130	**0.3118**

Table 5.1: Baseline MRR values for cosine similarity and common neighbor heuristics.

To ensure proper function of our common neighbor method, we also reproduced the heuristic results for ogbl-citation2 dataset. Our heuristics resulted in a Test MRR of **0.5031** while the leaderboard shows a Test MRR of **0.5119**. The deterministic nature of common neighbor should not allow for such differences (further discussed in section 6.1). We cloned their implementation and removed all references to our work. This independent heuristic also resulted in a Test MRR of **0.5031**. The baseline results for ogbl-citation2 are summarised in Table 5.2.

Method	Dataset	Valid MRR	Test MRR
common neighbor	ogbl-citation2	0.5004	0.5031
common neighbor	ogbl-citation2 independent OGB *	0.5004	0.5031
common neighbor	ogbl-citation2 leaderboard OGB	0.5147	0.5119

Table 5.2: Baseline MRR values for common neighbor using the ogbl-citation2 dataset, result marked with * are the actual results when reproducing the leaderboards code independently. The implementation is found in this repository.

5.2 Reproduction

Reproducing the results of the ogbl-citation2 leaderboard result for the GCN ensured the pipeline's functionality. Additionally, it provided a straightforward starting point for modeling since the data split was already available. We trained two models with different set of hyperparameters for the reproduction.

Hyperparameters

To reproduce the results we used the hyperparameters provided by the original repository. However, as these settings did not achieve a similar MRR as reported on the leaderboard, we used the hyperparameters of our arXiv CS reference model. Due to the similar domain (paper-citations), we assumed that the same hyperparameters could be applied effectively. The second model was trained for only 17 epochs because of GPU resource constraints. The used hyperparameters for both models are found in Table 5.3.

Hyperparameter	ogbl-citation2 repository	arXiv CS reference
Model architecture	GCN	GCN
Epoch	48	17
Batch size	65536	38349
Number of layers	3	2
Hidden channels	512	256
Learning rate	0.0005	0.0004

Table 5.3: Reproduction with the hyperparameters of the repository and the best hyperparameters of the arXiv CS reference model described in section 5.3.

Results

The first model with the default ogbl-citation2 parameters resulted in a Test MRR of **0.696** which significantly differs from the results of the leaderboard with a Test MRR of $\boldsymbol{\mu = 0.848 \pm 0.002}$. The best model parameters of the arXiv CS reference models resulted in a Test MRR of **0.695** but with 31 fewer epochs of training. All the results are found in Table 5.4.

Datasplit	ogbl-citation2 Repository	Best arXiv CS run	Leaderboard
Train MRR	0.791	0.789	-
Valid MRR	0.695	0.695	0.847 ± 0.002
Test MRR	0.696	0.695	0.848 ± 0.002

Table 5.4: MRR results comparison for different runs and the official leaderboard results from Stanford (2024).

The Figure 5.1 shows the AUROC, Training MRR and NLL loss for the training of the model with the original parameters. The loss converges with two spikes at epoch 9 and 25. The spikes are mirrored upwards in the training MRR. The AUROC curve increases very fast and stays at a value close to 1.

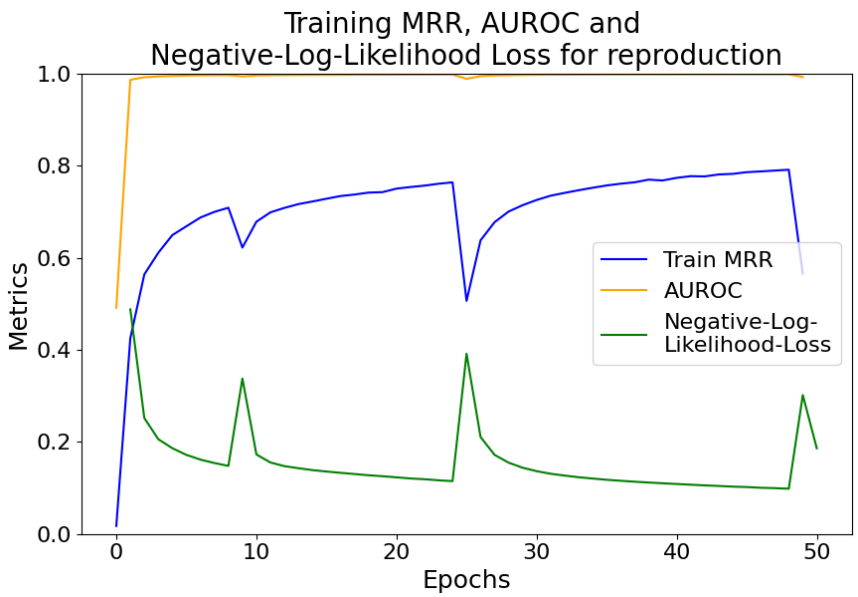


Figure 5.1: Training MRR, AUROC, and loss for reproduction run. The default parameter run is shown.

5.3 GCN arXiv CS Reference

A reference model was trained on the arXiv CS dataset with the GCN architecture for later comparison to a fine-tuned model.

Hyperparametertuning

The first initialisation of hyperparameter tuning involved a broad range of values to explore the search space comprehensively. The Table 5.5 shows the initial grid used for the random search:

Hyperparameter	**Values**
Epoch	150
Batch size	[32768, 262144]
Number of layers	{2, 3}
Hidden channels	{128, 256, 384, 512}
Learning rate	[0.0001, 0.001]

Table 5.5: Hyperparameter grid for first random search with 16 runs.

Based on the initial results, several adjustments were made to the hyperparameters. The batch size range was reduced to focus on smaller values. The number of layers was fixed to 2. The learning rate range was narrowed to smaller values, and the range for hidden channel size was increased. The adjusted hyperparameter grid for the second random search is shown below in Table 5.6:

Hyperparameter	Values
Epochs	150
Batch size	[32768, 65536]
Number of layers	2
Hidden channels	{512, 768, 1024, 1280, 1536}
Learning rate	[0.0001, 0.0007]

Table 5.6: Hyperparameter grid for second random search with 16 runs.

To further optimise the learning rate, a focused search was conducted with more epochs. The specific grid for learning rate optimisation is shown in Table 5.7:

Hyperparameter	Values
Epoch	2100
Batch size	38349
Number of layers	2
Hidden channels	512
Learning rate	{0.0003, 0.0004, 0.0005}

Table 5.7: Hyperparameter grid for learning rate optimisation with 3 runs.

MRR Curves

The model with the best hyperparameters shows increasing MRR curves (Figure 5.2). To estimate variability, the model was trained for 5 runs, each with different parameter initialisations.

The MRR curves have a lot of spikes in their course. Validation and Test MRR are very similar with the Test MRR being slightly less high. Train MRR is about 0.15 MRR points less than the others. The model with the highest Validation MRR (comet run) of all runs had the highest Validation MRR at epoch 2071, in this epoch the model achieved a Test MRR of **0.8478**.

All runs combined achieve, for their best epochs (best validation MRR), a Test MRR of $\boldsymbol{\mu} = \mathbf{0.8457 \pm 0.0021}$.

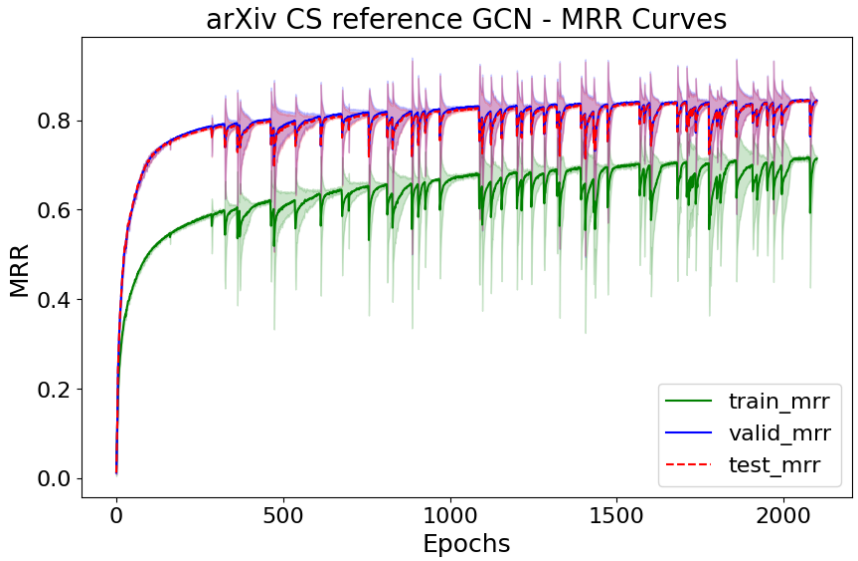


Figure 5.2: MRR curves for the GCN arXiv CS Reference model. The one standard deviation interval for the 5 runs is also shown with a shade in their respective color.

Qualitative Results

In this section, the model is qualitatively analysed. The explanation of all the analyses can be found in section 4.7. The associated visualisations can be viewed at the following link: Comet run.

As shown in Figure 5.3, source nodes that are not part of a strongly connected component frequently have a higher reciprocal rank compared to those within strongly connected components.

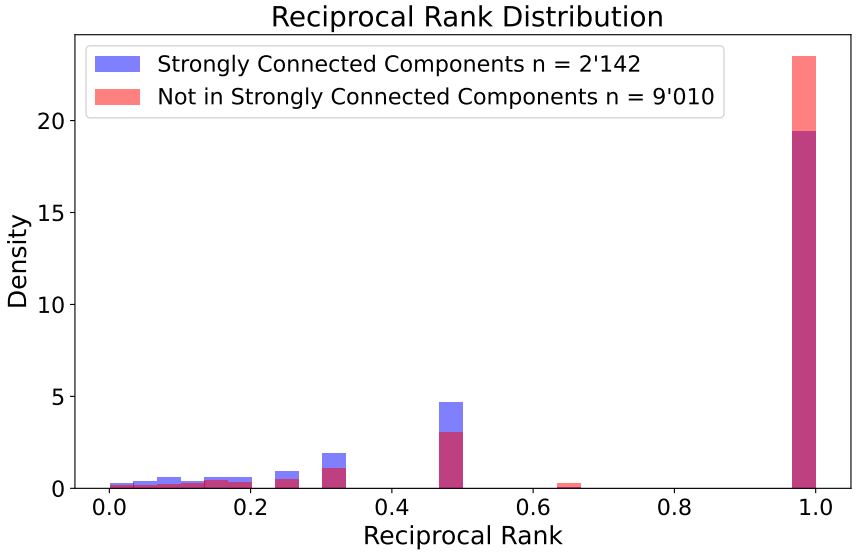


Figure 5.3: Distribution of the reciprocal rank for the GCN arXiv CS reference model in the test set, depending on whether nodes are in a strongly or not in a strongly connected component (detailed explanation in section 4.7).

In Figure 5.4, it is evident that nodes which are correctly predicted (positive link was ranked first) tend to have a higher cosine similarity. An exception occurs at a cosine similarity of 1, where predictions are comparatively often incorrect.

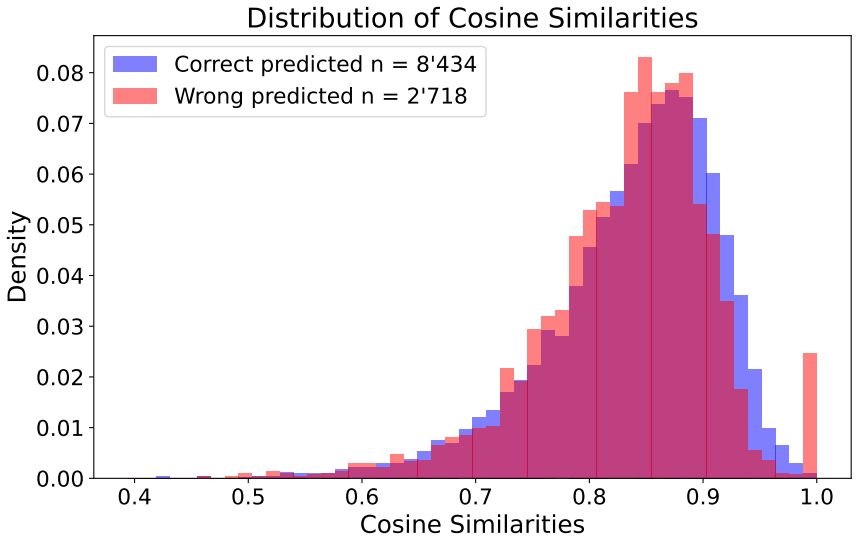


Figure 5.4: Distribution of cosine similarity for the first ranked predictions in GCN arXiv CS reference model, correct predicted means that the positive link is ranked first (detailed explanation in section 4.7)

The Figure 5.3 and Figure 5.4 are discussed and conclusions are drawn in section 6.4.

Modelling Results

After training the GCN, the predictions were further refined using additional features in a RF model. Detailed explanations are provided in section 4.9. This model achieved an MRR of $\boldsymbol{\mu = 0.9273 \pm 0.0021}$.

To more accurately assess which features positively influence the model, Permutation Importance was utilised (see Figure 5.5). Each feature within the five models was randomly permuted twice, highlighting their individual impact.

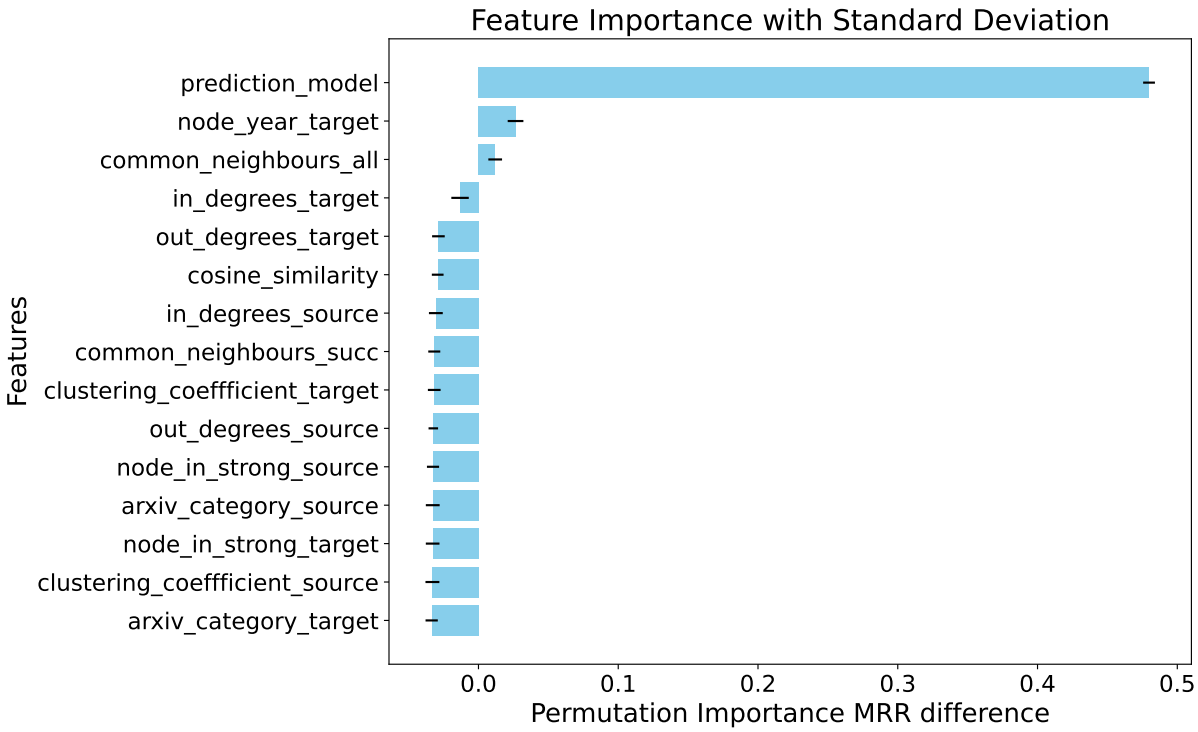


Figure 5.5: Permutation importance of the RF model using all features including the GCN arXiv CS reference model predictions. This model achieved an MRR of $\mu = 0.9273 \pm 0.0021$. The figure shows bars representing the permutation importance, with ticks indicating the standard deviation of the cross-validated importance values.

Given that the results were significantly higher than those of the GCN reference model, it is evident that the additional features enhance the modeling and provide a more accurate prediction. However, the presence of negative Permutation Importance suggests potential correlations between certain features, indicating that some features might not contribute positively or could be redundant. Due to the computationally intensive nature of these calculations and the ambiguity regarding which features are genuinely impactful, the four features with the highest permutation importance were subsequently selected. This model configuration reached an MRR of $\mu = 0.8939 \pm 0.0032$.

Despite a slight decrease in MRR, the primary objective was to identify relevant features, rendering the lower result acceptable. The recalculated permutation importance (see Figure 5.6) indicated the positive contributions of the model's predictions, including the target node's year, the number of indegrees of the target node, and the common neighbors from both incoming and outgoing links.

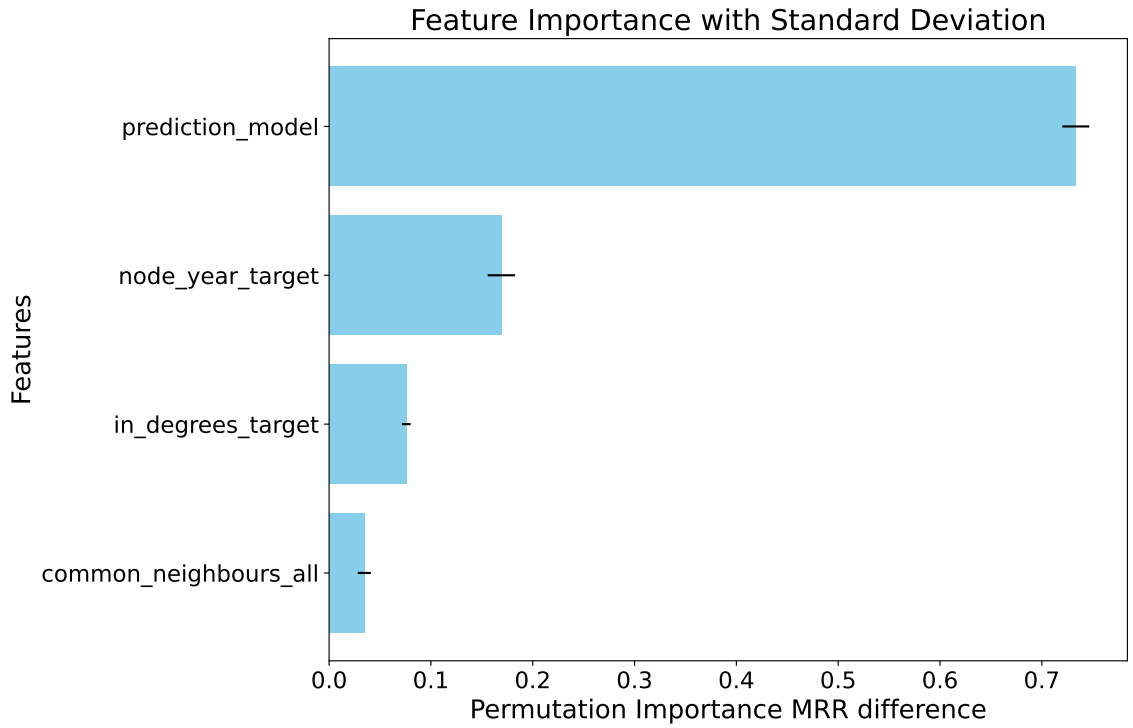


Figure 5.6: Permutation importance of the RF model using four selected features (GCN arXiv CS reference model predictions, target node year, common neighbors, and indegrees). This model achieved an MRR of $\mu = 0.8939 \pm 0.0032$. The figure shows bars representing the permutation importance, with ticks indicating the standard deviation of the cross-validated importance values.

Furthermore, an RF model trained on all validation data was tested on the test set, achieving an MRR of **0.8926**. The corresponding permutation importance, initialised randomly five times, is shown in Figure 5.7. All features demonstrated positive permutation importance.

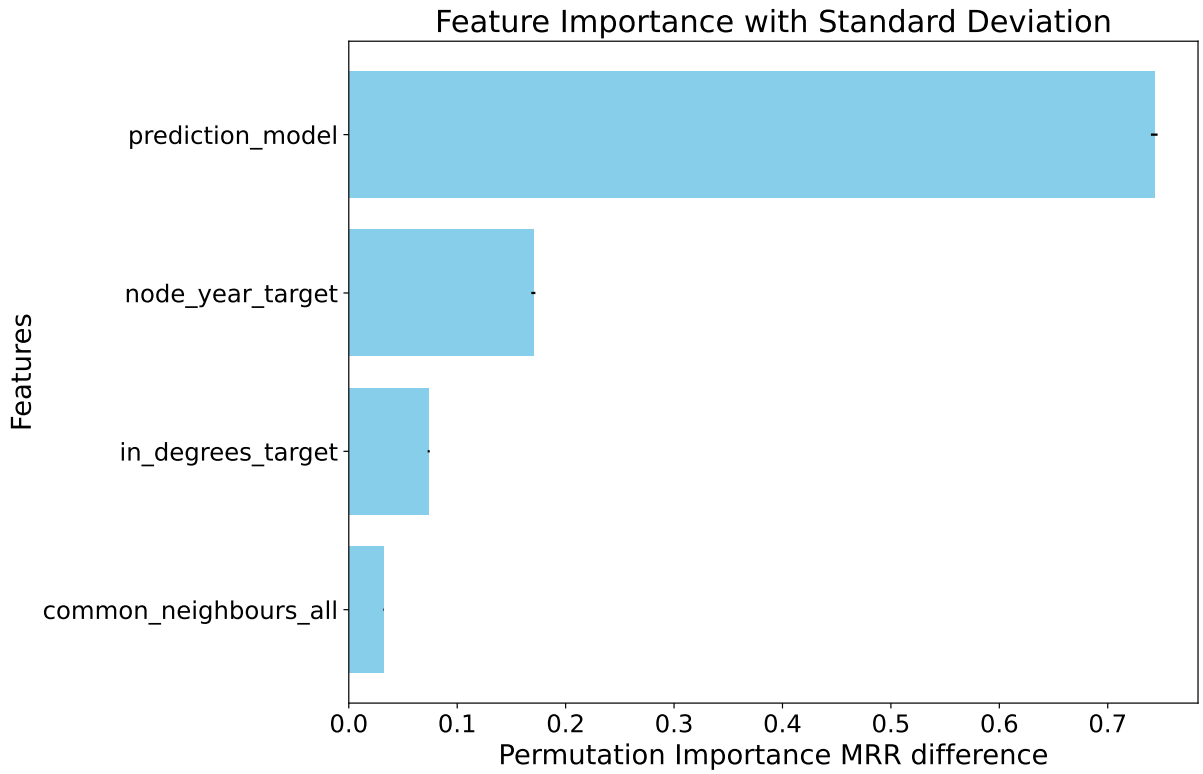


Figure 5.7: Permutation importance of the RF model using four selected features (GCN arXiv CS reference model predictions, target node year, common neighbors, and indegrees) on the test set. This model achieved an MRR of 0.8926. The permutation importance was initialised randomly five times. The figure shows bars representing the permutation importance, with ticks indicating the standard deviation of the cross-validated importance values.

5.4 GCN arXiv w/o CS Pre-training

A model for later fine-tuning was pre-trained on the arXiv w/o CS dataset with the GCN architecture.

Hyperparameter Tuning

The first initialisation of hyperparameter tuning involved a broad range of values to explore the search space comprehensively. As the reference model was trained on a hidden channel size of 512 and 2 layers, the pre-training model was already fixed on these to ensure the comparability between model capacities. Table 5.8 shows the initial grid used for the random search:

Hyperparameter	Values
Epoch	25
Batch size	[32768, 65536]
Number of layers	2
Hidden channels	512
Learning rate	[0.0001, 0.0005]

Table 5.8: Hyperparameter grid for first random search with 16 runs.

Based on the results, the best validation MRR was chosen. The hyperparameter grid for a longer training run is defined in Table 5.9:

Hyperparameter	Values
Epoch	200
Batch size	38349
Number of layers	2
Hidden channels	512
Learning rate	0.0004

Table 5.9: Hyperparameter grid for final longer run of the pre-training model.

MRR Curves

Train and validation MRR curves increase fast and are steadily improving. Some training irregularities are observable in the form of regular spikes. The epoch with the highest Validation MRR was epoch 199, in this epoch the model achieved a Test MRR of **0.8138**.

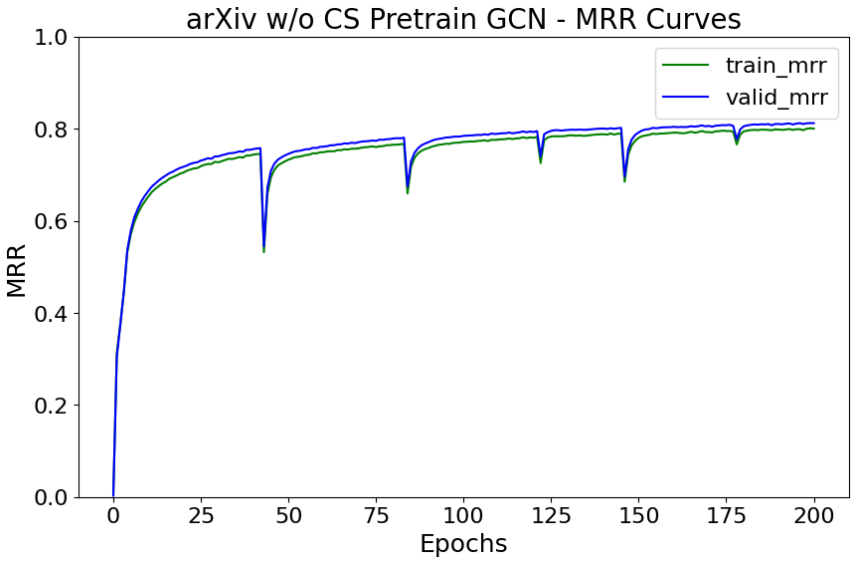


Figure 5.8: MRR Curves for the GCN arXiv w/o CS pre-training model.

5.5 GCN arXiv CS Fine-tuning

A model is fine-tuned on the arXiv CS dataset with the GCN architecture. For this, the best pre-trained model was loaded, which was trained on the arXiv w/o CS dataset seen in section 5.4.

Hyperparameter Tuning

The first initialisation of hyperparameter tuning involved a broad range of batch size and learning rate. As the reference model was trained on a hidden channel size 512 and 2 layers, the pre-training model was already fixed on these parameters to ensure the comparability between model capacities. The freezing of model layers was also tested in the parameter tuning process. Table 5.10 shows the initial grid used for the random search:

Hyperparameter	Values
Epoch	25
Batch size	[25566, 57520]
Number of layers	2
Hidden channels	512
Learning rate	[0.0001, 0.0005]
Freezing layers	{True, False}

Table 5.10: Hyperparameter grid for Random Search with 20 runs.

MRR Curves

The model with the best hyperparameters was trained longer for 2000 epochs. Some training irregularities are observable in the form of spikes. The epoch with the highest Validation MRR was epoch 1896, in this epoch the model achieved a Test MRR of **0.8653**.

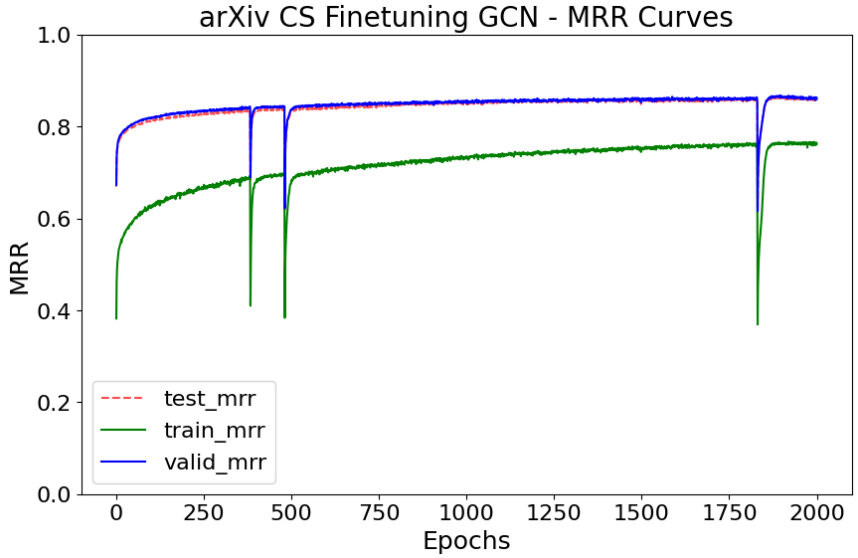


Figure 5.9: MRR Curves for the GCN arXiv CS fine-tuning model. For this, the best pre-trained model was loaded, which was trained on the arXiv w/o CS dataset seen in section 5.4.

Qualitative Results

In this section, the model is qualitatively analysed. The explanation of all the analyses can be found in section 4.7. The associated visualisations can be viewed at the following link: Comet run.

As shown in Figure 5.10, source nodes that are not part of a strongly connected component frequently exhibit a reciprocal rank of one compared to those within strongly connected components.

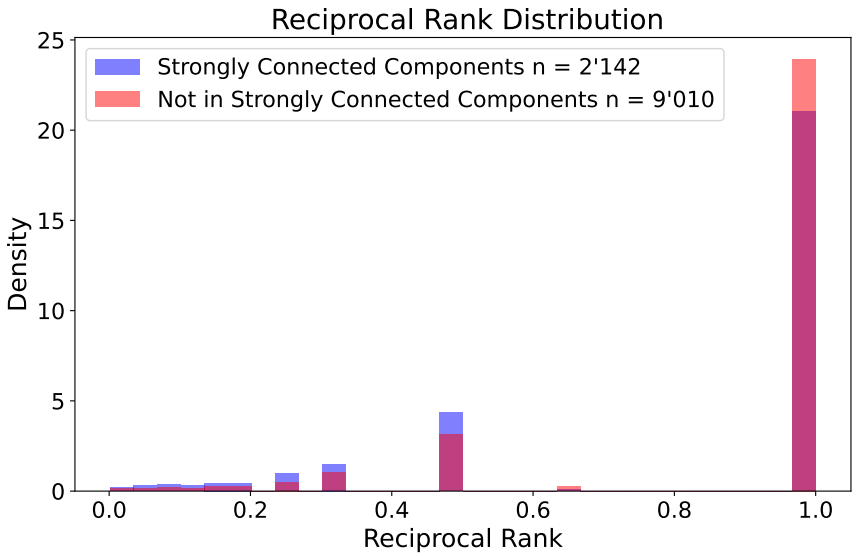


Figure 5.10: Distribution of the reciprocal rank for the GCN arXiv CS fine-tuning model in the test set, depending on whether nodes are in a strongly or not in a strongly connected component (detailed explanation in section 4.7).

In Figure 5.11, it is evident that nodes which are correctly predicted (positive link was ranked first) tend to have a higher cosine similarity. An exception occurs at a cosine similarity of ≈ 1, where predictions are comparatively often incorrect.

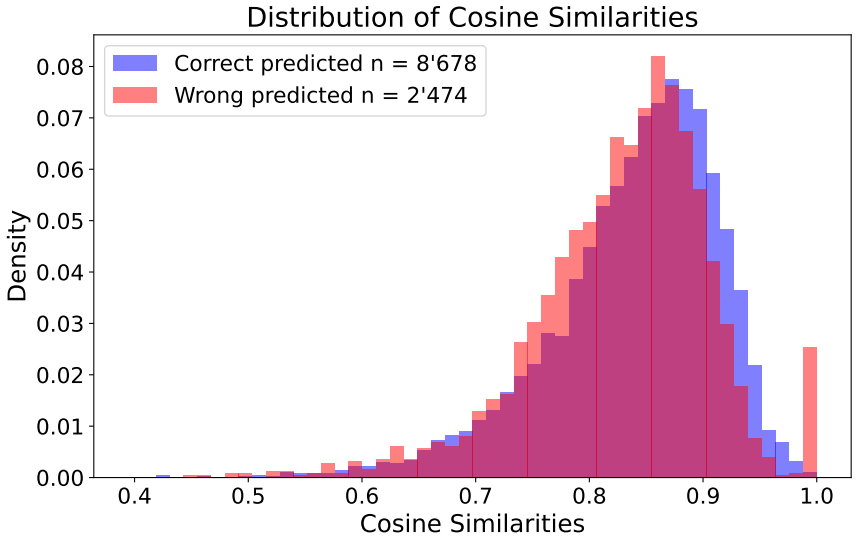


Figure 5.11: Distribution of cosine similarity for the first ranked predictions in the GCN arXiv CS fine-tuning model, correct predicted means that the positive links is ranked first (detailed explanation in section 4.7)

Modelling Results

After training the fine-tuned GCN, the predictions were further refined using additional features in a RF model. Detailed explanations are provided in section 4.9. Using all features on the validation set, an MRR of $\boldsymbol{\mu = 0.9384 \pm 0.0035}$ was achieved. Again, the four features with

the highest permutation importances were selected. These include the prediction model, target node year, common neigbors, and indegrees target, as shown in Figure 5.12.

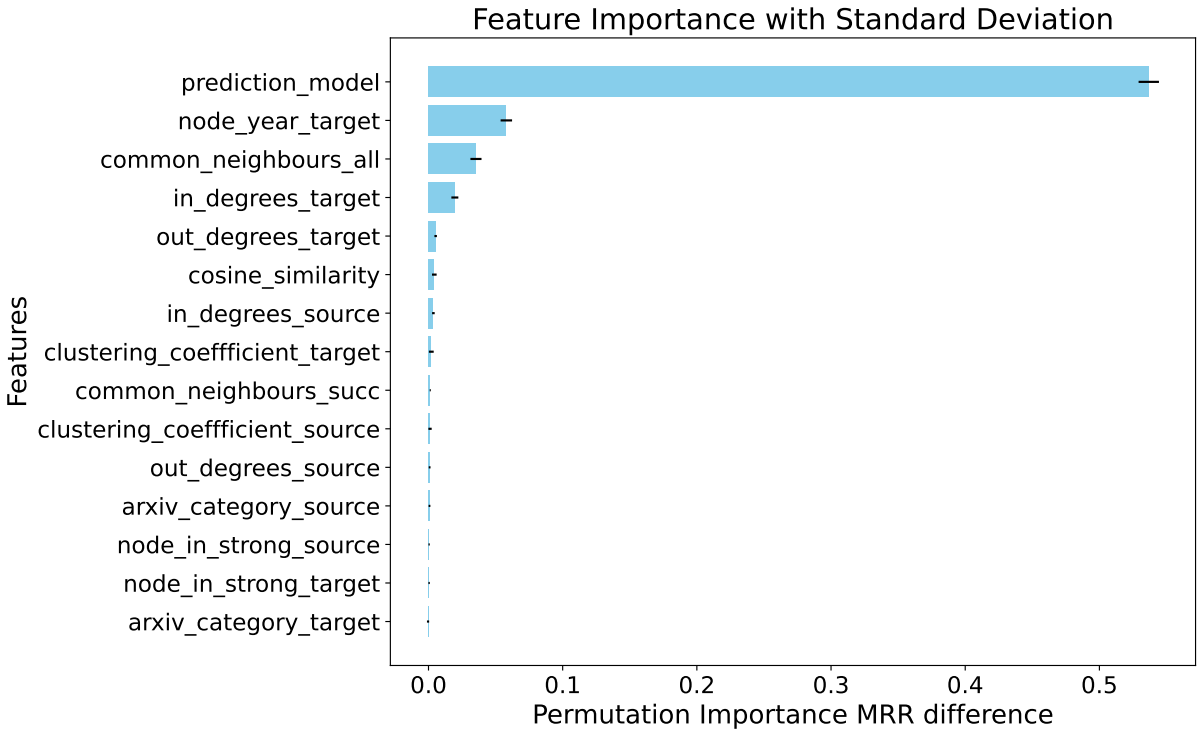


Figure 5.12: Permutation importance of the RF model using all features including the GCN arXiv CS fine-tuning model predictions. This model achieved an MRR of $\mu = 0.9384 \pm 0.0035$ on the validation set. The figure shows bars representing the permutation importance, with ticks indicating the standard deviation of the cross-validated importance values, highlighting the variability and reliability of each feature's contribution.

With these features, an MRR of $\boldsymbol{\mu = 0.9092 \pm 0.0031}$ was achieved. As demonstrated in Figure 5.13, all features demonstrated positive permutation importance.

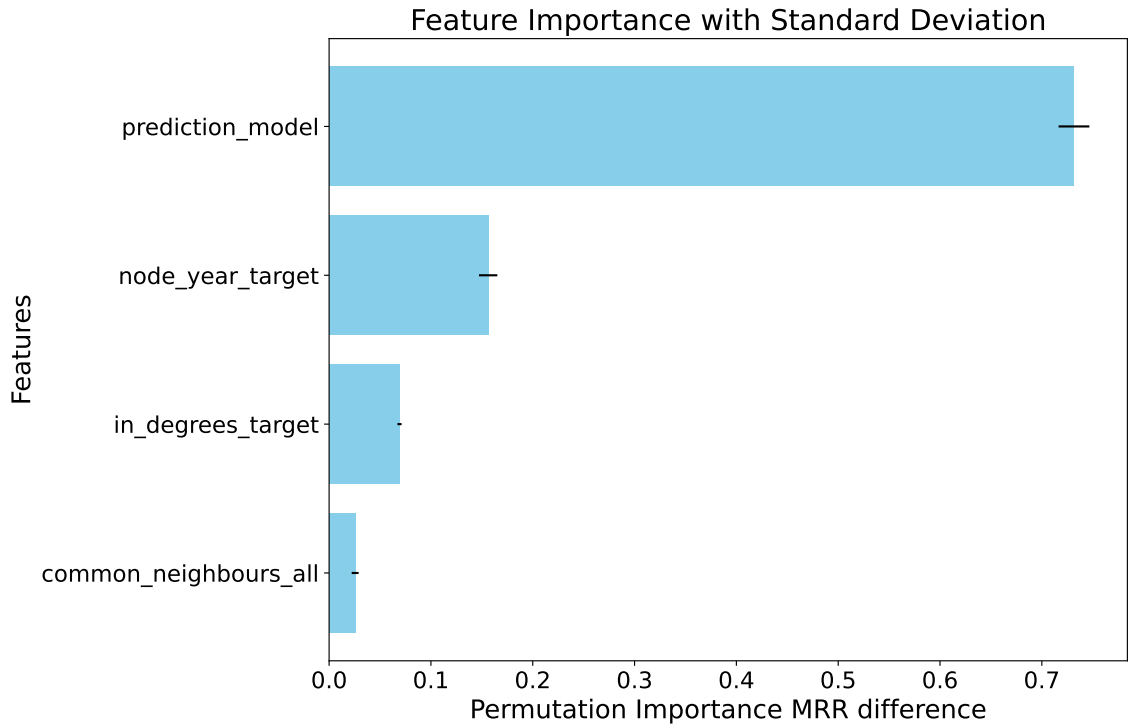


Figure 5.13: Permutation importance of the RF model using four key features (GCN arXiv CS fine-tuning model predictions, target node year, common neighbors, and indegrees). This model achieved an MRR of $\mu = 0.9092 \pm 0.0031$ on the validation set. All selected features demonstrated positive permutation importance. The figure shows bars representing the permutation importance, with ticks indicating the standard deviation of the cross-validated importance values, providing insights into the consistency of each feature's impact.

Furthermore, an RF model trained on all validation data was tested on the test set, achieving an MRR of **0.8856**. The corresponding permutation importance, initialised randomly five times, is shown in Figure 5.14. All features demonstrated positive permutation importance.

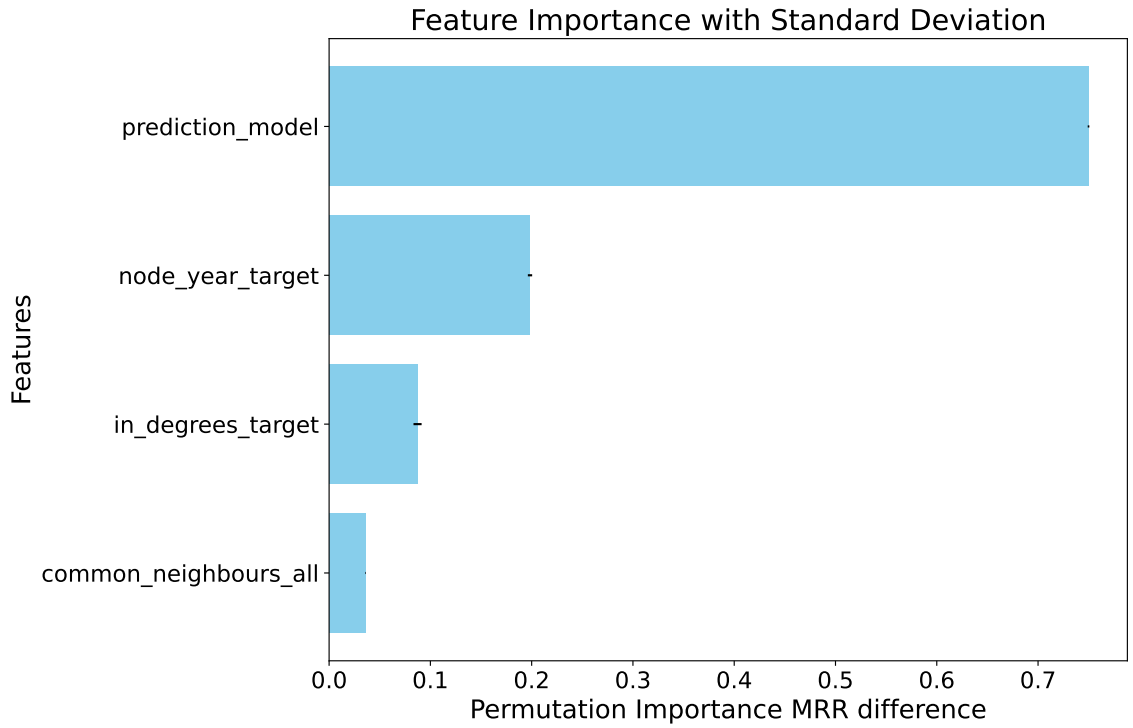


Figure 5.14: Permutation importance of the RF model using four key features on the test set including the GCN arXiv CS fine-tuning model predictions. This model achieved an MRR of 0.8856. The permutation importance was initialised randomly five times. All features demonstrated positive permutation importance. The figure shows bars representing the permutation importance, with ticks indicating the standard deviation of the cross-validated importance values, illustrating the stability and confidence in the feature importance results.

Transfer Metrics

In comparison to the reference model, the fine-tuned model benefits from transfer which resulted in the transfer metrics shown in Table 5.11.

Model	Jumpstart	Asymptotic performance	Transfer ratio
GCN arXiv CS Fine-tuning	0.6553	0.0196	0.0232

Table 5.11: Transfer metrics on the test set for the GCN arXiv CS fine-tuning model compared to GCN arXiv CS reference model.

5.6 NGCN arXiv CS Reference

A reference model for later comparison was trained on the arXiv CS dataset with the NGCN architecture.

Hyperparametertuning

The first initialisation of hyperparameter tuning involved a broad range of values to explore the search space comprehensively. Because of GPU memory constraints the most capable possible combination of hidden channel size and number of layers was 512 and 3 respectively. The Table 5.12 shows the initial grid used for the random search:

Hyperparameter	Values
Epoch	150
Batch size	[32768, 65536]
Number of layers	{2, 3}
Hidden channels	{128, 256, 384, 512}
Learning rate	[0.0001, 0.0007]

Table 5.12: Hyperparameter grid for First Random Search with 15 runs.

Based on the initial results, several adjustments were made to the hyperparameters. The batch size range was reduced to focus on smaller values. The learning rate range was narrowed to bigger values, and the range for hidden channel size was narrowed upwards.

The adjusted hyperparameter grid in Table 5.12 for the second random search is shown below:

Hyperparameter	Values
Epoch	150
Batch size	[10240, 43008]
Number of layers	{2, 3}
Hidden channels	{384, 512}
Learning rate	[0.0003, 0.0007]

Table 5.13: Hyperparameter grid for second random search with 10 runs.

Based on the results of the second grid, several adjustments were made to the hyperparameters. The batch size range was more reduced to focus on smaller values. The learning rate range was narrowed. The number of layers was fixed to 2 as well as the hidden channel size which was fixed to 384.

Hyperparameter	Values
Epoch	150
Batch size	[9000, 18000]
Number of layers	2
Hidden channels	384
Learning rate	[0.0003, 0.0006]

Table 5.14: Hyperparameter grid for third random search with 16 runs.

MRR Curves

The model with the best hyperparameters was trained longer and shows increasing MRR curves. The MRR curves are smooth in their course. Validation and Test MRR are very similar with the Test MRR being slightly less high. Train MRR is about 0.1 MRR points less than the others. The model was trained for 2000 epochs. The epoch with the highest Validation MRR was epoch 1983, in this epoch the model achieved a Test MRR of **0.8654**.

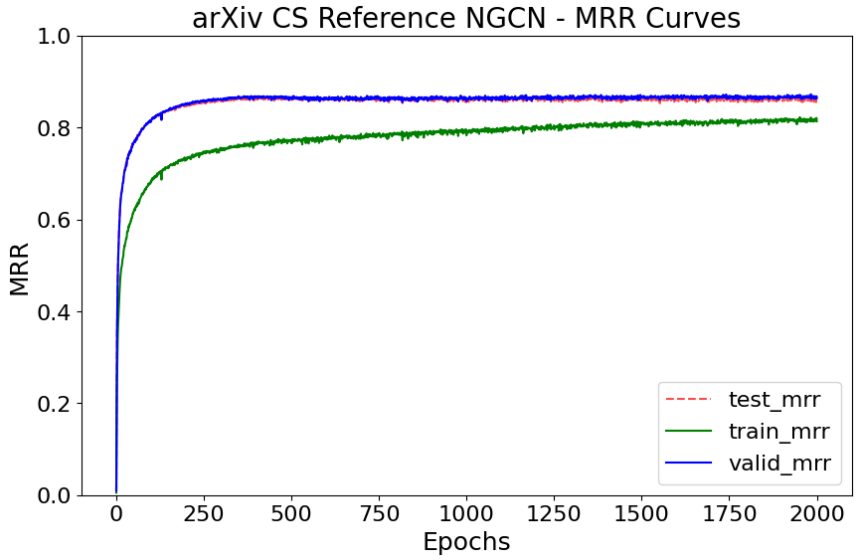


Figure 5.15: MRR curves for the NGCN arXiv CS reference model.

5.7 NGCN arXiv w/o CS Pre-training

A model for later fine-tuning was trained on the arXiv w/o CS dataset with the NGCN architecture.

Hyperparameter Tuning

The first initialisation of hyperparameter tuning involved a broad range of values to explore the search space comprehensively. As the reference model was trained on a hidden channel size of 384 and 2 layers, the pre-training model was already fixed on these to ensure the comparability between model capacities. Table 5.15 shows the initial grid used for the random search:

Hyperparameter	Values
Epoch	25
Batch size	[32768, 65536]
Number of layers	2
Hidden channels	384
Learning rate	[0.0001, 0.0005]

Table 5.15: Hyperparameter grid for first random search with 14 runs.

Based on the results, the best validation MRR was chosen. The hyperparameter grid for longer training runs is defined in Table 5.16:

Hyperparameter	Values
Epoch	200
Batch size	[10240, 43008]
Number of layers	2
Hidden channels	384
Learning rate	[0.0003, 0.0007]

Table 5.16: Hyperparameter grid for second random search with 11 runs.

MRR Curves

Train and validation MRR curves increase rapidly at the beginning and are steadily improving. The epoch with the highest Validation MRR was epoch 178, in this epoch the model achieved a Test MRR of **0.8088**.

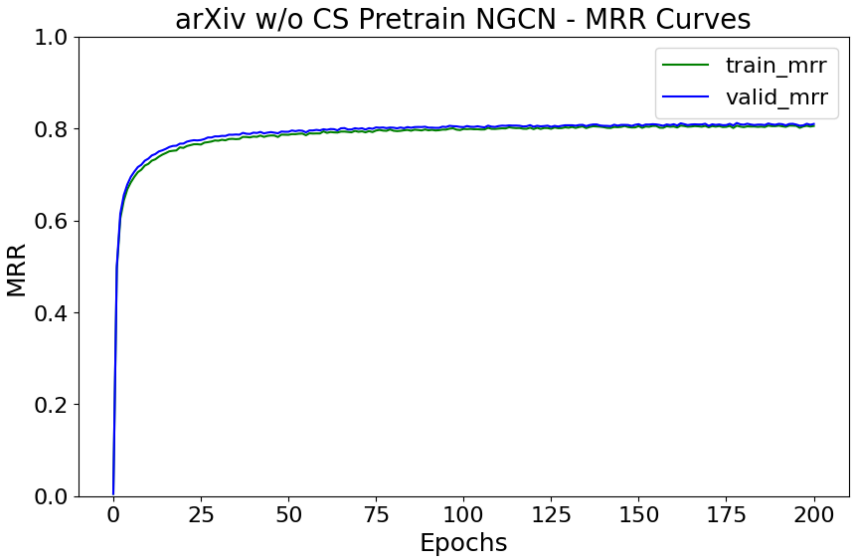


Figure 5.16: MRR curves for the GCN arXiv w/o CS pre-trained model.

5.8 NGCN arXiv CS Fine-tuning

A model was fine-tuned on the arXiv CS dataset with the NGCN architecture. To this end, the most suitable pre-trained model was loaded, which was trained on the arXiv w/o CS dataset seen in section 5.7.

Hyperparameter Tuning

The first initialisation of hyperparameter tuning involved a broad range of batch size and learning rate. As the reference model was trained on a hidden channel size of 384 and 2 layers, the pre-training model was already fixed on these to ensure the comparability between model capacities. The freezing of model layers was also tested in the parameter tuning process. The Table 5.17 shows the initial grid used for the random search:

Hyperparameter	Values
Epoch	25
Batch size	[6000, 18000]
Number of layers	2
Hidden channels	384
Learning rate	[0.0007, 0.0014]
Freezing layers	{True, False}

Table 5.17: Hyperparameter grid for random search with 15 runs.

MRR Curves

The model with the best parameters was trained longer for 1841 epochs. The epoch with the highest Validation MRR was epoch 1397, in this epoch the model achieved a Test MRR of 0.8593.

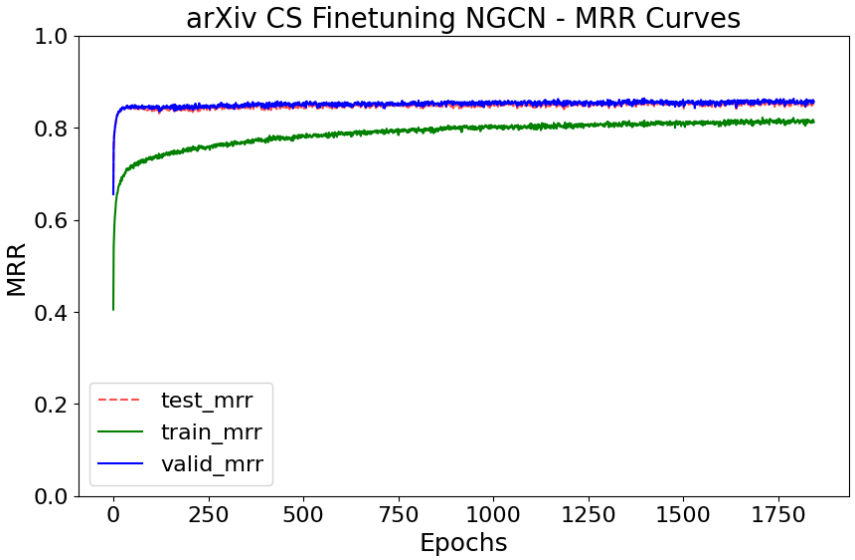


Figure 5.17: MRR curves for the GCN arXiv CS fine-tuning model. For this, the best pre-trained model was loaded, which was trained on the arXiv w/o CS dataset seen in section 5.7.

Transfer Metrics

In comparison to the reference model the fine-tuned model does not profit from transfer, it even negatively transferred, which resulted in the following transfer metrics shown in Table 5.18.

Model	Jumpstart	Asymptotic performance	Transfer ratio
NGCN arXiv CS Finetuning	0.6485	-0.0071	-0.0082

Table 5.18: Transfer metrics on the test set for the NGCN arXiv CS fine-tuning model compared to NGCN arXiv CS reference model.

5.9 Comparison of Results

Below is a comprehensive summary of all the quantitative results across models. Table 5.19 and Table 5.20 provide a clear overview of the presented results. Detailed information on the hyperparameters used can be found in the Appendix E.

Method / Model	Valid MRR	Test MRR	Experiment link
Baselines (5.1):			
Cosine similarity arXiv CS	0.1833	0.1827	comet
Common neighbor arXiv CS	0.3130	0.3118	comet
Common neighbor ogbl-citation2	0.5004	0.5031	comet
Reproduction (5.2):			
Reproduction OGB Parameter	0.6950	0.6960	comet
Reproduction arXiv CS Parameter	0.6950	0.6950	comet
Reference:			
GCN arXiv CS reference * (5.3)	0.8486 ± 0.0014	0.8457 ± 0.0021	comet
NGCN arXiv CS reference (5.6)	0.8722	0.8654	comet
Pre-train:			
GCN arXiv w/o CS Pre-training (5.4)	0.8120	0.8138	comet
NGCN arXiv w/o CS Pre-training (5.7)	0.8118	0.8088	comet
Fine-tuning:			
GCN arXiv CS Fine-tuning (5.5)	0.8673	0.8653	comet
NGCN arXiv CS Fine-tuning (5.8)	0.8636	0.8593	comet

Table 5.19: Comparison of all the results. Models or Heuristics are grouped and named with their respective result chapter. Validation and Test MRR are shown for comparison. For each result the experiment link in comet-ml is provided. Models marked with * are run 5 times. Standard deviation is indicated by $\pm$.

Method / Model	Jumpstart	Asymptotic performance	Transfer ratio	Experiment link
Fine-tuning:				
GCN arXiv CS Fine-tuning (5.5)	0.6553	0.0196	0.0232	comet
NGCN arXiv CS Fine-tuning (5.8)	0.6485	-0.0071	-0.0082	comet

Table 5.20: Comparison of all the transfer learning metrics. Fine-tuned models are named with their respective result chapter. Jumpstart, asymptotic performance and transfer ratio are shown. For each result the experiment link in comet-ml is provided. Kooverjee et al. (2022) used the AUROC score, we use Test MRR of the epoch with the highest Validation MRR for the reference models and the fine-tuned models to calculate the transfer metrics.

6 Discussion

This discussion addresses key research questions to evaluate the performance of GNNs compared to traditional heuristic models and the impact of transfer learning and model capacity on GNN performance.

The research questions guiding our discussion are:

- Can a GNN achieve a higher metric (e.g. MRR) than a heuristic model?
- Can the benefits of transfer learning be utilised with a pre-trained model to achieve an increase in jumpstart, asymptotic performance or transfer ratio (4.7) compared to a GNN reference model?
- Are there edges of specific graph structures that a GNN cannot predict?
- Can a GNN trained with transfer learning predict these edges?
- Does an increase in model capacity increase the model performance for transfer learning?

We begin by validating the GNN pipeline, discussing the choice of metrics, and addressing model variability and sampling methods. These decisions are important for understanding the results.

6.1 Pipeline Decisions

Given the relatively recent advent and limited general knowledge of GNNs, this discussion begins by validating the functionality of the GNN pipeline, a crucial step in ensuring the robustness and reliability of the methodologies employed. Additionally, the rationale behind selecting the MRR over the AUROC score for subsequent analyses will be explained. Furthermore, GNNs specificities like random sampling are discussed.

Pipeline Functionality

The primary objective was to verify the correct construction of the pipeline. Initially, an attempt was made to reproduce the results from the ogbl-citation2 leaderboard. Using the CN heuristic, we achieved results with a slight deviation in MRR of 0.0143 on the validation set and 0.0088 on the test set, these results are shown in Table 5.2. These deviations cannot be fully explained due to the deterministic nature of the CN heuristic. Therefore, the original leaderboard implementation by Muhan Zhang et al. (2024) was executed, yielding the same deviated results. This suggests that the dataset or the negative sampling must have been adjusted, confirming the correctness of our implementation.

Further validation involved testing the GNN pipeline. Initially, the pipeline's ability to learn with respect to the given task was verified by overfitting on a single batch. The model successfully memorised the whole batch of edges.

For the reproduction model (5.2), we attempted to recreate the results using the hyperparameters provided in the OGB repository from Weihua Hu and Mathias Fey (2024). This effort resulted in a deviation of 0.152 in MRR for both the validation and test sets (Table 5.4), indicating failure to reproduce the results accurately. Due to limited computing resources, extensive hyperparameter tuning was not feasible. However, the best hyperparameters from the GCN reference model (5.3) were tested, producing similar results to those in the ogbl-citation2 leaderboard while using significantly fewer training epochs (5.3).

These results do not ensure reproducibility of the original repository and leaderboard entry with the provided hyperparameter values. This outcome may be attributed to factors similar to those observed in the CN case, such as potential adjustments to the dataset or negative sampling. Alternatively, it could be the result of using different hyperparameter values. While

we cannot conclusively identify the reasons for our failure to reproduce the results, our pipeline is still capable of training GNNs effectively.

Training Curves

In machine learning, the performance metric, like MRR, is typically higher on the training set than on the validation or test sets, as the model is trained specifically on the training data, optimising its performance there compared to unseen data. However, our results, illustrated in Figure 5.2, deviate from this norm. Furthermore, this discrepancy is inversely correlated with the dataset size, as detailed in Table 6.1. Larger datasets tend to exhibit a smaller gap between training and evaluation metrics.

Dataset	Edge count	mean MRR gap
ogbl-citation2	30'561'187	-0.0787
arXiv w/o CS	12'823'767	0.0122
arXiv CS	1'166'243	0.1581

Table 6.1: Mean MRR differences (Validation to Train over all epochs) for different datasets. There is an inverse correlation to the difference in edge counts (dataset size) and the difference in MRR, models used are found in sections 5.2, 5.3 and 5.4.

Our explanation for the observed performance discrepancy is threefold. Firstly, as illustrated in Figure 6.1, the distributions of pairwise cosine similarity of embeddings within the training and validation set respectively are shown. The validation set shows a distribution shifted to the right, indicating that the validation nodes are more similar to each other compared to those in the training set. This higher similarity among validation nodes might provide an advantage for the model, making it easier to predict validation edges more accurately than those in the training set.

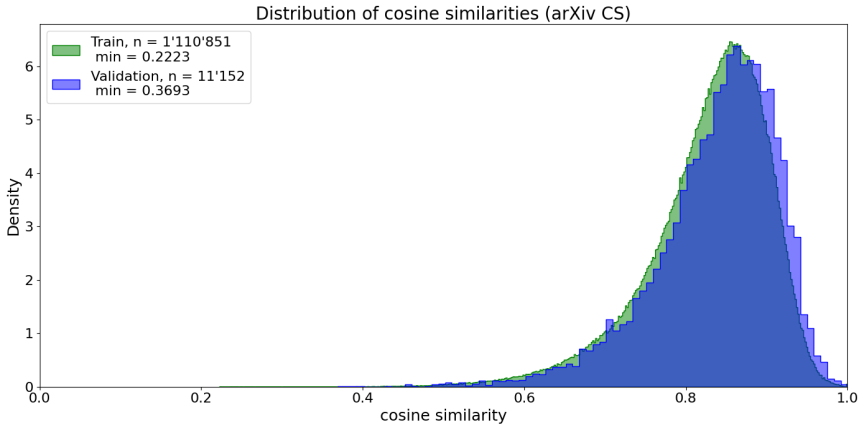


Figure 6.1: Distribution of cosine similarities in the arXiv CS dataset differentiating between validation and train nodes, some validation nodes may intersect with the trainset.

A second factor contributing to the performance discrepancy is the random sampling of negative links during training. For the validation edges, a fixed set of negative samples is used to evaluate performance. In contrast, for the training set, negative links are resampled at each training step. This resampling introduces additional noise into the training process, making the training set more challenging to learn. However, this noise acts as a regularisation factor, preventing the model from overfitting and ensuring that the evaluation task remains non-trivial. J. Li et al. (2023) provides a more detailed analysis of the advantages of random sampling in their research.

Finally, in the arXiv CS dataset, all links that do not fall within the computer science domain have been removed by OGB. This could have a stronger impact on the training data, as there are fewer links available due to the earlier publication years.

While these factors contribute to our understanding of the performance discrepancy, they do not fully explain the phenomenon, especially the inverse correlation of the dataset size remains unsolved (Table 6.1). Further analysis is needed to comprehensively understand and address this issue.

Model Variability

Due to the extensive training times for each model run (up to 24 hours per run) we exclusively trained the arXiv CS reference model five times to establish an estimate for the variability of GCN models (Table 5.19). The GCN model entry on the ogbl-citation2 leaderboard reports a Test MRR of 0.8474 ±0.0021. In comparison, our reference model achieves a Test MRR of 0.8457 ±0.0021, demonstrating similar performance and variability. We use this reference model estimate as our proxy for all further model variabilities, as they were either GCN or NGCN architectures.

Metric Choice

Further investigation into appropriate metrics revealed that the AUROC score showed little sensitivity to changes in loss during training. This prompted a deeper analysis of the AUROC metric's drawbacks in the context of link prediction.

AUROC has significant disadvantages in the presence of highly imbalanced data, as highlighted by Davis and Goadrich (2006) and Saito and Rehmsmeier (2015). In our sampling, detailed in section 4.7, the ratio between positive and negative links is 1 to 1000, which is highly imbalanced.

A significant issue with AUROC is its high tolerance for the False Positive Rate (FPR). For instance, in a scenario with 1000 negative links and one positive link, if 10 negative links are predicted with higher probabilities than the positive link, the FPR would be 0.99 and the True Positive Rate (TPR) would be 1, assuming the threshold is set appropriately. This situation leads to an overestimation of the model's performance.

Alternatives such as Precision-Recall curves or rank metrics are more effective (J. Li et al., 2023). Therefore, this thesis focuses exclusively on the MRR metric in the subsequent discussion.

Random Sampling

An additional area for improvement is the selection of negative samples in the training process. More realistic samples could be chosen instead of random ones, as it is easier for the model to identify negative links when there is no connection between nodes (J. Li et al., 2023).

A straightforward yet enhanced approach would involve omitting future citations and self-loops from the sampling process.

A more sophisticated method, proposed by J. Li et al. (2023), would be the Heuristic Related Sampling Technique (HeaRT). Simply put, this technique involves sampling from both directions not only from the source node but also selecting additional negative source nodes, while the target node remains the same as in the positive link. This approach might not be optimal in our case, as the prediction task is primarily viewed from the perspective of the source node.

Subsequently, the negative samples are ranked using various heuristics, and the most challenging negative links are selected. A more detailed explanation can be found in J. Li et al. (2023).

6.2 RQ1: Improved Predictions with GNN

The objective of this research question is to demonstrate whether GNNs offer advantages in link prediction compared to traditional heuristics. This investigation lays the groundwork for exploring the potential of transfer learning.

As shown in Table 5.19, both graph structure heuristics (CN) and node feature heuristics (Cosine Similarity) exhibit significantly lower MRR (0.1827 and 0.3118) compared to the arXiv CS reference model (0.8457 ± 0.0021). This confirms that GNNs, specifically the GCN architecture in this case, produce better predictions.

This result aligns with existing literature (Kipf and Welling, 2017, Hamilton et al., 2018, Zhang and Li, 2021), where traditional heuristics consistently underperform compared to GNNs.

Kipf and Welling (2017) demonstrated that GCNs can effectively aggregate information from a node's local neighborhood, capturing not only direct connections but also higher-order dependencies through multiple convolutional layers. This ability to learn complex, non-linear relationships is a key advantage over heuristics like CN, which only considers one structural feature like immediate neighbors and ignores broader graph topology.

6.3 RQ2: Improvements of Transfer Learning

This research question aims to demonstrate the transferability of a task (link prediction) inside a domain (paper citations) by highlighting the improvements achieved from the reference model (5.3) to the fine-tuned model (5.5), as measured by transfer learning metrics (4.7).

Our results demonstrate that transfer learning enhances the performance of a fine-tuned model compared to its reference model. The GCN architecture benefits from a significant jump start (0.6553) and some improvements in asymptotic performance (0.0196), as shown in Table 5.11. The substantial jump start highlights the similarity between the data distributions, allowing the pre-trained model to begin training with a relatively high MRR metric.

Additionally, the fine-tuned model demonstrates a faster training time compared to the reference model. For this comparison, we evaluated the test MRR (≈ 0.82) at 10 hours of training time for the reference model. We then plotted the time required for the fine-tuned model to achieve the same test MRR. As illustrated in Figure 6.2, the fine-tuned model reaches the same performance metric substantially quicker. Specifically, the fine-tuned model achieves the equivalent test MRR in approximately 2.3 hours, which is roughly 4.35 times faster than the reference model.

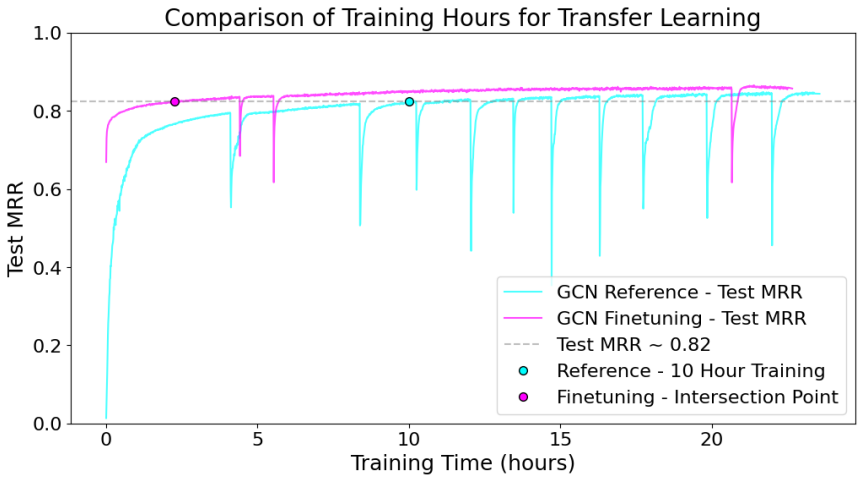


Figure 6.2: MRR Curves for the GCN arXiv CS Reference and GCN arXiv CS Fine-tuning models plotted against training time in hours. The cyan point marks the test MRR of the reference model at 10 hours of training. The gray dashed line indicates this chosen test MRR value. The magenta point highlights the intersection of this line with the fine-tuned model's curve. Appendix B extends the discussion for the NGCN architecture.

The asymptotic performance, or the final training results, do not differ substantially indicating a potential performance ceiling. Notably, the GCN architecture struggled to overfit on the training data, as illustrated in Figure 5.2. This suggests a possible constraint in the model's capacity. To address this, increasing the number of channels per layer or incorporating additional linear layers could potentially enhance asymptotic performance. This observation underpins the motivation for analysing our NGCN architecture in section 6.6.

6.4 RQ3: Challenging Graph Structures for GNNs

The objective of this research question is to identify areas where the model underperforms. This analysis helps in determining what needs to be augmented or examined further to enhance the GCN. Additionally, it lays the groundwork for further research questions (6.5).

One issue highlighted in Figure 5.3 is the necessity for further analysis concerning strongly connected components. The model performs better when the source node is not part of a strongly connected component. This might be due to erroneous citations that are difficult to detect because they are absent. Another hypothesis could be that these components possess different structural characteristics.

As noted at the start of this discussion, adjusting the negative sampling method is advisable. Furthermore, Figure 5.4 highlights an additional issue. Detailed analysis of high similarity scores revealed that various source nodes had been sampled as target nodes, leading to the prediction task effectively predicting self-loops. This scenario is unrealistic and should be avoided to ensure more accurate results. Apart from this specific condition, it is interesting to note that correct predictions, which rank first, tend to have a higher similarity compared to incorrect predictions. This aligns with the premise that existing links are generally closer to each other (details are shown in Figure C.5).

When the predictions of the GCN arXiv CS reference model are further trained with additional features in a RF model the predictions improve significantly ($\mu = 0.8457 \pm 0.0021$ to 0.8856). However, this could also be attributed to the validation nodes being more recent, hence closer to the test nodes. Furthermore, Figure 5.7 suggests that the target node year, the indegree of

the target, and CN neighbor values from the converted undirected graph could further enhance the model. This indicates that the GNN-based model is unable to capture this information.

6.5 RQ4: Solving Challenging Graph Structures with Transfer Learning

The aim of this research question is to explore whether certain edges can be newly predicted or if some are consistently less accurately predicted. This examination helps to explain the specific types of edges for which transfer learning may offer benefits.

In our analyses (5.10, 5.11 and 5.14), no significant differences were observed when compared to the reference model discussed in section 6.4. Consequently, the strengths and weaknesses of our approach relative to the reference model appear to be similar. This finding suggests that while transfer learning may not broadly enhance edge prediction capabilities, it maintains comparable performance with the reference model.

Kooverjee et al. (2022) systematically evaluated model architectures against predefined graph structures (Modularity etc.). They showed that GCN (Kipf and Welling, 2017) and Graph Isomorphism Network (Xu et al., 2019) rely more on structural information, while GraphSAGE (Hamilton et al., 2018) uses structural features for jumpstart. Additionally, the study confirms effective transfer for graph classification with both GCN and GraphSAGE, and refutes the idea that transfer success is due to graph structure alone, highlighting the importance of node attributes. For a more comprehensive understanding of transfer learning dynamics, it is recommended to further explore this research question using the framework provided by Kooverjee et al. (2022), which evaluates in detail structural differences, of graph datasets with graph augmentation and synthetic graph generation, as well as different model architectures.

6.6 RQ5: Advantage of Model Capacity for Transfer Learning

The aim of this research question is to explore whether model capacity is advantageous in transferring. This is measured by an increase in the transfer metrics compared to models with less capacity.

In our thesis, the NGCN pre-training model (5.7), which used a diverse dataset spanning multiple disciplines, showed similar performance to the GCN pre-training model (5.4). However, when the NGCN architecture is fine-tuned on our specific arXiv CS dataset, the model's performance decreased in comparison to the reference model (5.6).

The results of the NGCN model, which includes more linear layers and thus has a greater capacity to learn complex embeddings, indicate an unsuccessful transfer. In fact, the final transfer is negative. While the jumpstart (0.6485), is considerably high, the asymptotic performance is negative. Specifically, the asymptotic performance value of -0.0071 is greater than the established variability of the GCN architecture. This finding indicates that the observed negative transfer is significant and exceeds the model's expected variability, considering the limitations (with only a single proxy for variability) discussed in section 6.1.

The parameter count difference between NGCN and GCN is 49'408 (increase of $\approx$ 8%). This increase is marginal. Due to limited GPU resources, it was not possible to train larger models. To fully explain a relationship between model capacity and performance, models with a substantial increase in capacity, for example a hidden channel size of 2048, would need to be trained.

The marginal capacity difference of our compared models indicates no advantages in increased model capacity for transfer in our domain and the arXiv CS dataset.

Model	Parameter count
GCN arXiv CS Fine-tuning (5.5)	591'873
NGCN arXiv CS Fine-tuning (5.8)	641'281

Table 6.2: Comparison of model capacity by parameter count, parameters of aggregation and predictor module are summed to a single value

7 Conclusion & Outlook

The goal of this thesis was to investigate the transferability of GNNs in the context of link prediction and to assess whether GNNs outperform traditional heuristic methods.

One of the findings of this thesis is that the GCN architecture effectively captured both structural and feature information, outperforming heuristic models. Furthermore, fine-tuned models exhibited faster training times and better initial performance metrics. This observation confirms the benefits of transfer learning thereby saving both time and computational resources.

Another important finding was the difficulties GNNs encountered in predicting edges within strongly connected components. A potential improvement could be realised by adding additional features, such as the target node's year, target node indegree, and common neighbor values from the converted undirected graph. This suggests that adding contextual information may enhance prediction accuracy.

However, it was also found that increased model capacity did not consistently lead to improved transfer learning performance. This suggests that a marginal model capacity increase does not necessarily yield better results and that the model architecture needs to be carefully tailored to the specific task. The lack of significant increases in model capacity in our work suggests the need for further research.

Other model architectures could potentially enhance transferability. Kooverjee et al. (2022) demonstrated successful transfer using other architectures, such as the Graph Isomorphism Network (Xu et al., 2019) and GraphSAGE (Hamilton et al., 2018). Additionally, architectures incorporating attention mechanisms, such as Graph Attention Networks (Veličković et al., 2018), may also enhance transferability.

In future work, we encourage researchers to follow the approach of Kooverjee et al. (2022), who employed synthetic graphs in their transfer learning research. This methodology provided valuable insights into the structural components of graphs and their transferability.

Additionally, the pipeline could be improved by using strategic sampling for negative links. Choosing negative examples more carefully could optimise model performance.

Another area that requires further investigation is the lower Train MRR. Additional research is needed to identify the causes of the lower values and to establish confidence in this anomaly. To achieve this, others should replicate our work and validate our findings.

In summary, this thesis has highlighted the strengths and weaknesses of GNNs in the context of link prediction and their transferability. It has also identified key areas for future research and optimisation. The findings of this investigation provide valuable insights and lay the groundwork for further studies aimed at enhancing the efficiency and accuracy of transfer learning of GNNs in link prediction.

References

Altmann, A., Toloşi, L., Sander, O., & Lengauer, T. (2010). Permutation importance: A corrected feature importance measure. *Bioinformatics*, *26*(10), 1340–1347. https://doi.org/10.1093/bioinformatics/btq134

Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, *13*(2012), 281–305.

Borile, C., Perotti, A., & Panisson, A. (2023, April). Evaluating Link Prediction Explanations for Graph Neural Networks. Retrieved March 22, 2024, from https://arxiv.org/pdf/2308.01682.pdf

Bradley, A. P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, *30*(7), 1145–1159. https://doi.org/10.1016/S0031-3203(96)00142-2

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Brochier, R., Guille, A., & Velcin, J. (2019). Global Vectors for Node Representations [arXiv:1902.11004 [cs]]. *The World Wide Web Conference*, 2587–2593. https://doi.org/10.1145/3308558.3313595

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., … Amodei, D. (2020, July). Language Models are Few-Shot Learners [arXiv:2005.14165 [cs]]. Retrieved June 5, 2024, from http://arxiv.org/abs/2005.14165

Comet ML, I. (2024, August). Comet. Retrieved August 13, 2024, from https://www.comet.com/site/

Dąbrowski, M., & Michalik, T. (2017). How effective is Transfer Learning method for image classification, 3–9. https://doi.org/10.15439/2017F526

Davis, J., & Goadrich, M. (2006). The relationship between Precision-Recall and ROC curves. *Proceedings of the 23rd international conference on Machine learning*, 233–240. https://doi.org/10.1145/1143844.1143874

Davison, M., & Sireci, S. (2012). Multidimensional Scaling. https://doi.org/10.1016/B978-012691360-6/50013-6

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. *CVPR09*.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019, May). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [arXiv:1810.04805 [cs]]. Retrieved June 5, 2024, from http://arxiv.org/abs/1810.04805

Diestel, R. (2006). *Graphentheorie* (3., neu bearb. und erw. Aufl). Springer.

Fagiolo, G. (2007). Clustering in Complex Directed Networks [arXiv:physics/0612169]. *Physical Review E*, *76*(2), 026107. https://doi.org/10.1103/PhysRevE.76.026107

Gupta, N. (2021). A Pre-Trained Vs Fine-Tuning Methodology in Transfer Learning. *Journal of Physics: Conference Series*, *1947*(1), 012028. https://doi.org/10.1088/1742-6596/1947/1/012028

Hamilton, W. L., Ying, R., & Leskovec, J. (2018, September). Inductive Representation Learning on Large Graphs [arXiv:1706.02216 [cs, stat]]. Retrieved June 27, 2024, from http://arxiv.org/abs/1706.02216

He, K., Zhang, X., Ren, S., & Sun, J. (2015, December). Deep Residual Learning for Image Recognition [arXiv:1512.03385 [cs]]. Retrieved June 5, 2024, from http://arxiv.org/abs/1512.03385

Howard, J., & Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 328–339. https://doi.org/10.18653/v1/P18-1031

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., & Leskovec, J. (2021, February). Open Graph Benchmark: Datasets for Machine Learning on Graphs [arXiv:2005.00687 [cs, stat]]. https://doi.org/10.48550/arXiv.2005.00687

Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., & Leskovec, J. (2020, February). Strategies for Pre-training Graph Neural Networks [arXiv:1905.12265 [cs, stat]]. Retrieved June 5, 2024, from http://arxiv.org/abs/1905.12265

Hu, Z., Dong, Y., Wang, K., Chang, K.-W., & Sun, Y. (2020). GPT-GNN: Generative Pre-Training of Graph Neural Networks [Publisher: arXiv Version Number: 1]. https://doi.org/10.48550/ARXIV.2006.15437

Hu, Z., Fan, C., Chen, T., Chang, K.-W., & Sun, Y. (2019, May). Pre-Training Graph Neural Networks for Generic Structural Feature Extraction [arXiv:1905.13728 [cs, stat]]. Retrieved June 5, 2024, from http://arxiv.org/abs/1905.13728

Jatnika, D., Bijaksana, M. A., & Suryani, A. A. (2019). Word2Vec Model Analysis for Semantic Similarities in English Words. *Procedia Computer Science*, *157*, 160–167. https://doi.org/10.1016/j.procs.2019.08.153

John, D., & Edmund, W. (2008). Reachability. https://doi.org/10.1093/oi/authority.20110803100406488

Kipf, T. N., & Welling, M. (2017, February). Semi-Supervised Classification with Graph Convolutional Networks [arXiv:1609.02907 [cs, stat]]. Retrieved May 7, 2024, from http://arxiv.org/abs/1609.02907

Kooverjee, N., James, S., & Van Zyl, T. (2022). Investigating Transfer Learning in Graph Neural Networks. *Electronics*, *11*(8), 1202. https://doi.org/10.3390/electronics11081202

Li, J., Shomer, H., Mao, H., Zeng, S., Ma, Y., Shah, N., Tang, J., & Yin, D. (2023, November). Evaluating Graph Neural Networks for Link Prediction: Current Pitfalls and New Benchmarking [arXiv:2306.10453 [cs]]. Retrieved May 22, 2024, from http://arxiv.org/abs/2306.10453

Li, X., Grandvalet, Y., Davoine, F., Cheng, J., Cui, Y., Zhang, H., Belongie, S., Tsai, Y.-H., & Yang, M.-H. (2020). Transfer learning in computer vision tasks: Remember where you come from. *Image and Vision Computing*, *93*, 103853. https://doi.org/10.1016/j.imavis.2019.103853

Louppe, G. (2015, June). Understanding Random Forests: From Theory to Practice [arXiv:1407.7502 [stat]]. Retrieved June 5, 2024, from http://arxiv.org/abs/1407.7502

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013, October). Distributed Representations of Words and Phrases and their Compositionality [arXiv:1310.4546 [cs, stat]]. Retrieved August 3, 2024, from http://arxiv.org/abs/1310.4546

Muhan Zhang, Julian McGinnis, & Fanxu Meng. (2024, June). Facebookresearch/SEAL_ogb [original-date: 2020-10-09T17:47:45Z]. Retrieved June 11, 2024, from https://github.com/facebookresearch/SEAL_OGB

NetworkX developers. (2023, January). NetworkX. Retrieved February 28, 2024, from https://networkx.org/

Nuutila, E., & Soisalon-Soininen, E. (1994). On finding the strongly connected components in a directed graph. *Information Processing Letters*, *49*(1), 9–14. https://doi.org/10.1016/0020-0190(94)90047-7

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019, December). Py-

Torch: An Imperative Style, High-Performance Deep Learning Library [arXiv:1912.01703 [cs, stat]]. Retrieved August 3, 2024, from http://arxiv.org/abs/1912.01703

Prince, S. J. D. (2023). Graph neural networks. In *Understanding deep learning* (pp. 240–267). The MIT Press.

Pykes, K. (2022, October). Hyperparameter Optimization With Comet. Retrieved July 30, 2024, from https://www.comet.com/site/blog/hyperparameter-optimization-with-comet/

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2023, September). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer [arXiv:1910.10683 [cs, stat]]. Retrieved August 13, 2024, from http://arxiv.org/abs/1910.10683

Ruder, S., Peters, M. E., Swayamdipta, S., & Wolf, T. (2019). Transfer Learning in Natural Language Processing. *Proceedings of the 2019 Conference of the North*, 15–18. https://doi.org/10.18653/v1/N19-5004

Saito, T., & Rehmsmeier, M. (2015). The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets [Publisher: Public Library of Science]. *PLOS ONE*, *10*(3), e0118432. https://doi.org/10.1371/journal.pone.0118432

snap-stanford. (2020). Graph Neural Networks. Retrieved April 24, 2024, from https://snap-stanford.github.io/cs224w-notes/machine-learning-with-networks/graph-neural-networks

Stanford, c. (2024, June). Leaderboards for Link Property Prediction. Retrieved June 5, 2024, from https://snap-stanford.github.io/ogb-web/docs/leader_linkprop/

Sundararajan, M., Taly, A., & Yan, Q. (2007). Axiomatic Attribution for Deep Networks.

Taylor, M. E., & Stone, P. (2009). Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research*(10), 1633–1685.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018, February). Graph Attention Networks [arXiv:1710.10903 [cs, stat]]. Retrieved June 27, 2024, from http://arxiv.org/abs/1710.10903

Weihua Hu & Mathias Fey. (2024, June). Snap-stanford/ogb [original-date: 2019-11-22T22:13:57Z]. Retrieved June 25, 2024, from https://github.com/snap-stanford/ogb

West, D. B. (2001). *Introduction to graph theory* (2. ed). Prentice Hall.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2021). A Comprehensive Survey on Graph Neural Networks [arXiv:1901.00596 [cs, stat]]. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(1), 4–24. https://doi.org/10.1109/TNNLS.2020.2978386

Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019, February). How Powerful are Graph Neural Networks? [arXiv:1810.00826 [cs, stat]]. Retrieved June 27, 2024, from http://arxiv.org/abs/1810.00826

Yang, Q., Zhang, Y., Dai, W., & Pan, S. J. (2020a, January). Model-Based Transfer Learning. In *Transfer learning* (pp. 45–57). Cambridge University Press.

Yang, Q., Zhang, Y., Dai, W., & Pan, S. J. (2020b, January). *Transfer Learning* (1st ed.). Cambridge University Press. https://doi.org/10.1017/9781139061773

Yuan, H., Yu, H., Gui, S., & Ji, S. (2022, July). Explainability in Graph Neural Networks: A Taxonomic Survey [arXiv:2012.15445 [cs]]. Retrieved February 29, 2024, from http://arxiv.org/abs/2012.15445

Zhang, M., & Li, P. (2021). Nested Graph Neural Networks [Version Number: 1]. https://doi.org/10.48550/ARXIV.2110.13197

Zhu, Q., Yang, C., Xu, Y., Wang, H., Zhang, C., & Han, J. (2021). Transfer Learning of Graph Neural Networks with Ego-graph Information Maximization. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, & J. W. Vaughan (Eds.), *Advances*

in Neural Information Processing Systems (pp. 1766–1779, Vol. 34). Curran Associates, Inc. https : / / proceedings . neurips . cc / paper _ files / paper / 2021 / file / 0dd6049f5fa537d41753be6d37859430-Paper.pdf

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., & He, Q. (2021). A Comprehensive Survey on Transfer Learning. *Proceedings of the IEEE*, *109*(1), 43–76. https://doi.org/10.1109/JPROC.2020.3004555

Declaration of Authenticity

We hereby declare

- that we have written this performance record independently and on our own;
- that we have correctly cited all text passages not originating from ourself according to common scientific citation rules and that I have clearly mentioned the sources used;
- that we have declared in an index (Table 7.1) all aids used (e.g. AI tools such as chatbots, translation/paraphrasing tools) or programming applications and have indicated their use at the corresponding text passages;
- that we have acquired all intangible rights to any materials we may have used, such as images or graphics, or that these materials were created by us;
- that the topic, the paper or parts of it have not been used in a performance record of another module, unless this has been expressly agreed with the lecturer in advance and is indicated in the paper;
- that we are aware that our work may be checked for plagiarism and for third-party authorship of human or technical origin (artificial intelligence);
- that we are aware that the School of Engineering FHNW will pursue a violation of this declaration of authenticity or its underlying obligations of the study and examination regulations of the School of Engineering FHNW and that disciplinary consequences (reprimand or expulsion from the study program) may result.

Name: Jan Zwicky Thomas Mandelz

Signature:  

Auxiliary Tools

At the end of this thesis, it's important to note the AI tools used in its preparation.

Auxiliary tools	**Usage**	**Affected sections**
ChatGPT	Assisted in linguistically improving, summarising and translating texts.	Whole thesis
ChatGPT Image Generation	Was utilised to generate the title page.	Title page
DeepL Translate	Was utilised for translations from German to English.	Whole thesis
Zotero	Creation of the short references in the text and the bibliography.	Whole thesis

Table 7.1: Declaration and desciption of auxiliary tools.

A Computing Resources

The following tables summarise the different computing nodes and their configurations used in our experiments. Different nodes were selected based on the dataset requirements, primarily constrained by the GPU memory needed to store node embeddings for further computation. The CSCS cluster was primarily utilised for the arXiv CS dataset, the I4DS Slurm cluster for the arXiv w/o CS dataset, and runpod.io to reproduce the ogbl-citation2 dataset models.

Compute-Nodes	GPU	Number of GPUs	VRAM per GPU
gpu22a-b	NVIDIA GeForce RTX 3080	4	10.24GB
gpu23a-d	NVIDIA RTX A4500	4	20.47GB
node14	NVIDIA TITAN RTX	3	24.58GB
node15	NVIDIA GeForce RTX 2080 Ti	4	11.26GB
sdas2	NVIDIA GeForce RTX 2080 Ti	4	11.26GB

Table A.1: Available I4DS Slurm compute nodes with GPU configuration.

Compute-Nodes	GPU	Number of GPUs	VRAM per GPU
piz-daint	NVIDIA Tesla P100	1	16.00 GB

Table A.2: Available CSCS Slurm compute nodes with GPU configuration.

Compute-Nodes	GPU	Number of GPUs	VRAM per GPU
runpod.io	RTX A6000	1	48.00 GB
runpod.io	A40	1	48.00 GB

Table A.3: Available runpod.io compute pods with GPU configuration.

B Extended Discussion

B.1 RQ2: Extensions

This section extends the findings related to section 6.3 by focusing on the NGCN architecture. The fine-tuned model for the NGCN architecture (5.8) does also demonstrate a faster training time compared to the reference model (5.6). For this comparison, we evaluated the test MRR (≈ 0.86) at 10 hours of training time for the reference model. But as illustrated in Figure B.1, the fine-tuned model never reaches the same performance metric, rendering a direct performance comparison invalid.

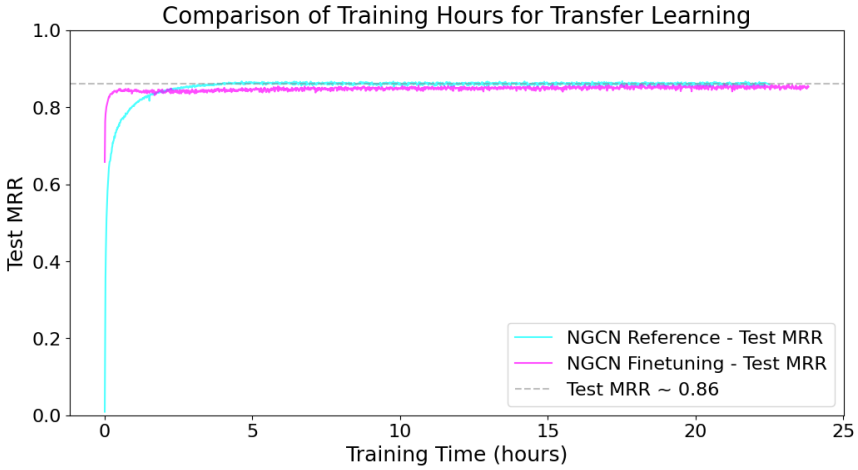


Figure B.1: MRR Curves for the NGCN arXiv CS Reference and NGCN arXiv CS Fine-tuning models plotted against training time in hours. The gray dashed line indicates this chosen test MRR value. There is no intersection between the gray line and the MRR curve of the fine-tuned model making the comparison invalid.

C Exploratory Data Analysis

In this section, we present additional analyses of the exploratory data analysis.

Weakly Connected Components

In this analysis, the size of weakly connected components (as detailed in section 2.1) is examined. If the distributions differ significantly, individual components could potentially be removed during pre-training.

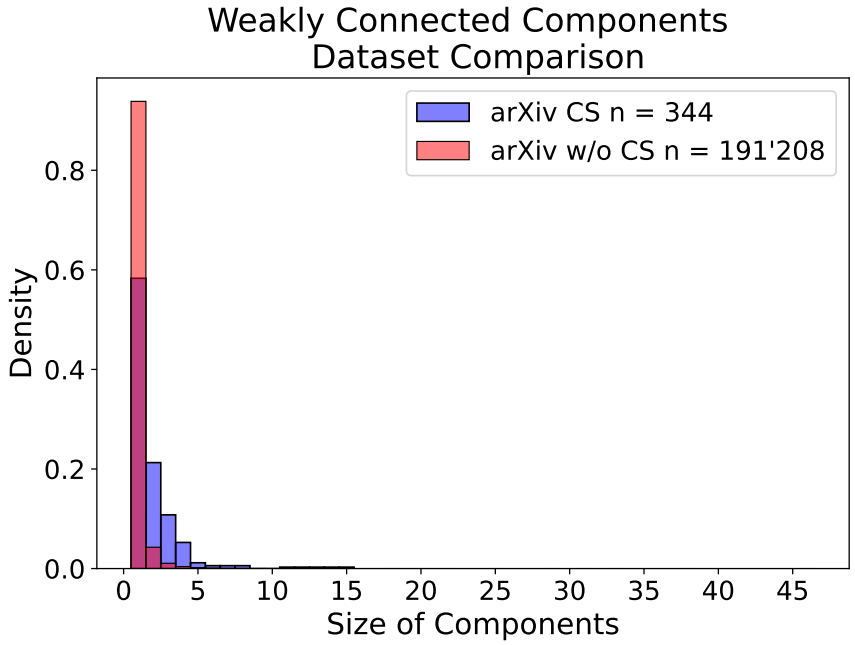


Figure C.1: Visualisation of weakly connected components in the arXiv datasets. Largest weakly connected component for arXiv CS is 168'709 and largest weakly connected component for arXiv w/o CS is 1'113'981

The distributions are similar because neither dataset contains multiple large weakly connected components.

The arXiv w/o CS dataset contains over 150'000 nodes with no edges, which could introduce additional noise during pre-training. However, this noise might also serve as a regularisation factor, hence these nodes will not be removed.

Indegree

The indegree distribution is relevant because it reveals how strongly central papers influence various other papers. This needs to be consistent across datasets because differences in graph structure could make transferability harder.

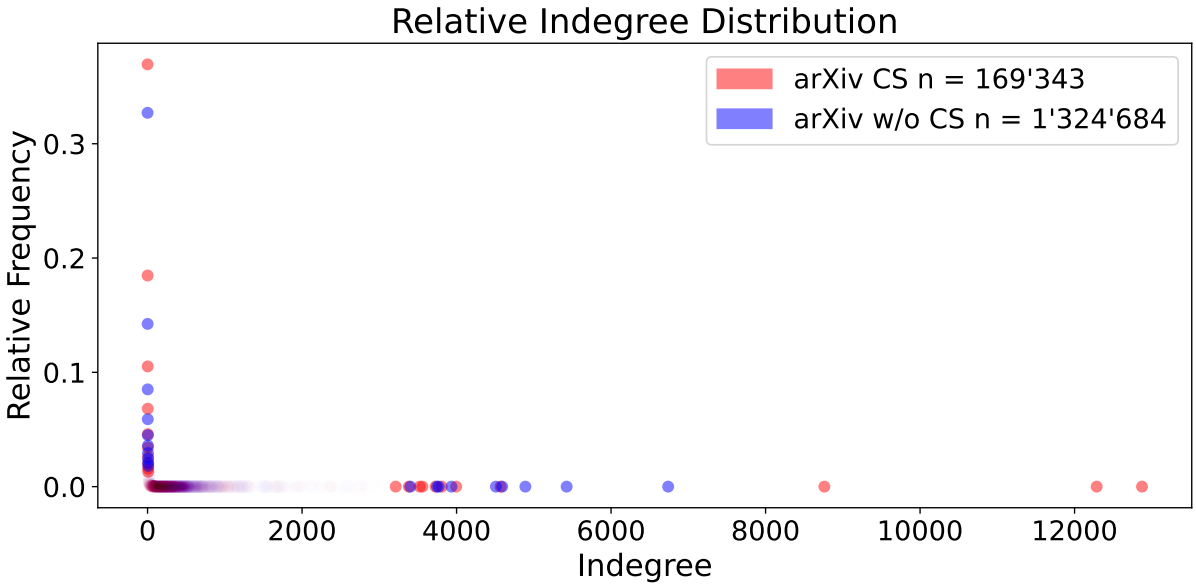


Figure C.2: Relative Indegree Distribution for arXiv CS and arXiv w/o CS datasets: The plot visualises the relative indegree frequencies with blue dots representing the arXiv w/o CS dataset and red dots for the arXiv CS dataset. The peak indegree for the arXiv w/o CS dataset is approximately 12,500, indicating the maximum connectivity of nodes in this dataset.

The distributions are similar in that the frequency indegree is highest at the beginning and then decreases.

However, there are also differences, such as fewer central indegree nodes. This should be considered when analysing the models. Depending on the need, the graph for arXiv w/o CS might need adjustment to increase similarity. Nevertheless, the similarity in node indegrees is sufficient to enable transferability.

Clustering Coefficients

The clustering coefficient indicates how well individual nodes are interconnected. This is a way to compare the structure of a graph. Ideally, both graphs should have a similar distribution of clustering coefficients.

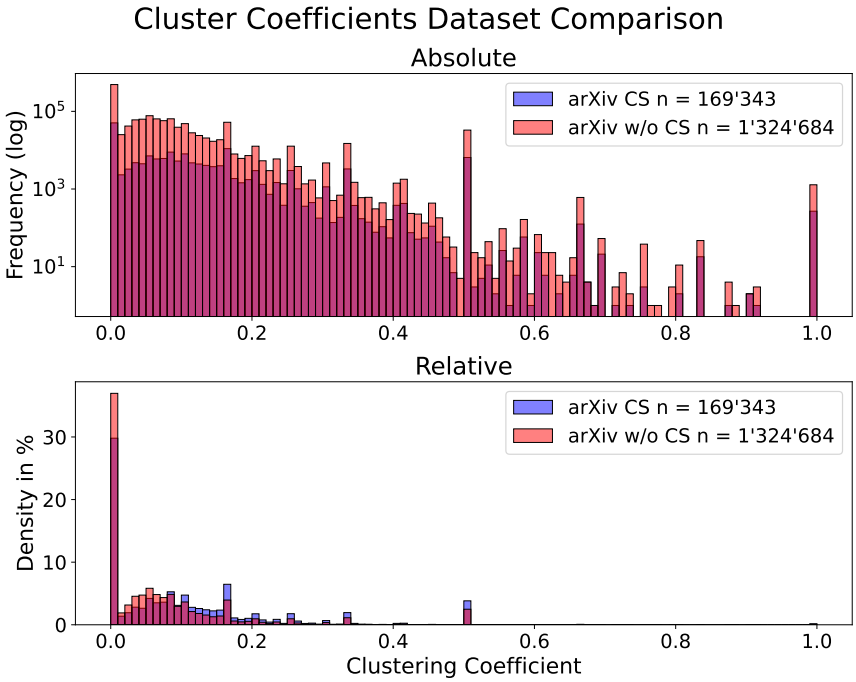


Figure C.3: Clustering Coefficient Distribution of the arXiv CS and arXiv w/o CS datasets: This histogram compares the clustering coefficients, with blue bars representing the arXiv CS dataset and red bars for the arXiv w/o CS dataset. The arXiv CS dataset exhibits higher clustering coefficients, reflecting a denser or more interconnected node structure, with an average coefficient of 0.1073 compared to 0.0754 for the arXiv w/o CS dataset.

Most clustering coefficients in both datasets are less than or equal to 0.5. However, the arXiv CS dataset has higher clustering coefficient values, with an average of 0.1073 compared to 0.0754 for the arXiv w/o CS dataset.

The differences between the datasets may result from the removal of links (citations) from Computer Science to other fields in the arXiv CS dataset. If pre-training does not yield satisfactory results, this aspect could be further investigated. For example, individual fields such as Mathematics, Economics, and Physics could be used in separate networks for pre-training, with edges deleted in a similar manner.

Common Neighbor

If the distributions of random edges are very similar to the actual edges, the baseline model using the Common Neighbor algorithm (CN) would be redundant. The goal is to verify if the baseline model is worthwhile.

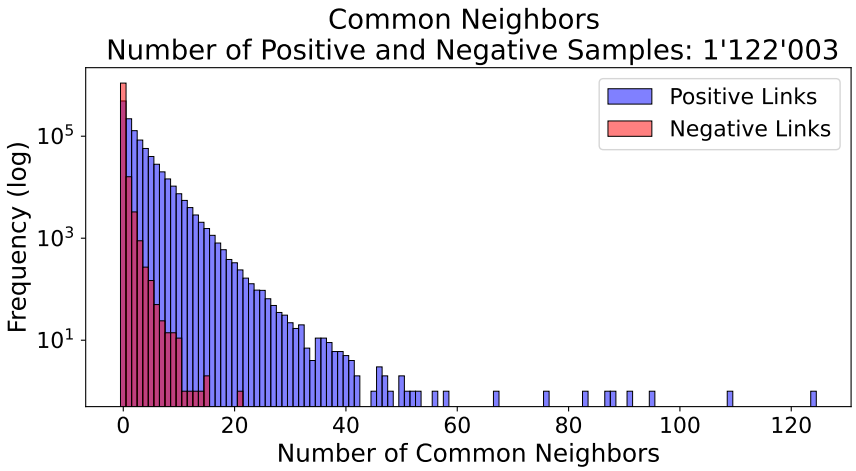


Figure C.4: Common Neighbor Distribution of Positive and Negative Edges in the arXiv CS Dataset: This histogram displays the distribution of common neighbors, with blue bars indicating positive edges (actual links) and red bars representing negative edges (absence of links). The graph shows that positive edges generally have a higher number of common neighbors compared to negative edges, underscoring the effectiveness of the Common Neighbor algorithm as a viable baseline model for predicting link formation.

The distributions are very different. Therefore, it can be assumed that the Common Neighbor algorithm works as a baseline. However, there is a large proportion with very few common neighbors, indicating that the Common Neighbor model will not perform well.

Cosine Similarity

Figure C.3 provides an initial understanding of the influence of features. If a paper cites another paper, it should be more similar in terms of features compared to a random paper. Additionally, this provides a basis to qualitatively assess the later results and sets the groundwork for a baseline heuristic. Thus, it is examined whether the similarity scores of a paper citing another paper are higher than those of a random paper. This should be the case for both datasets.

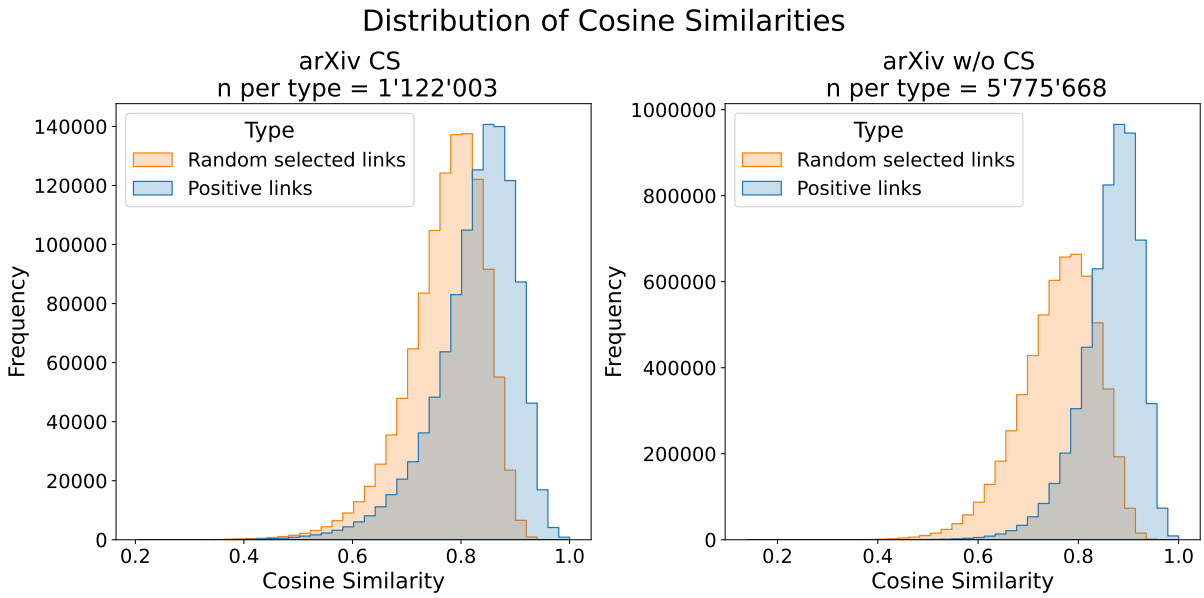


Figure C.5: Cosine Similarity of Positive and Negative Edges in the arXiv CS Dataset: This histogram compares cosine similarity scores for positive (actual citations) and negative (non-citations) edges. Notably, 50% of the papers from the arXiv w/o CS dataset were randomly excluded to facilitate computation. The plot confirms that papers citing each other generally exhibit higher similarity than random pairs, with more pronounced differences observed in the non-CS dataset. This analysis forms a baseline for evaluating the effectiveness of similarity-based heuristics in link prediction models.

It can be confirmed that the similarity scores of a paper citing another paper are higher than those of a random paper. It is also interesting to see that the differences are larger in the arXiv w/o CS dataset. This could be because the different research fields are more distinct from each other compared to Computer Science.

To maintain this similarity, negative samples during pre-training could be selected more often or exclusively within the same research group.

Reachability

Figure C.6 displays the complete reachability analysis plot without the visual cut-off in Figure 3.3.

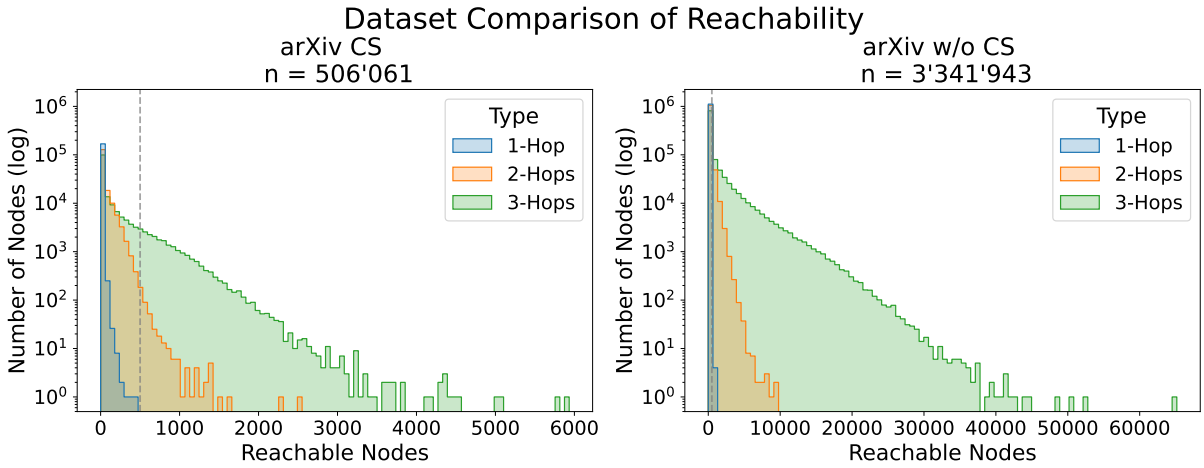


Figure C.6: Reachability in the arXiv datasets. The colors represent different hop distances: blue for 1-hop, orange for 2-hops, and green for 3-hops. Nodes are counted only once for each k-hop. Our arbitrary threshold of 500 papers is marked with a dashed gray line. Be advised the y-axis is logarithmic scaled.

D Qualitative Evaluation

This section provides additional descriptions and outlines the qualitative evaluation conducted at both the model and sample levels.

D.1 Model Level Evaluation

For an in depth qualitative evaluation on a model level (all predicted links) we created visualisations which compare the rank metric to various graph features or cosine similarities.

The figures (Figure D.1, Figure D.2, Figure D.3, Figure D.4) are used to compare the Reciprocal Rank (RR) with the features Indegree, Outdegree, Clustering Coefficient, and the average Cosine Similarity of the neighbors to the source node. Additionally, Figure D.5 shows the prediction probability of the model and the cosine similarity for all links on the first rank. The purpose of these figures is to evaluate different aspects of the model as accurately as possible. This helps to identify the model's strengths and weaknesses.

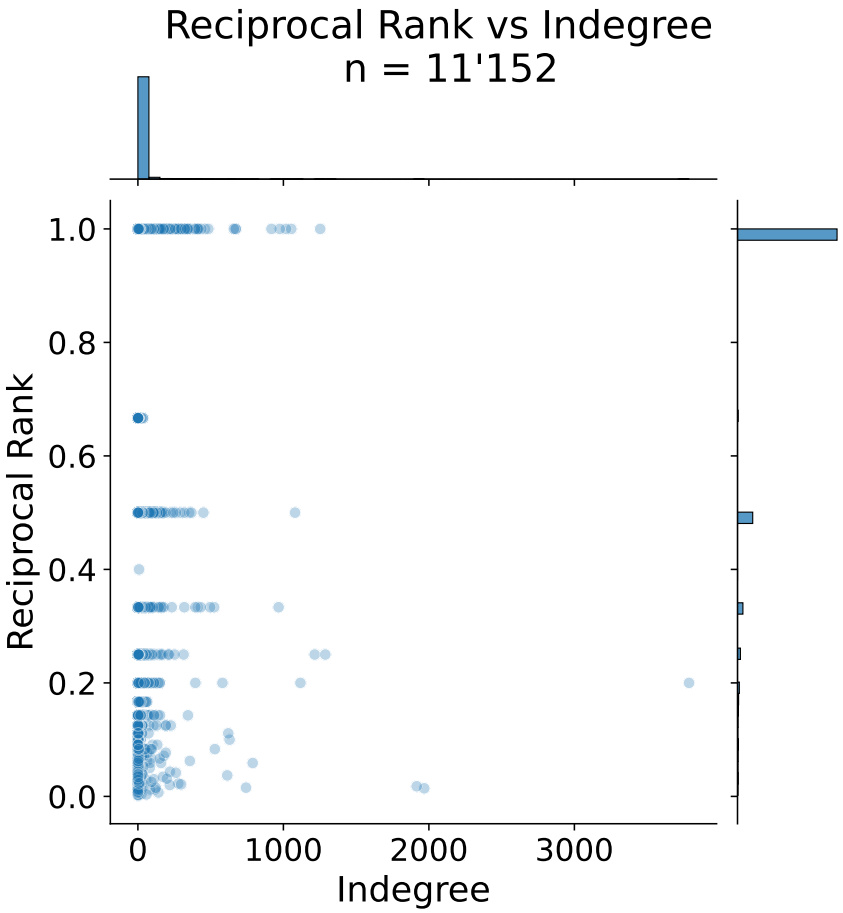


Figure D.1: Example figure of reciprocal rank vs. indegree: This scatter plot visualises the relationship between reciprocal rank (RR) and the indegree of nodes within the model.

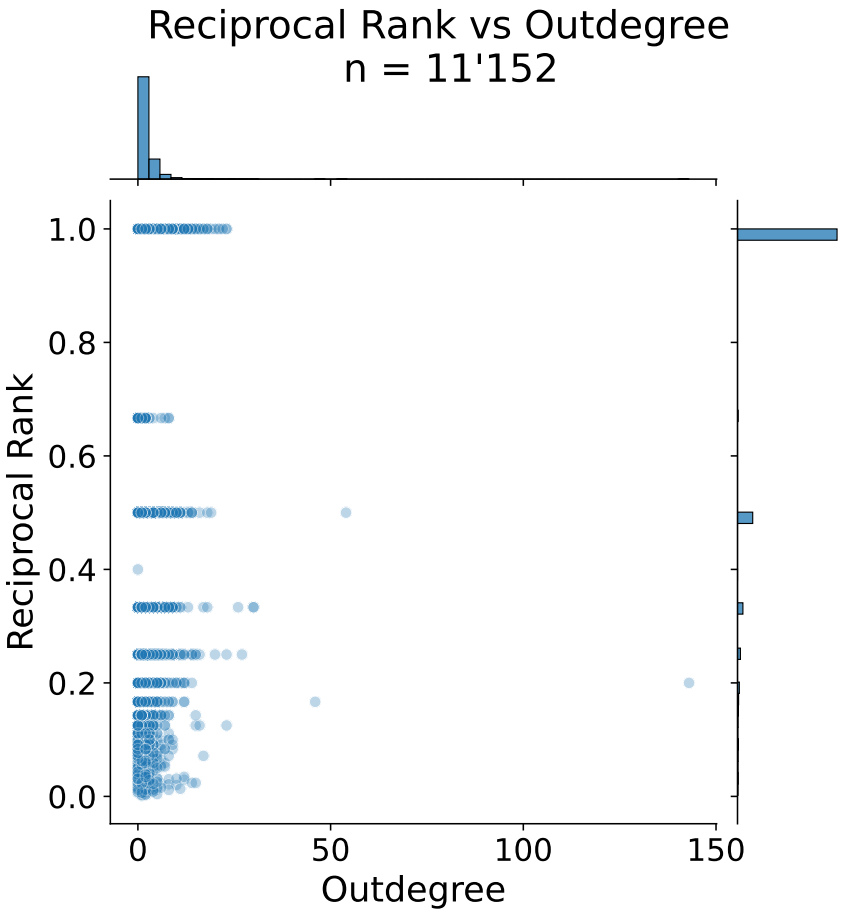


Figure D.2: Example figure of reciprocal rank vs. outdegree: This graph displays the relationship between reciprocal rank (RR) and the outdegree of nodes. The visualisation helps in assessing whether the outdegree of a node affects the efficacy of the model's predictions, thus guiding potential model adjustments.

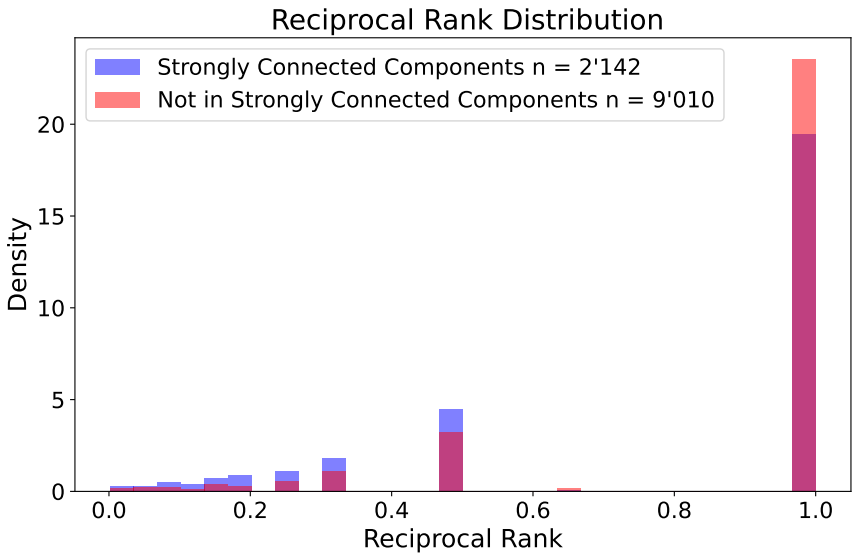


Figure D.3: Example figure of reciprocal rank (RR) vs. clustering coefficient: This chart plots the RR against the clustering coefficient for each predicted link.

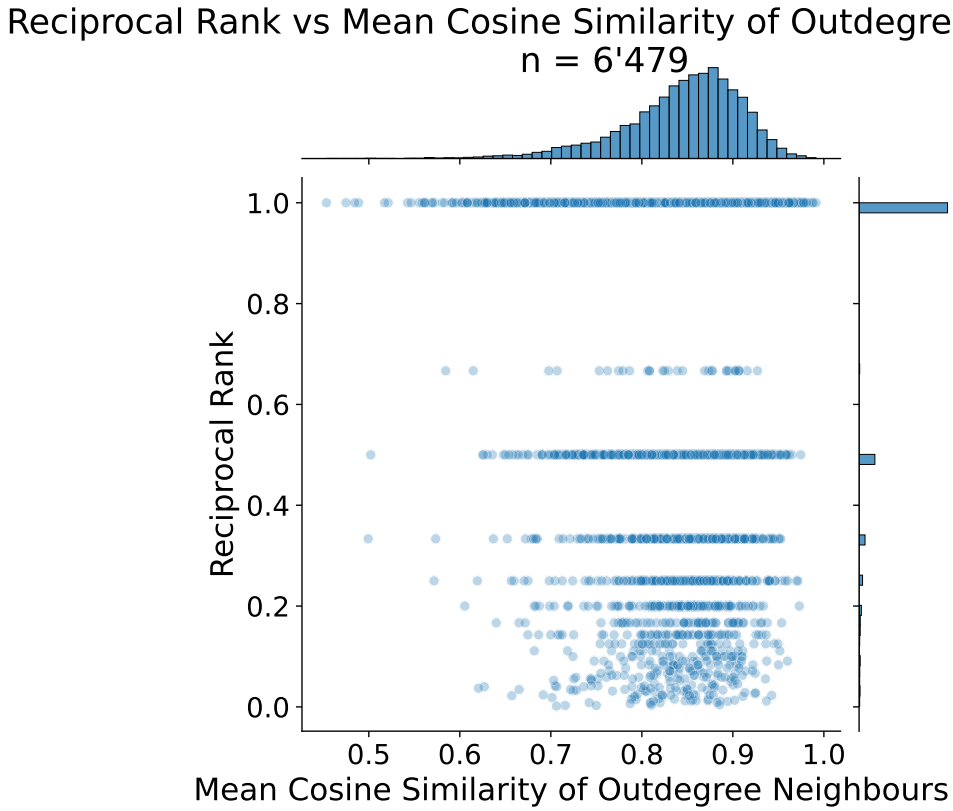


Figure D.4: Example figure of reciprocal rannk (RR) vs. cosine similarity of neighbors: This analysis illustrates the relationship between RR and the average cosine similarity of neighbors to the source node. This measure provides insights into how similarity between nodes neighbors influences the model's predictions.

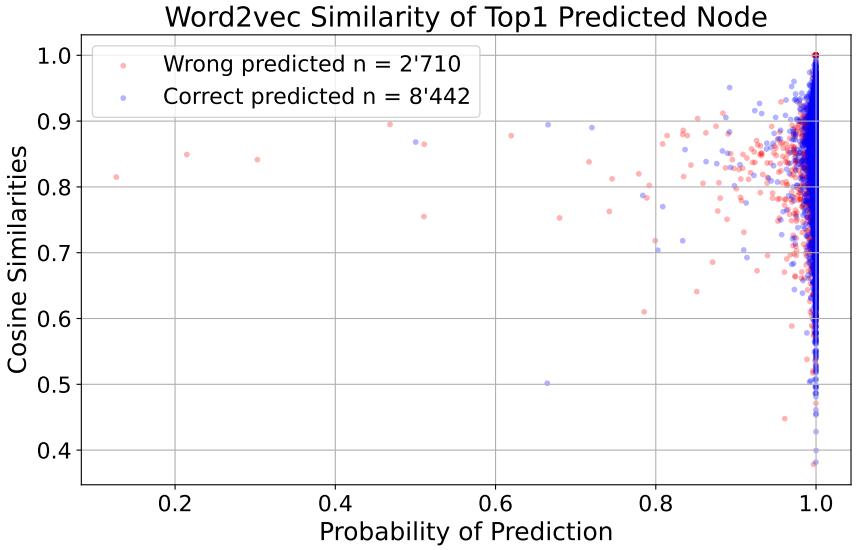


Figure D.5: Word2Vec Similarity and Prediction Probability for Top-1 Predicted Node: This plot compares the model's prediction probability and the Word2Vec similarity for the top-1 predicted node.

D.2 Sample Level Evaluation

For an in depth qualitative evaluation on a sample level (a single predicted link) we tried to combine an explainable AI approach in combination with the similarities of the Word2Vec vectors (content of the papers). All of this should be represented in a single visualisation. The strengths of this approach are the incorporation of a visual representation of distance (similarity based) between nodes which are used for the prediction in the GNN model as well as the explainable AI algorithm which adds the importance for the prediction visually.

Plainly put, the visualisation shows the importance of nodes based on their distance, represented by similarity, for all nodes used in the prediction, emphasising the local context.

This highly complex visualisation has also the weakness of combining many algorithms and is therefore very challenging to read and interpret. Nonetheless, we show and describe the visualisation in the following paragraphs.

Figure D.6 can be divided into two parts. The right image shows the positive target node, while the left image displays the negative link with the highest probability. The source node is the same in both images. All calculations for the two images were performed separately. The following calculations are relevant for each image.

The selection of nodes includes those within a 1-hop or 2-hop distance from the source or target node (for an explanation, see section 2.1).

The positions of the nodes were determined using dimensionality reduction from the 128 Word2Vec dimensions down to two dimensions. Multidimensional Scaling (MDS) (Davison and Sireci, 2012) was used for this purpose. MDS creates a matrix of cosine similarities that contains the similarities between the nodes. Then, the positions of the objects in the two-dimensional space are determined to best match the original distances. This process is iteratively optimised using STRESS (Standardised Residuals Sum of Squares) until the discrepancies are minimised. Additionally, the average and maximum errors of the dimensionality reduction were calculated to assess the reliability of the interpretation.

The color of the nodes was determined by their importance. Node importance can be explained using various explainability approaches, as demonstrated by Yuan et al. (2022). Borile et al. (2023) have shown which approaches are effective for link prediction. Based on this research, Integrated Gradients was chosen as the most suitable method for assessing node importance.

For Integrated Gradients, a neutral baseline is established, in this case, a zero vector. Multiple intermediate steps are interpolated between this baseline and the actual input. For each of these intermediate steps, the gradient of the prediction with respect to the input is calculated. The gradient indicates how sensitive the model's prediction is to changes in the input.

The gradients of all intermediate steps are then summed and averaged. This integration of gradients provides the attributions for each input feature. The result indicates the importance of each feature for the model's prediction. The larger the integrated gradient of a feature, the more important it is for the prediction (Sundararajan et al., 2007).

Since integrated gradients calculate the importance of the Word2Vec features, the importances of all Word2Vec features need to be summed to determine the node importance in our visualisation. Additionally, the highest node importance was scaled to 1.

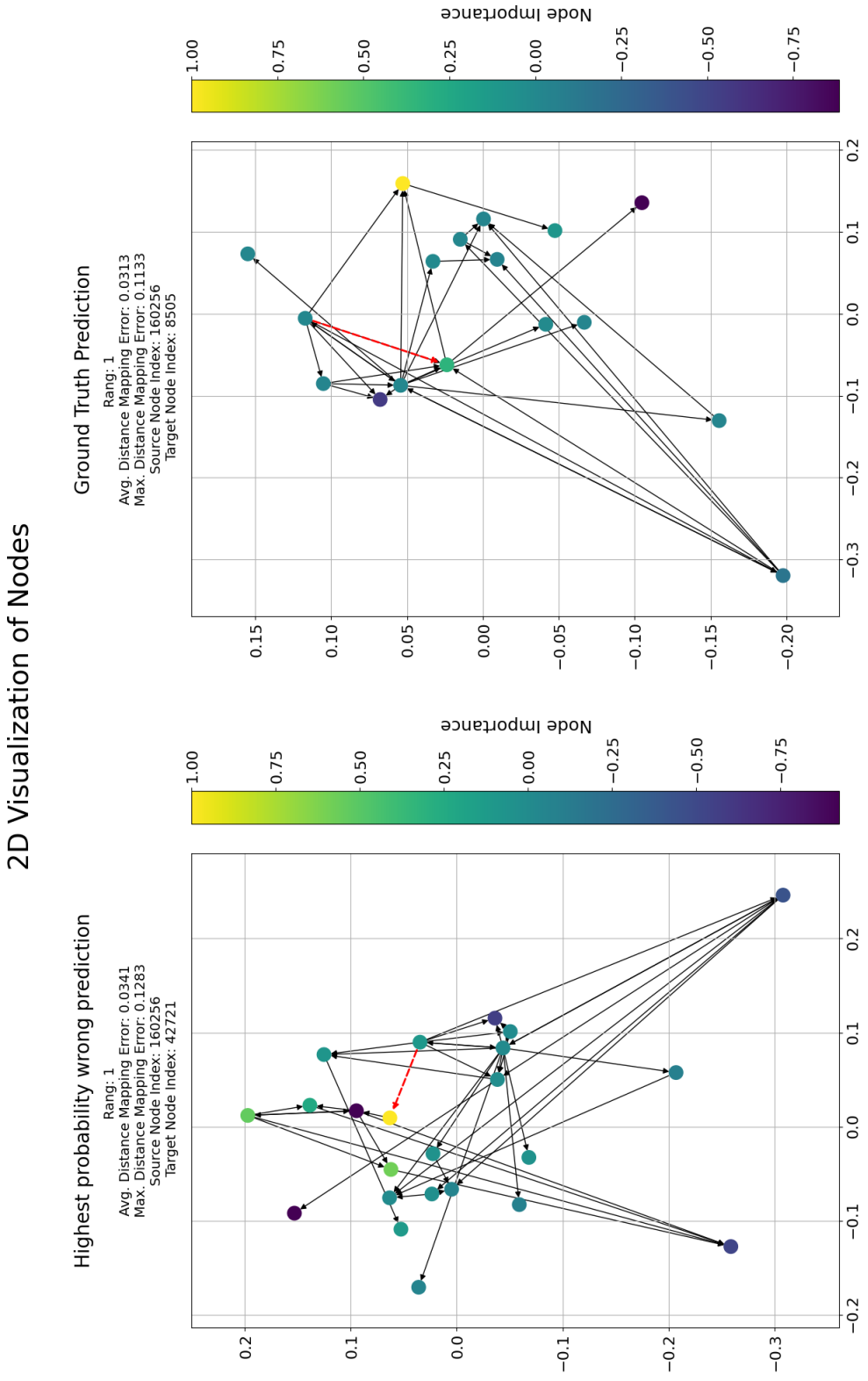


Figure D.6: Example figure of 2D visualisation of nodes within 2-hop computational graphs. The left image shows the positive target node, while the right image displays the negative link with the highest probability, with the same source node for both images. Node positions were determined using MDS to reduce the 128 Word2Vec dimensions to two dimensions. The color of the nodes indicates their importance, calculated using Integrated Gradients, where red denotes the predicted link.

E Hyperparameter

We list all final hyperparameters used for our models in Table E.1 and Table E.2.

Model	Epoch	Batch size	Learning rate	Experiment link
Reference:				
GCN arXiv CS reference (5.3)	2071	38349	0.0004	comet
NGCN arXiv CS reference (5.6)	1983	9511	0.0005	comet
Pre-train:				
GCN arXiv w/o CS Pre-training (5.4)	199	38349	0.0004	comet
NGCN arXiv w/o CS Pre-training (5.7)	178	11862	0.0008	comet
Fine-tuning:				
GCN arXiv CS Fine-tuning (5.5)	1896	28530	0.0003	comet
NGCN arXiv CS Fine-tuning (5.8)	1397	7853	0.0008	comet

Table E.1: Final hyperparameter used for training the respective models, learning rates are rounded to the fourth decimal, actual values are found in comet-ml experiment.

Model	Hidden channels	Number of Layers	Freezing layers	Experiment link
Reference:				
GCN arXiv CS reference (5.3)	512	2	-	comet
NGCN arXiv CS reference (5.6)	384	2	-	comet
Pre-train:				
GCN arXiv w/o CS Pre-training (5.4)	512	1	-	comet
NGCN arXiv w/o CS Pre-training (5.7)	384	2	-	comet
Fine-tuning:				
GCN arXiv CS Fine-tuning (5.5)	512	2	False	comet
NGCN arXiv CS Fine-tuning (5.8)	384	2	False	comet

Table E.2: Final hyperparameter used for training the respective models, learning rates are rounded to the fourth decimal, actual values are found in comet-ml experiment.