UNIVERSITY OF CALIFORNIA
RIVERSIDE

Exploring Frontiers in Graph Learning

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

William L. Shiao

September 2024

Dissertation Committee:

Dr. Evangelos E. Papalexakis, Chairperson
Dr. Greg Ver Steeg
Dr. Yue Dong
Dr. Neil Shah

The Dissertation of William L. Shiao is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

This is the culmination of an 8-year journey at UC Riverside. It began with my undergraduate degree, continued to my Master's, and finally ended with my Ph.D. These years have truly shaped the trajectory of my life in countless ways and would not have been possible without the unwavering support and love of my friends and family. I owe so much to the people who have helped me through the years, and I would like to take this opportunity to thank each of them.

First and foremost, I would like to express my deepest gratitude to my advisor, Evangelos (Vagelis) Papalexakis. Our paths crossed during my second year as an undergraduate, and we have worked together ever since. At the time, pursuing a Ph.D. had never crossed my mind, but Vagelis's enthusiasm and encouragement convinced me otherwise. After 7 years of working together, I can truly say that he is the kindest and most supportive person I have had the pleasure to work with. It's hard to count the number of times I joined a meeting with Vagelis where I was overwhelmed and fully convinced the project was doomed to fail, only to come out an hour later full of optimism and hope. On top of that, he was also always only a Slack message away — ready to help with anything from listening to a practice presentation or proofreading a particularly difficult email. His generosity, as well as the freedom and flexibility he offered, made it possible to travel and still enjoy life throughout the course of my degree.

I would also like to thank the members of my committee: Yue Dong, Greg Ver Steeg, and Neil Shah, for their invaluable feedback and guidance. I had the pleasure of being managed and mentored by Neil during the 12 months I interned at Snap. There, I learned

about the wondrous world of GNNs — now the focus of this thesis — and greatly improved my research skills. Neil and my other mentors, Yozen Liu and Tong Zhao, were always there when I needed to discuss something. Their guidance was instrumental to my work, and their kindness made even the dreary Seattle winter enjoyable. There, I had the chance to meet some amazing co-workers, collaborators, and friends: Zhichun Guo, Clark Ju, Yushun Dong, Vijay Prakash Dwivedi, Ankit Bara, Matt Kolodner, Shubham Vij, Xiaotian Han, and Wei Jin. From collaborating on projects to going on fishing trips and even helping me build a wall from Bubly cans (much to the chagrin of Snap workplace employees), each of them made my internships much more fun and productive.

I would like to thank my labmates, with whom I spent the majority of my waking hours (and possibly several non-waking ones): Sara Abdali, Dawon Ahn, Biqian Cheng, Pravallika Devineni, Negin Entezari, Ekta Gujral, Rutuja Gurav, Miguel Gutierrez, Yiran Luo, Ravdeep Pasricha, Het Patel, Uday Singh Saini, Yorgos Tsitsikas, and Yunshu Wu. Their companionship and support made the long hours in the lab much more enjoyable. I would also like to thank the other residents in the lab: Satish Chandran, Shahrzad Haji Amin Shirazi, and Yahya Sattar, for their camaraderie and support. My work would not be possible without my collaborators: Kevin Chan, Jia Chen, Benjamin Miller, Tina Eliassi-Rad, Zubair Qazi, and Paul Yu, who brought fresh perspectives and expertise to our projects. I would also like to thank the UCR staff: Victor Hill, Vanda Yamaguchi, Mindi Mobley, Bart Kats, Sean Mahoney, and Tara Barthol, for providing valuable administrative support and computational resources.

Next, I would like to thank my family. My father always supported my decision to pursue a Ph.D., but he made me think through my decision by having me sit down and calculate the opportunity cost compared to working. This exercise ensured that pursuing a Ph.D. was what I genuinely wanted, and it helped me refocus and get back on track whenever I wavered during my journey. Throughout my life, he has been an infallible source of sage advice and wisdom.

My mother was also an important source of strength, letting me know that there was always a warm hug, delicious meal, and clean bed awaiting me at home, should I need it. My brother, Theo, has been my best friend and closest confidant, providing a listening ear and a shoulder to lean on during both the good times and the bad. Our shared experiences, from heart-to-hearts to the occasional fistfight, have forged an unbreakable bond, and his support has been invaluable throughout my journey. My many cousins — Timothy and Aaron Chang, Pim Tamavimoke, Matthew, Johnathan, Sarah, and Michael Capparelli — who always managed to make time when Theo and I were back home, made returning home extra enjoyable, and really allowed me to relax. Whether through a quick taco run, a 6-hour game of *Terra Mystica*, or a backpacking trip to the Channel Islands, they helped me de-stress and create unforgettable memories.

I would like to thank my friends, many of whom have been there since my undergraduate days. There are the "Oracles" (although, despite the name, our stock performance was anything but Oracle-like): JiHwan Kim, Joshua Sun, Paris Hom, Nikhil Gowda, Aditya Acharya, Erin Wong, Brandon Lam, Eric Ong, and John Shin. We became close because of a spur-of-the-moment trip to Hawaii, but our friendship has only strengthened since then.

Many of the most memorable events throughout my Ph.D. have occurred on trips with them, be it riding ATVs in Vegas, getting lost in the snow on Mt. Rainier, or almost dying while surfing in Hawaii. I would also like to thank the great housemates (and now friends) I've had over the years who made me happy to return home: Brandon Lam, Satish Chandran, Chandan Sidhu, Timothy Koo, Patrick Lam, Eric Gong, Garret Lim, Edward Huh, and Jackson Huang.

Additionally, I owe my gratitude to my friends for their support: Jay Park, Ivan Liang, Carolyn Kong, Niharika Battula, Amanda Xaypraseuth, Brittney Mun, Daniel Stinson-Diess, Kennen DeRenard, Jon Chee, Aaroh Mankad, Ed Zabrensky, Colin Lee, David Feng, Enoch Chang, Andre Castro, Mohit Gupta, Donald Morton, Peter Cunha, and Anthony Stenzel. Each of them, in their own unique way, has contributed to my personal and professional growth.

Finally, I would like to thank the others whom I have forgotten to name and who have helped me along the way. I am forever grateful for the support, guidance, and love I have received from so many people during this incredible 8-year journey at UC Riverside.

Funding Acknowledgments

This research was supported in part by the National Science Foundation under CAREER grant no. IIS 2046086 and the CREST Center for Multidisciplinary Research Excellence in Cyber-Physical Infrastructure Systems (MECIS) grant no. 2112650. Additional support was provided by the Agriculture and Food Research Initiative Competitive Grant no. 2020-69012-31914 from the USDA National Institute of Food and Agriculture.

Several UCR coauthors were also sponsored by the Combat Capabilities Development Command Army Research Laboratory under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The coauthors of work that appears in this thesis, Benjamin A. Miller and Tina Eliassi-Rad, were supported in part by the United States Air Force under Air Force Contract No. FA8702-15-D-0001.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Combat Capabilities Development Command Army Research Laboratory, the United States Air Force, the U.S. Government, or any other sponsoring agency. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

Some of this work was also funded by and completed during my internship at Snap Inc. We would also like to thank UCR Research Computing and Ursa Major for the Google Cloud resources provided to support this research.

In loving memory of 爷爷, 婆婆, and 公公, whose journeys ended before mine.

ABSTRACT OF THE DISSERTATION

Exploring Frontiers in Graph Learning

by

William L. Shiao

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, September 2024
Dr. Evangelos E. Papalexakis, Chairperson

Graphs have been used to model many different types of data, ranging from social networks to the human brain. We can then formulate real-world problems as tasks (like link prediction or node classification) on these graphs. For example, item recommendation can be viewed as link prediction on a user-item purchase graph and bot detection as node classification on a social interaction graph. Traditionally, these graph tasks have been solved through statistical or spectral analysis. However, relatively recent works have proposed the idea of Graph Neural Networks (GNNs), which allow us to solve these problems in a highly scalable and effective manner.

In our recent work, we focus on exploring three frontiers in graph learning. First, we bridge ideas from different domains and apply them to graph learning to define new models and methods. Second, we work on solving tasks in a fast and space-efficient manner, improving their practical utility. Finally, we revisit and challenge established concepts in graph learning to see if they are really true. This dissertation summarizes four pieces of work, each of which fits into two or more of the aforementioned frontiers.

The first work in our dissertation, TenGAN, examines the task of multiplex graph generation - generating graphs that have multiple views, each with different edges but the same nodes. This is a task that is typically solved with statistical preferential attachment models. However, these models are inflexible and require the explicit modeling of desired graph attributes. Modeling these multiplex graphs with a naive generative model is also difficult due to the large number of parameters required. We propose borrowing ideas from tensor decompositions to implicitly compress the parameters in the network, greatly reducing the number of parameters required.

Our second work shifts its focus towards efficient link prediction with GNNs. Currently, most scalable method is node-based link prediction, where we compute node embeddings for candidate pairs and use them to compute the probability of a link existing between them. Most existing state-of-the-art methods rely on contrastive learning, which uses both positive samples (e.g., neighbors) and negative samples (e.g., non-neighbors). However, these negative samples are often expensive to obtain. A recent group of methods, dubbed non-contrastive learning, have been proposed to avoid the expensive negative sampling step but have only been evaluated on node classification tasks. In this work, we perform a detailed benchmark of existing non-contrastive methods on link prediction, discover a key limitation, and mitigate it by proposing a simple modification. This results in an improvement of up to 120

Our third work, CARL-G, also focuses on speeding up self-supervised node representation learning. We do this by observing some similarities between contrastive learning and clustering. We then propose a new framework and loss function that reformulates node

representation learning as a clustering problem. Our key contribution is finding that an existing class of unsupervised clustering evaluation metrics known as Clustering Validation Indices (CVIs) can serve as effective substitutes for existing contrastive losses. With the choice of specific CVIs, we can reduce the cost of computing the loss function to linear time (in contrast to the quadratic time complexity of most contrastive losses). Then, also taking advantage of decades of research in speeding up clustering, we can compute better node representations in much less time. CARL-G trains 79× than the best-performing node classification baseline and 1,500× faster than the best-performing node clustering and similarity search baseline.

Our fourth and final work focuses on improving the inductive capabilities of recommendation systems in a fast and space-efficient manner. Recommendation models typically store an embedding for each unique user and item in embedding tables. However, when performing inference with real-world recommendation systems, we often encounter users/items that were not present during training. The most common solutions to this are to either randomly use an embedding from an out-of-vocabulary (OOV) embedding table or a fixed vector like the mean embedding. Our work challenges the assumption that these simple methods are sufficient, and we explore 9 different OOV embedding methods on 5 different models to find more effective approaches. We find that locality-sensitive-hashing (LSH) based methods generally perform better than the competing methods resulting in a 3.74% mean improvement over the industry-standard baseline.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Graphs are capable of representing a wide array of real-world data, from social networks to user purchases. For example, a social network could be represented as a graph where nodes represent users and edges represent users adding each other as friends. We could then perform various tasks on this graph — for example, node classification could be used to find "bots" and link prediction could be used to recommend potential friends.

Traditionally, these graph-based tasks have been tackled using statistical or spectral analysis techniques. However, Graph Neural Networks (GNNs) have steadily increased in popularity over recent years. These GNNs provide a highly scalable and effective framework for solving various graph tasks. In the recommendation systems (which can also be framed as a link prediction task) community, neural networks have also seen increasing popularity.

1.1 Research Questions

In this thesis, we examine four developing and relatively under-explored areas in graph learning:

1. **Multiplex Graph Generation**: Multiplex graphs are more complex and harder to generate than traditional graphs. The existing methods are purely statistical — how can we leverage GNNs and other modern advancements to improve multiplex graph generation?

2. **Non-Contrastive Link Prediction**: Non-contrastive learning for GNNs has recently become popular. However, these models focus solely on node classification, not link prediction. Do they work for link prediction, and if not, can we improve them?

3. **Contrastive Learning and Clustering on Graphs**: Contrastive learning on graphs has many similarities to traditional clustering. However, clustering has the additional benefit of having been studied for decades longer. Can we connect graph contrastive learning to clustering and use that connection to improve graph contrastive learning?

4. **Improving OOV Support in Recommendation Systems**: Out-of-vocabulary (OOV) values are categorical values first seen at inference time. This means that we may have no associated embedding for those values. The industry-standard way to handle this is by randomly selecting a row from an embedding table for OOV values. Can we improve upon this approach without paying a high computational price?

1.1.1 Frontiers in Graph Learning

We answer the above questions by exploring three frontiers in graph learning:

1. **Bridge Ideas**: We bridge ideas from different domains and apply them to graph learning to define new models and methods. For example, tensor decompositions have been shown to compress large tensors effectively and are frequently applied to multi-view graphs. We can then combine those ideas with modern GNNs to create new models.

2. **Efficiency**: Can we solve tasks in a time and space-efficient manner? Many models work well on smaller graphs but quickly fail as graphs get larger. When solving each of the above research questions, we take care to consider the efficiency of the methods.

3. **Revisiting & Challenging Established Concepts**: Are the existing assumptions in the field correct? Can we propose a better way to frame the problem?

Below, we elaborate on each of the above research questions.

1.1.2 Multiplex Graph Generation

Chapter 3 examines the task of multiplex graph generation — generating graphs with multiple views, each with different edges but the same nodes. This is a task that is typically solved with statistical preferential attachment models. However, these models are inflexible and require explicitly modeling desired graph attributes (e.g., degree distribution or clustering coefficient). One approach that has been proposed for generating traditional graphs is to use a neural network [19, 52, 41]. However, this is non-trivial to extend to multiplex

graphs for two reasons. First, multiplex graphs must also consider the relationship between modes, making it difficult to adapt graph models directly. Second, naively modeling these multiplex graphs with a neural network is difficult due to the large number of parameters required (compared to a traditional graph). We propose a novel method, TenGAN, that borrows ideas from tensor decompositions to implicitly compress the parameters in the network, greatly reducing the number of parameters required. This also allows us to preserve the interactions between different modes.

1.1.3 Non-Contrastive Link Prediction

Chapter 4 focuses on efficient link prediction with GNNs. Currently, the most scalable method for link prediction is node-based link prediction, where we compute node embeddings for candidate pairs and use them to compute the probability of a link between them. Most existing state-of-the-art methods rely on contrastive learning, which uses both positive samples (e.g., neighbors) and negative samples (e.g., non-neighbors). However, these negative samples are often expensive to obtain. This is due to the format in which graphs are typically stored — we typically store them as adjacency lists (or sparse adjacency matrices) where we can retrieve the neighbors of a node in $\mathcal{O}(1)$ time but may need $\mathcal{O}(n)$ time to retrieve its non-neighbors. A recent group of methods, dubbed non-contrastive learning, have been proposed to avoid the expensive negative sampling step but have only been evaluated on node classification tasks. In this chapter, we perform a detailed benchmark of existing non-contrastive methods on link prediction, discover a key limitation, and mitigate it by proposing a simple modification. This results in an improvement of up to 120% in Hits@50 compared to existing non-contrastive methods and a 14× speedup over contrastive methods.

1.1.4 Contrastive Learning and Clustering on Graphs

Chapter 5 also focuses on speeding up self-supervised node representation learning. We do this by observing some similarities between contrastive learning and clustering. We draw a theoretical connection between contrastive learning and clustering. We then use this knowledge to propose a new framework and loss function that reformulates node representation learning as a clustering problem. Our key contribution is finding that an existing class of unsupervised clustering evaluation metrics known as Clustering Validation Indices (CVIs) can serve as effective substitutes for existing contrastive losses. With the choice of specific CVIs, we can reduce the cost of computing the loss function to linear time (in contrast to the quadratic time complexity of most contrastive losses). Then, also taking advantage of decades of research in speeding up clustering, we can compute better node representations in much less time. CARL-G trains 79× than the best-performing node classification baseline and 1,500× faster than the best-performing node clustering and similarity search baseline.

1.1.5 Improving OOV Support in Recommendation Systems

Chapter 6 focuses on improving the inductive capabilities of recommendation systems in a fast and space-efficient manner. Recommendation models typically store an embedding for each unique user and item in embedding tables. However, when performing inference with real-world recommendation systems, we often encounter users/items that were not present during training. The most common solutions are randomly using an embedding from an out-of-vocabulary (OOV) embedding table that is updated during training with

synthetically created OOV values or a fixed vector like the mean user/item embedding. Both of these intuitively seem like sub-optimal approaches since one would expect that similar users should have similar embeddings. Our work challenges the established assumption that these simple OOV handling methods are sufficient, and we explore 9 different OOV embedding methods on 5 different models to find more effective approaches. We find that locality-sensitive-hashing (LSH) based methods generally perform better than the competing methods, resulting in a 3.74% mean improvement over the industry-standard baseline.

1.2 Thesis Outline

This thesis addresses several research questions in graph learning, including multi-view graph generation, non-contrastive link prediction, connecting clustering and contrastive learning, and the handling of OOV values in recommendation systems. We improve these by focusing on three frontiers: bridging ideas from other fields, focusing on time and space efficiency, and challenging existing ideas.

Chapter 2 provides the necessary background and notation for this thesis. Chapter 3 discusses improving multi-view graph generation. Chapter 4 covers evaluating and improving non-contrastive link prediction. Chapter 5 connects contrastive learning on graphs with clustering. Chapter 6 describes how we can improve the handling of OOV values in recommendation systems. Finally, Chapter 7 summarizes this thesis and its contributions.

Chapter 2

Background

In this chapter, we provide the notation and definitions for the material used in this thesis.

2.1 Introduction

In Section 2.2, we provide the necessary definitions and notations. In Section 2.3.1, we introduce Graph Neural Networks (GNNs) which are used throughout this thesis. We defer chapter-specific background information to their respective chapters.

2.2 Notation

We denote a graph as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of n nodes (i.e., $n = |\mathcal{V}|$) and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ be the set of edges. Let the node-wise feature matrix be denoted by $\boldsymbol{X} \in \mathbb{R}^{n \times f}$, where f is the number of raw features, and its i-th row $\boldsymbol{x}_i$ is the feature vector for the i-th node. Let $\boldsymbol{A} \in \{0, 1\}^{n \times n}$ denote the binary adjacency matrix. We denote the graph's learned node representations as $\boldsymbol{H} \in \mathbb{R}^{n \times d}$, where d is the size of the latent dimension, and $\boldsymbol{h}_i$ is the

representation for the i-th node. Let $\boldsymbol{Y} \in \{0,1\}^{n \times n}$ be the desired output for link prediction, as $\mathcal{E}$ and $\boldsymbol{A}$ may have validation and test edges masked off. Similarly, let $\hat{\boldsymbol{Y}} \in \{0,1\}^{n \times n}$ be the output predicted by the decoder for link prediction. Let ORC be a perfect oracle function for a link prediction task, i.e., $\text{ORC}(\boldsymbol{A}, \boldsymbol{X}) = \boldsymbol{Y}$. Let $\mathcal{N}(u) = \{v \mid (u,v) \in \mathcal{E} \vee (v,u) \in \mathcal{E}\}$. Let $\mathcal{N}(u)$ be a function that returns the set of neighbors for a given node u (i.e., $\mathcal{N}(u) = \{v | (u,v) \in \mathcal{E}\}$). Note that we use the terms "embedding" and "representation" interchangeably in this thesis.

2.3 Definitions

Below, we define a few terms used throughout our work which helps set the context for our thesis.

2.3.1 Graph Neural Networks (GNNs)

A message-passing Graph Neural Network consists of several iterations, which for a node u, can be described as follows:

$$\boldsymbol{h}_u^{(k+1)} = \text{UPDATE}^{(k)}\left(\boldsymbol{h}_u^{(k)}, \text{AGGREGATE}^{(k)}(\{\boldsymbol{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\})\right) \tag{2.1}$$

where UPDATE and AGGREGATE are differentiable functions, and $\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u$.

2.3.2 Link Prediction with GNNs

Many new approaches have also been developed with the recent advent of graph neural networks (GNNs). A predominant paradigm is using node-embedding-based methods [75, 14, 211, 230]. Node-embedding-based methods typically consist of an encoder $\boldsymbol{H} = \text{ENC}(\boldsymbol{A}, \boldsymbol{X})$ and a decoder $\text{DEC}(\boldsymbol{H})$. The encoder model is typically a message-passing-based

Graph Neural Network (GNN) [107, 75, 226]. The decoder model is usually an inner product or MLP applied on a concatenation of Hadamard product of the source and target learned node representations [155, 198]. This thesis, for the purposes of link prediction, focuses exclusively on these node-embedding-based methods.

2.3.3 Multiplex Graphs

A multiplex graph consists of several views, where each view is a graph. Each of the views contains the same nodes, but with different edges. Formally, a multiplex graph G with k views can be written as $G = (V, (E_1, E_2, \dots E_k))$, where V is the shared vertex set and E_i are the edges at the i-th layer. It is worth noting that not all of the nodes need to be connected within each view.

One example of a multiplex graph is a social network, where each edge in a given view represents a different mode of communication. A time-evolving graph with a constant number of nodes could also be represented as a multiplex graph, with each view representing a different timestamp. Finally, a knowledge base could be seen as a multiplex graph, where each node represents an entity and each view represents a different relation. It is worth noting that this differs from the traditional notion of a *multigraph*, where nodes can have multiple edges between them, but each edge is of the same type. Edges in a multiplex graph can have multiple types.

Definition 2.3.1 (Augmentation) *An augmentation* $\textsc{Aug}^+$ *is a label-preserving random transformation function* $\textsc{Aug}^+ : (\boldsymbol{A}, \boldsymbol{X}) \rightarrow (\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{X}})$ *that does not change the oracle's expected value:* $\mathbb{E}[\textsc{Orc}(\textsc{Aug}^+(\boldsymbol{A}, \boldsymbol{X}))] = \boldsymbol{Y}$.

Definition 2.3.2 (Corruption) *A corruption* $\textsc{Aug}^-$ *is a label-altering random transformation* $\textsc{Aug}^- : (\boldsymbol{A}, \boldsymbol{X}) \to (\check{\boldsymbol{A}}, \check{\boldsymbol{X}})$ *that changes the oracle's expected value:* $\mathbb{E}[\textsc{Orc}(\textsc{Aug}^-(\boldsymbol{A}, \boldsymbol{X}))] \neq \boldsymbol{Y}$.[1]

Definition 2.3.3 (Contrastive Learning) *Contrastive methods select anchor samples (e.g. nodes) and then compare those samples to both positive samples (e.g. neighbors) and negative samples (e.g. non-neighbors) relative to those anchor samples.*

Definition 2.3.4 (Non-Contrastive Learning) *Non-contrastive methods select anchor samples, but only compare those samples to variants of themselves, without leveraging other samples in the dataset.*

Definition 2.3.5 (OOV Values) *We consider a value Out-Of-Vocabulary (OOV) if it is a categorical value that does not exist at training time but appears at inference time.*

Definition 2.3.6 (IV Values) *We consider a value in-vocabulary (IV) if it is a categorical value that exists at both training and evaluation time — i.e., it is not an OOV value.*

Definition 2.3.7 (Tranductive Setting) *We define the transductive setting as an evaluation setting where no new categorical values (OOV values) are present during inference.*

Definition 2.3.8 (Inductive Setting) *We define the inductive setting as an evaluation setting where some OOV values appear at inference time. We consider it fully inductive if all of the evaluation samples contain OOV values, and partially inductive if the evaluation samples consist of both IV and OOV values.*

[1] Note that the definition of these functions are different from the corruption functions in Zhu et al. [236] (which we define as *augmentations*) and are instead similar to the corruption functions in Veličković et al. [186].

Chapter 3

Generating Multiplex Tensor Graphs

In this work, we explore multiplex graph (networks with different types of edges) generation with deep generative models. We discuss some of the challenges associated with multiplex graph generation that make it a more difficult problem than traditional graph generation. We propose TenGAN, the first neural network for multiplex graph generation, which greatly reduces the number of parameters required for multiplex graph generation. We also propose 3 different criteria for evaluating the quality of generated graphs: a graph-attribute-based, a classifier-based, and a tensor-based method. We evaluate its performance on 4 datasets and show that it generally performs better than other existing statistical multiplex graph generative models. We also adapt HGEN, an existing deep generative model for heterogeneous information networks, to work for multiplex graphs and show that our method generally performs better.

Graphs are used to represent many different types of data—from protein interactions [144] to social networks [135]. There are a similarly large number of useful graph-related

tasks, like link prediction and node classification. One of these tasks is graph generation, which is the focus of this work. Graph generation models can generally be split into two types: statistical attribute-based generative models and deep generative models. Statistical generative models like the Erdős-Rényi (ER) [48], Barabási-Albert (BA) [11], and stochastic block models [84] have explicitly defined parameters like the attachment rate or the number of communities.

In contrast, many deep generative models can learn directly from one or more input graphs. This ability allows them to mimic attributes of the input dataset without defining them explicitly. The majority of these models are based on either RNNs (Recurrent Neural Networks), GANs (Generative Adversarial Networks) [63], or VAEs (Variational Autoencoders) [104]. Examples of these include GraphRNN [214], NetGAN [18], GraphVAE [105], and LGGAN [51].

However, sometimes a simple graph structure may not be sufficient to represent a dataset accurately. One example of this is the Enron dataset [152], where each node is a person and each edge is an email between them. Representing this as a standard graph would only show that two people communicated with each other, without any information on when they communicated. For example, the two people may have exchanged an email once, daily, or once a month–each of which would indicate a very different relationship. Representing this data as a multiplex graph allows us to fully represent this information.

Another use-case for a multiplex graph is to capture different types of social media interactions across the same users. Each node would represent a person and each edge type could represent a different form of interaction. An example of this could be a graph

representing Twitter interactions, where each edge could either represent a retweet, mention, or following. Using only one of these would unecessarily limit the information captured in the graph.

Traditional graph generation models and multiplex graph generation models are useful in many of the same ways. For example, they can be used to anonymize private data [197] in order to enhance the reproducibility of models trained on private datasets. This would allow the user to preserve interesting structures in the graph without leaking private user information.

However, all of the current multiplex network generation models are statistical generative models. BINBALL [13] adapts ideas from BA and ER models and proposes new multiplex preferential attachment rules. StarGen [57] further builds upon BINBALL by separating the parameters controlling the global and local degree of nodes, increasing the diversity of individual layers. ANGEL [58] uses a hub-and-spoke-based model to generate multiplex graphs, allowing it to better mimic certain structures. These existing works pose some major limitations, namely:

1. *Explicit parameterization.* The existing models declare a set of parameters that affect the output of the graph. These parameters are inflexible and may lead to overfitting to a particular graph attribute while neglecting another.

2. *Limited datasets.* All three of the methods focus on Airline Transportation Networks (ATNs), specifically on the EU airline dataset [25]. It is difficult to determine if the methods will work for different datasets. In this work, we explore the task of multiplex graph generation on datasets from wildly varying domains.

3. *Limited evaluation criteria.* It is difficult to quickly compare the performance of the models on different datasets. The performance evaluation of the models is primarily done visually by comparing the distributions of various topological properties. This also ignores some other potentially interesting characteristics of the generated graphs like whether or not a classifier can distinguish between real and generated samples, especially useful for tasks like graph anonymization.

The first problems can be resolved by using a neural network to learn *directly* from a set of input graphs. However, extending a traditional graph generation network is not straightforward since the layers are often correlated. There are also many more parameters required. Furthermore, it is difficult to evaluate the quality of the generated graphs. We further investigate these issues in Section 3.1 below.

In this work, we tackle these issues and propose TenGAN, a tensor-based GAN, to generate multi-view graphs. With minor modifications, our approach readily generalizes to other data sources that can be modelled well with tensor decompositions. For example, tensor decompositions have been shown to work well on a variety of data, including fMRI [137] and EEG data [36], NBA game data [140], network traffic data [12], and spatio-temporal urban computing data [195]. However, in this work, we focus on the domain of multiplex graphs and reserve exploring other domains for future work.

Our contributions include:

- **Novel method**: We propose a novel GAN-based method to generate multiplex graphs that uses tensor decomposition to reduce the number of parameters required.

- **Evaluation criteria**: We propose 3 different evaluation metrics for multiplex graph generation and evaluate their effectiveness.
- **Thorough experimentation**: We conduct thorough experiments on 4 different datasets across 2 different models to evaluate the performance of our method. We also modify an existing method for heterogeneous graphs to work with multiplex graphs and compare our method against it.

3.1 Problem Formulation

We consider the following problem:

> **Given** a set of multiplex graphs $\mathcal{G}$ drawn from some unknown distribution $p_{real}(\mathcal{G})$, **learn** a distribution $p_{gen}(\mathcal{G})$ such that, on average, a graph $\hat{G} \sim p_{gen}(\mathcal{G})$ is indistinguishable from a graph $G \sim p_{real}(\mathcal{G})$ with respect to a set of criteria $\mathcal{C}$ while maintaining diversity (i.e., $G_1 \neq G_2$ w.h.p. for $G_1, G_2 \sim p_{gen}$).

Some examples of criteria in $\mathcal{C}$ may include graph attributes (like clustering coefficient and degree distribution) or the correlation between different slices. However, it is not fully clear what these criteria should be and is one of several challenges in multiplex graph generation, some of which are listed below:

Challenge 1: Inadequate Evaluation Criteria. Before we can decide on an appropriate model, we must determine our evaluation criteria. This is challenging because we need to consider not only the graph attributes of each view but also the relationships between each view. This

means that many of the graph evaluation metrics commonly used in graph generation are insufficient for multiplex graph generation. In this work, we propose 3 methods of evaluation for multiplex graph generation models, described in Section 3.2.4 below.

Challenge 2: Sampling from Multiplex Graphs. Many existing multiplex datasets consist of a single graph with multiple views. However, since we are emulating a set of multiplex graphs, we need many smaller multiplex graphs to form a distribution. In the traditional graph setting, there are existing datasets that consist of multiple graphs (e.g., the PPA dataset[87]). However, to the best of our knowledge, there are no such existing datasets for multiplex graphs. Therefore, we need a sampling method that samples smaller multiplex sub-graphs from a larger multiplex graph. We describe our method in Section 3.2.1 below.

Challenge 3: Large Number of Parameters. If we naively attempt to generate a multiplex graph, the number of parameters required will explode. This is because we need $\mathcal{O}(k \times n^2)$ parameters to generate an adjacency tensor for a multiplex graph with k views and n nodes. We propose TenGAN (described in Section 3.2.2) that generates a compressed tensor-decomposition-based representation to solve this problem.

3.2 Proposed Method

We propose TenGAN, a GAN-based model that first generates factors of a tensor decomposition model, then uses those to generate the adjacency tensor. We propose two variants: TenGAN-CP, which uses the CPD and TenGAN-R, which uses the RESCAL

decomposition. We first sample sub-multiplex graphs from the dataset, as described in Section 3.2.1. Then, we train our GAN on the sampled multiplex graphs. Finally, we sample random graphs from the generator (by passing in different random noise vectors) and evaluate them with the metrics described in Section 3.2.4.

3.2.1 Sampling

Many generative models require multiple input samples, rather than a single example. For example, LGGAN [51] is trained on 2-hop and 3-hop egonets extracted from the original source graph. However, it has been shown that this can lead to biased samples that may not necessarily be representative of the original graph, especially in terms of in-degree and community structure [115]. To help avoid this, we perform random-walk sampling across each view. We then use the induced subgraph on the remainder of the views.

However, one issue with many large multiplex datasets is that most of the nodes may be disconnected in any given view. In extreme examples, like in some knowledge graphs, almost all nodes will be disconnected in each view. Oftentimes, even the union of all edges across all views will still result in a disconnected graph.

Another issue is that these datasets are often too large or have too many views. For example, the NELL dataset [26, 176] has over 2 million views. Random sampling of the dataset would produce extremely or completely sparse entries. In order to produce better quality samples, we use the sampling method of ParCube [49].

This computes the following importance score for each slice along each mode. For example, the importance scores for each slice along the three modes for a tensor $\boldsymbol{\mathcal{T}}$ would, respectively, be:

$$\boldsymbol{a}_i = \sum_{j=1}^{J}\sum_{k=1}^{K} \boldsymbol{\mathcal{T}}_{i,j,k}; \quad \boldsymbol{b}_j = \sum_{i=1}^{I}\sum_{k=1}^{K} \boldsymbol{\mathcal{T}}_{i,j,k}; \quad \boldsymbol{c}_k = \sum_{i=1}^{I}\sum_{j=1}^{J} \boldsymbol{\mathcal{T}}_{i,j,k}$$

We then randomly sample indices for each mode, with the probability of a given index being selected proportional to its score. For example, a given index i is selected with probability $\boldsymbol{a}_i / \sum_{x=1}^{I} \boldsymbol{a}_x$. This results in a more dense tensor and therefore a multiplex graph with more connected nodes, reducing the chance of getting empty (or near-empty) tensors as inputs to our model.

3.2.2 Architecture

Our model is a GAN and consists of a generator network and a discriminator network. The generator consists of a MLP, followed by two or three smaller MLPs to generate the factor matrices/tensors (depending on the factorization method). We further describe the two generator architectures below in Section 3.2.2. The discriminator uses the max pool of several Graph Convolutional Networks (GCNs) [106] (one per view) followed by a fully-connected layer to predict if a sample is generated or drawn from the original real dataset. A diagram of our architecture is shown in Figure 3.1.

TenGAN-CP Architecture

TenGAN-CP uses a shared feature extractor layer and splits into separate networks, each of which generates a different factor in the CPD. This can be considered a higher-order

extension of the BRGAN-B [171] architecture and uses the tensor CPD instead of the matrix SVD.

After generating the factors, we calculate the sum of the outer products of vectors from our factor matrices $\mathbf{A}, \mathbf{B}$, and $\mathbf{C}$: $\sum_{i=1}^{r} \boldsymbol{a}_i \circ \boldsymbol{b}_i \circ \boldsymbol{c}_i$. As shown in Section 3.2.3, this reduces the number of parameters needed to generate a given multiplex graph. We use the loss function from Wei et al. [199] (which is based on Arjovsky et al. [7]):

$$L = \mathop{\mathbb{E}}_{[\![\mathbf{A},\mathbf{B},\mathbf{C}]\!]\sim\mathbb{P}_g} [D([\![\mathbf{A},\mathbf{B},\mathbf{C}]\!])] - \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathbb{P}_r} [D(\boldsymbol{x})] + \lambda_1 \mathrm{GP} + \lambda_2 \mathrm{CT}$$

where GP is the gradient penalty term from Gulrajani et al. [69], CT is the consistency term from Wei et al. [199], and $\mathbf{A}, \mathbf{B}, \mathbf{C}$ is the output from the generator. $\mathbb{P}_g$ denotes the distribution of graphs generated by the generator, and $\mathbb{P}_r$ denotes the real distribution of graphs. $D(\cdot)$ denotes the output of the discriminator on a given tensor. The goal of this loss function is to minimize the difference between the expected values of the discriminator's output on the generated data and the input (real) data.

TenGAN-R Architecture

We also propose the TenGAN-R architecture, which is initially similar to the TenGAN-CP architecture, with the distinction that we use the RESCAL decomposition instead of the CPD. This results in more parameters for the same value of r, but performs better on certain datasets.

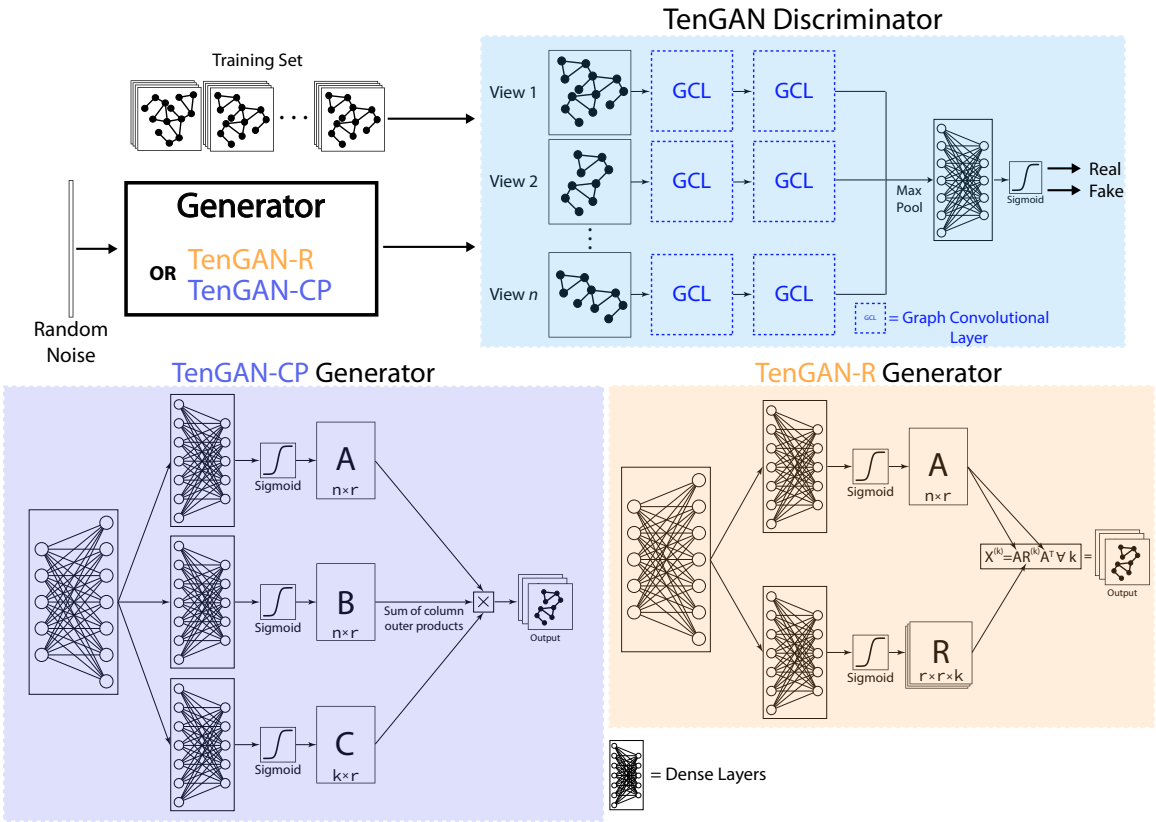


Figure 3.1 Diagram of the TenGAN discriminator (top) and two different generator architectures (bottom). TenGAN-CP is based on the CP decomposition, and generates the factor matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ before combining them into the output adjacency tensor. TenGAN-R is based on the RESCAL decomposition, and first generates the factor matrix $\mathbf{A}$ and factor tensor $\boldsymbol{\mathcal{R}}$ before combining them into the adjacency tensor.

3.2.3 Parameter Complexity

If we attempted to generate an adjacency tensor for a multiplex graph with n nodes and k views directly, we would have to use $\mathcal{O}(kn^2)$ parameters in the final layer. However, if we generate the CPD factors first, we only need $\mathcal{O}(r(n+k))$ parameters in the final layer, where r is a hyperparameter that increases the quality of the fit at the cost of more parameters. This offers savings for $r < n^2$, and we show that our models work well for this case in Table 3.2. For the RESCAL-based formulation, we need $\mathcal{O}(nr + kr^2)$ parameters in the final layer. In this case, we only reduce the number of parameters in the case where $r < n$.

3.2.4 Evaluation Metrics

Another difficult task in multiplex graph generation is evaluating the quality of the generated graphs. Gretton et al. [65] found that measuring the Maximum Mean Discrepancy (MMD) between distributions of different graph statistics works well for simple graphs. We propose 3 methods for evaluating the structural similarity between generated and input graphs:

MMD-Based Evaluation

One method to evaluate the quality of generated multiplex graphs would be to apply the evaluation criteria used for simple graphs to each view. We measure the Mean MMD (M-MMD) score between the distributions of different graph attributes. More concretely, for each graph attribute, we take the mean of the MMD between the i-th view of a generated

graph G' and a graph G. We can use the clustering coefficient, degree distribution, and the orbit of the graphs similar to [214].

The main downside of this approach is that it does not take the relationship between the views into account. For example, consider the case where we generate multiplex graphs with two views. Let the list of the generated first and second views be $V_1' = g'^{(1)} \forall g' \in G'$ and $V_2' = g'^{(2)} \forall g' \in G'$. Then, suppose the MMD scores of V_1' and V_2' across all the graph attributes are 0. Then, the overall Mean MMD (M-MMD) would be 0. However, swapping V_1' and V_2', yields same M-MMD.

This is clearly an undesirable behavior in any case where each of the views are correlated with each other. An extreme example of this would be a multiplex graph g where $g^{(1)}$ has an edge iff $g^{(2)}$ does not have an edge. Then, it is possible for a generated graph g' to have $g'^{(1)} = g'^{(2)}$, but still have a perfect M-MMD of 0 in all the graph attributes. To address this issue, we propose the tensor-based evaluation method below.

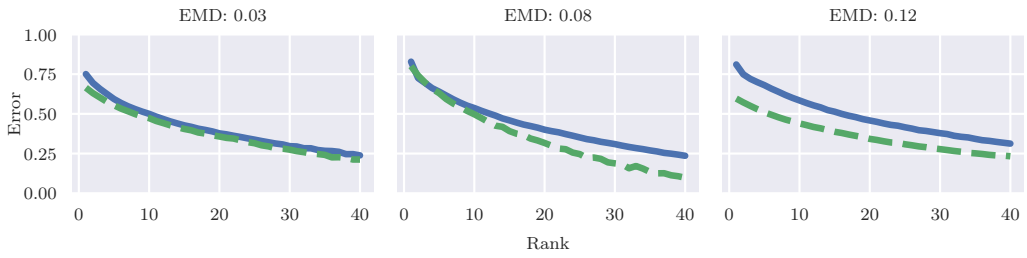


Figure 3.2 3 plots of pairs of CPD error values that have low, medium, and high EMD scores. CPD error vs. rank plot on 3 pairs of tensors. The dashed green lines are the generated results, and the solid blue lines are the original results. We can see that the EMD scores are lower when the lines are more similar and that the scores are higher when the lines are further apart. This is the intuition behind our tensor-based evaluation method.

Tensor-Based Evaluation

Multiplex graphs can be viewed as third-order tensors, where each slice is a graph across the same nodes, and tensor decompositions have been shown to be able to extract structure (like communities) from multiplex graphs [68, 59, 3, 169]. We take advantage of this fact by applying the CPD to each multiplex graph or tensor. The normalized reconstruction error of the decomposition for various values of r provides a heuristic for how much structure there is along the three modes of the tensor. We then compare the errors of the generated and original tensors across different ranks to see if they are similar in terms of trilinear structure. It may be possible to produce a similar reconstruction error for a given rank without matching the structure of the real graph, but we argue that it is highly unlikely for this to occur across many different values of r.

We randomly sample n tensors from the generated and real tensors and compute an error vector $\mathbf{e}$ of the errors across different ranks. We then calculate the sum of the Wasserstein metric (a.k.a. the earth mover's distance: EMD) between all n^2 pairs of error vectors. The lower this score, the more similar pairs are (on average). While this score works well across a fixed dataset, it is difficult to compare this score across datasets of different sizes. This is because the number of feasible r values changes with the size of the tensor; and a given dataset may naturally have a wider range of pairwise distances.

To solve this issue, we normalize the sum of generated-real distances by the sum of pairwise real-real distances. More formally, given real error matrix $\mathbf{E}$ and generated error matrix $\mathbf{E}'$ (where every row $\mathbf{E}_i$ is a vector of the i-th sample's CPD errors):

$$\textsc{TenScore} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\mathsf{EMD}(\mathbf{E}_i, \mathbf{E'}_j)}{\sum_{i=1}^{n}\sum_{j=1}^{n}\mathsf{EMD}(\mathbf{E}_i, \mathbf{E}_j)} \tag{3.1}$$

The lower the TenScore, the more realistic the generated samples are. TenScore also serves as an indicator for graph diversity. If it near 0, it likely means that the model is suffering from mode collapse—a common problem among GANs. We also propose a modified version of TenScore for knowledge base graphs: TenScore-R, which uses RESCAL's error.

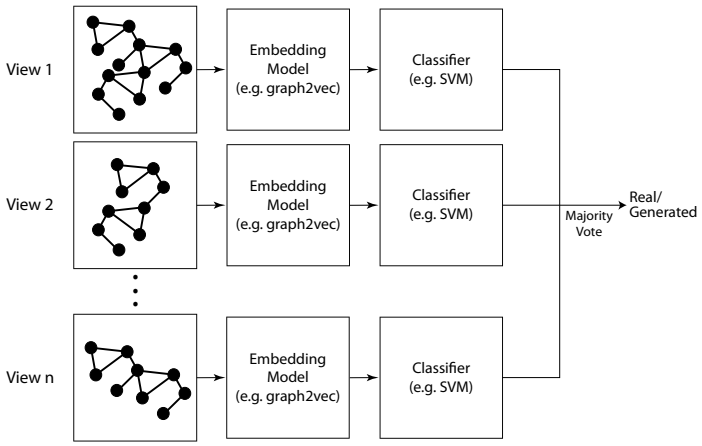


Figure 3.3 Diagram of the classifier-based evaluation model. We first calculate an embedding and train a classifier on each view before using the majority vote to guess if the result is a real or generated multiplex graph. Note that this is similar to, but different from the discriminator. This is because we use an established embedding model and a non-neural-network classifier.

Classifier-Based Evaluation

We train a classifier on generated and original data; then check to see if it correctly predicts the origin of an example. We calculate the accuracy and F1 score of the resulting

model (the closer to 0.5 or 50%, the better). In the model, we calculate a graph2vec [134] embedding for each view of the multiplex graph. Then, we split the embeddings into training/test data and train a SVM classifier for each view. Finally, we take the majority vote of the ensemble. These steps are shown in a diagram in Figure 3.3.

3.2.5 Implementation Details

We implemented this model in PyTorch [143] on Python 3.9. We used NetworKit [177] and NetworkX [74] for graph data, and Tensorly [110] for tensor decompositions. We extended portions of the GraphRNN [214] evaluation code and heavily modified HGEN [122] to work for multiplex graphs (see details in Section 3.3.2). We use the code from [157] for the BINBALL, StarGen, and ANGEL baselines. It was originally written for undirected networks, so we extend it to work for directed multiplex graphs. The code for our experiments is available here[1].

3.3 Experimental Evaluation

3.3.1 Datasets

We used 4 multiplex graph datasets selected to represent a wide variety of data types, including a social media network, a knowledge graph, a computer network communication graph, and a time-evolving network.

1. **Football** [64]: 248 English Premier League football players and clubs on Twitter, where each of the 6 views corresponds to a different interaction between the accounts

[1] `https://github.com/willshiao/tengan`

(follows, followed-by, mentions, mentioned-by, retweets, retweeted-by). Note that 3 of the views are essentially transposes of the other 3.

2. **NELL-2** [26]: A sampled version of the NELL-2 dataset (from [176]) that consists of (*entity*, *relation*, *entity*) tuples. The original size is $12,092 \times 9,184 \times 28,818$, but we resample it to a $1,000 \times 4 \times 1,000$ tensor (where 4 is the number of views) for the purpose of evaluation. The sampling method is described in Section 3.2.1.

3. **Comm**: An enterprise communication network dataset of 1,558,594 computers. Each view corresponds to communications between nodes on one of five ports (22, 23, 80, 443, and 445), with one view for each port. In the view associated with port p, a directed edge from u to v exists if u initiates a connection to v over port p. Since we are unable to provide a copy of this network, descriptive statistics of each view in this network are shown below in Table 3.1.

4. **Enron** [152]: A multiplex graph of emails sent between Enron employees, where each view represents a two-month (60-day) time interval and edges represent emails. The original tensor (available at [176]) is 6,066 senders $\times$ 5,699 recipients $\times$ 244,268 words $\times$ 1,176 days. We collapse the words dimension and simply add an unweighted edge for each email sent in a given time interval. We also aggregate the slices so that each view represents a 60-day period to reduce the number of views. Finally, we sample 1,000 senders and 1,000 recipients using the methodology described in the supplementary material. Finally, we sample 1,000 senders and 1,000 recipients using the methodology described in Section 3.2.1.

We perform random walk sampling to extract a set of sub-multiplex-graphs from each dataset. We describe this process with more detail in Section 3.2.1.

Port	**# Nodes**	**# Edges**	**LCC Size**	**SP**	**CC**
22	295,077	1,039,721	205,043	3.9	0.10
23	9,304	17,408	5,578	3.1	0
80	640,492	3,495,394	374,767	4.4	0.04
443	1,172,959	6,705,799	439,911	4.7	0.02
445	437,119	7,927,648	219,089	3.8	0.06

Table 3.1 Since we are unable to share the data for the Comm dataset, we instead provide network statistics of the computer infrastructure graph. Data are over the course of one day on five TCP/IP ports: 22 (SSH), 23 (Telnet), 80 (HTTP), 443 (HTTPS), and 445 (MS Directory Services). For each view (port), we list the number of active nodes, the number of edges, the number of nodes in the largest strongly connected component (LCC Size), and the average shortest path length (SP) and average clustering coefficient (CC). SP and CC are computed based on 1,000 randomly sampled nodes (CC) or node pairs (SP) within the induced subgraph of the largest strongly connected component.

3.3.2 Comparison with Existing Methods

To the best of our knowledge, no other deep learning models for multiplex graph generation exist. The existing models are statistical and are built to match specific attributes of the underlying graph. We compare our method against BINBALL [13], StarGen [57], and ANGEL [58]. We also adapt HGEN [122]—a generative model for heterogeneous graphs—to work for multiplex graphs. A heterogeneous graph consists of nodes of different types and, therefore, edges of different types. An common example of this is a citation graph, where we might have nodes for authors, papers, and conferences. This is in contrast to multiplex graphs,

where we have different views of the same nodes. As such, it is difficult to directly compare the two methods—however, we attempt to convert multiplex graphs to heterogeneous graphs and evaluate its performance. The HGEN code[2] (as provided in the paper) does not support different edge types or a single node belonging to multiple classes, so we encountered the following issues (some of which may affect its performance).

Lack of multiplex graph support. Let n be the number of nodes and k be the number of views in our original multiplex graph. To work around the lack of support for different edge types, we create k nodes for each of the n nodes in the original graph, each with a different class in the range $[1, k]$. Then, we link together each of these k nodes in the heterogeneous network. This results in a total of nk nodes across k classes in the resulting heterogeneous network.

Graph size issues. While the actual HGEN model is efficient for generation, it requires HIN node embeddings for each node in the input graph. We chose to use hin2vec [56]— same as in the original HGEN code. However, we have nk nodes after the conversion to a HIN, causing the embeddings to take too long to calculate on some datasets (like the Comm dataset).

3.3.3 MMD-Based Evaluation

The MMD-based evaluations compares the similarity of different graph attributes for each slice between the real and generated graphs. From Table 3.2, we can see that TenGAN-CP and TenGAN-R generally perform fairly well on the Football, NELL-2 and Comm datasets. However, this is a layer-level comparison, and even BA (which treats each

[2] `https://github.com/lingchen0331/HGEN`

	Football						NELL-2					
Model Name	Deg	Clust	Orbit	F1	Acc	TenScore	Deg	Clust	Orbit	F1	Acc	TenScore
TenGAN-CP	**0.10**	0.45	**0.10**	0.57	0.70	0.92	0.48	0.70	0.31	0.94	0.94	**0.89**
TenGAN-R	1.09	1.12	0.75	0.94	0.94	**0.77**	0.93	0.77	0.91	**0.49**	**0.66**	2.64
HGEN	1.03	1.45	0.84	1.00	1.00	5.00	0.98	0.74	0.65	1.00	1.00	5.67
Barabási-Albert	1.07	1.50	0.60	1.00	1.00	2.93	0.81	0.31	0.30	1.00	1.00	13.16
BINBALL	1.10	1.30	0.99	0.64	0.66	11.64	**0.47**	**0.27**	**0.24**	0.92	0.93	3.33
StarGen	1.06	1.25	0.99	0.63	0.70	11.20	0.77	0.40	0.25	0.91	0.91	2.55
ANGEL	0.15	**0.37**	0.18	**0.48**	**0.52**	4.59	0.78	0.25	0.51	0.94	0.94	6.78
	Enron						**Comm**					
Model Name	Deg	Clust	Orbit	F1	Acc	TenScore	Deg	Clust	Orbit	F1	Acc	TenScore
TenGAN-CP	0.65	0.19	0.53	**0.87**	**0.89**	**0.86**	0.37	0.39	0.38	0.72	0.61	1.50
TenGAN-R	1.77	1.97	1.74	0.99	0.99	4.32	**0.34**	0.40	0.35	**0.69**	**0.56**	1.31
HGEN	0.59	0.02	**0.06**	1.00	1.00	0.90	-	-	-	-	-	-
Barabási-Albert	0.67	0.03	0.23	1.00	1.00	1.57	1.24	**0.001**	0.37	1.00	1.00	3.46
BINBALL	**0.32**	0.02	0.07	0.92	0.92	3.41	0.78	0.46	**0.06**	1.00	1.00	**0.89**
StarGen	0.64	0.16	0.07	0.99	0.99	4.61	0.90	0.58	0.07	1.00	1.00	1.06
ANGEL	0.58	**0.01**	0.19	0.98	0.98	1.34	0.99	0.08	0.20	1.00	1.00	1.14

Table 3.2 Results of TenGAN on the Football, NELL-2, Enron, and Comm datasets. **Lower is better for all of these metrics, including F1/Acc.** Deg, Clust, and Orbit are the mean MMD scores of our MMD-based evaluation method (see Section 3.3.3). F1 and Acc are the scores produced by the classifier-based method (see Section 3.2.4). Note that a higher F1/Acc score means that the classifier is better able to distinguish between positive/negative samples, implying that the generated samples are less realistic. TenScore is the score produced by the CPD-based tensor evaluation method (see Section 3.3.4). HGEN is unable to run on the Comm dataset due to issues mentioned in Section 3.3.2.

layer separately) performs decently well in this comparison. This is why the other evaluation methods are important, especially the tensor-based evaluation, which provides a holistic look at the generated tensors.

3.3.4 TenScore Evaluation

TenGAN-CP tends to perform best in terms of TenScore across all the datasets, with the exception of the Comm dataset (where BINBALL performs slightly better). TenGAN-CP has a TenScore of below 1 on the Football, NELL-2, and Enron datasets. This indicates that the mean EMD for all real-generated pairs is lower than the mean EMD for all real-real pairs in the dataset. None of the methods have a very low TenScore, which means that all of the methods exhibit a good amount of diversity comparable to that of the original data. However, some of the baseline methods have a very high TenScore, indicating that the generated graphs have a very different amount of trilinear structure from that of the input graphs.

Suprisingly, Barabási-Albert outperforms some of the other statistical methods on the Football and Enron datasets like BINBALL and StarGen. This is likely because BINBALL and StarGen focus on airport transportation networks and therefore focus on modelling behavior like hub-spoke formations [13, 57]. These structures are more present in the sparser NELL-2 and Comm datasets than the denser Football and Enron datasets.

3.3.5 Classifier-Based Evaluation

TenGAN-CP performs fairly well on the Football dataset, with an accuracy of 0.70 and F1 score of 0.57 (recall that the lower the accuracy, the better). TenGAN-R performs

well on the NELL-2 and Comm datasets. No model does a very good job of fooling the classifier on Enron, likely due to the higher number of views.

Most of the baselines do a poor job of fooling the classifier, with many baselines resulting in the classifier having 100% accuracy. One reason is because some generators are able to model some layers extremely well, but sometimes fail to model other layers. This leads to the classifier for certain layers to have very high accuracy, making the overall classifier very accurate. For example, BINBALL randomly assigns (based on a parameter p) a layer as a BA model or an ER model. This assignment can mean that the generated results for a given layer will be significantly different from those of the original graph, causing the classifier to be very accurate on that layer.

Another reason for the poor performance of the baselines is that the majority of them are statistical models and rely on general rules (e.g. preferential attachment for BA). While this may be able to mimic some attributes, the node and graph embeddings will likely greatly differ (except in the case where the original graphs exhibit simple structure).

3.3.6 Summary

TenGAN performs well on the majority of the datasets across all of the evaluation criteria. It performs especially well in the classifier-based and TenScore evaluations. This is likely because TenGAN learns directly from the data in contrast to most of the other baselines, which have to learn explicitly defined parameters instead. However, TenGAN-R does significantly worse than TenGAN-CP on most datasets, despite requiring more parameters. This is likely because TenGAN-R tends to have a hard time converging on the Football and Enron datasets. A possible reason for this is that the RESCAL decomposition

imposes a stricter requirement on the factor matrix $\mathbf{A}$ since it is shared across all layers, making it difficult for the model to learn well. The hyperparameter r is also important in how well the model performs. Generally, r has to be higher for sparser tensors and lower for denser tensors. We do not carefully tune r in this paper—we select a resonable default (e.g., $r = 100$) and increase/decrease it until the model converges.

3.4 Related Work

There have been several works on the topic of multiplex graph sampling. Interdonato et al. [91] found that methods that work on standard graphs like Metropolis-Hastings random walks, BFS, and forest fire sampling [115] can also be applied to multiplex graphs. To improve random walk sampling on multiplex graphs, Gjoka et al. [62] proposes union multigraph sampling — a method that uses the "union multigraph", which consists of all edges across all views in the multiplex graph. Union multigraph sampling then performs a random walk over this multigraph to sample it. While unbiased samples are useful, we sometimes want a biased sample to better sample nodes with special properties. Khadangi et al. [101] propose using learning automata to do so.

There has also been some previous work on multiplex graph generation. For example, Nicosia et al. [136] propose a model to grow a multiplex graph based on traditional preferential attachment models like the Barabási-Albert (BA) model [11]. BinBall [13] also builds upon the BA model and focuses on air transportation networks. StarGen [57] directly improves upon BinBall by using a per-layer edge count distribution and splitting the scaling factor of a new node into global and local factors. Kim and Goh [102] also uses single-layer preferential

attachment models and tunes the correlation between layers. ANGEL [58] specifically tries to emulate the hub-and-spoke structure found in many graphs.

In recent years, neural networks have also been applied to graph generation. GraphRNN [214] uses an RNN to model graphs as a sequence of nodes of edges. NetGAN [18] uses an LSTM to learn the distribution of biased random walks and reconstructs graphs from them. GraphVAE [105] uses a variational autoencoder to generate graphs. LGGAN [51] generates the adjacency matrix directly, along with its associated labels. BRGAN [171] generates rank-constrained graphs by first generating factor matrices, in a similar manner to TenGAN. There have also been several models for multi-scale graphs. The key difference between a multiplex and multi-scale graph is that a multiplex graph contains the same nodes with different edges in each view, while a multi-scale graph typically contains representations of the same underlying graph at different resolutions (different number of nodes) in each layer. Misc-GAN [235] generates a multi-scale graph before collapsing it into a standard graph, and DMGNN [116] predicts multi-scale graphs from previous ones.

To the best of our knowledge, there have been no other neural-network-based models for multiplex graph generation. HGEN [122] allows for the deep generation of heterogeneous networks by modeling random walks over the graph with a GAN. However, their approach largely focuses on the modeling of inter-layer edges with meta-paths. Heterogeneous networks refer to graphs with different node and edge types, while we focus on the case with shared nodes but different edges/views. We elaborate more on the precise definition of a multi-view graph in Section 2.3.3 above.

3.5 Conclusion

In this work, we discuss some of the issues associated with multiplex graph generation, as well as some solutions to those issues. One of these issues is the large number of parameters required to generate a multiplex graph using a neural network. We tackle this by proposing a novel GAN-based method that leverages the CPD and RESCAL decompositions to greatly reduce the number of parameters required.

Another issue with multiplex graph generation is a lack of evaluation criteria. We address this by proposing 3 different evaluation metrics that evaluate the realism of the graph along different aspects. We also modify HGEN, a model for heterogeneous networks, to work with multiplex graphs. We run our models on 4 different datasets, compare their results against HGEN and 3 other statistical multiplex generation models, and find that we perform better on the majority of them.

Chapter 4

Link Prediction with Non-Contrastive Learning

Graph neural networks (GNNs) are prominent in the graph machine learning domain, owing to their strong performance across various tasks. A recent focal area is the space of graph self-supervised learning (SSL), which aims to derive useful node representations without labeled data. Notably, many state-of-the-art graph SSL approaches are *contrastive* methods, which use a combination of positive and negative samples to learn node representations. Owing to challenges in negative sampling (slowness and model sensitivity), recent literature introduced *non-contrastive* methods, which instead only use positive samples. Though such methods have shown promising performance in node-level tasks, their suitability for link prediction tasks, which are concerned with predicting link existence between pairs of nodes, and have broad applicability to recommendation systems contexts, is yet unexplored. In this work, we extensively evaluate the performance of existing non-contrastive methods for link

prediction in both transductive and inductive settings. While most existing non-contrastive methods perform poorly overall, we find that, surprisingly, BGRL generally performs well in transductive settings. However, it performs poorly in the more realistic inductive settings where the model has to generalize to links to/from unseen nodes. We find that non-contrastive models tend to overfit to the training graph and use this analysis to propose TenGAN, a novel non-contrastive framework that incorporates cheap corruptions to improve the generalization ability of the model. This simple modification strongly improves inductive performance in **5/6** of our datasets, with up to a **120%** improvement in Hits@50—all with comparable speed to other non-contrastive baselines, and up to **14**× faster than the best-performing contrastive baseline. Our work imparts interesting findings about non-contrastive learning for link prediction and paves the way for future researchers to further expand upon this area.

4.1 Introduction

Graph neural networks (GNNs) are ubiquitously used modeling tools for relational graph data, with widespread applications in chemistry [29, 71, 72, 124], forecasting and traffic prediction [42, 181], recommendation systems [211, 81, 159, 182, 53], graph generation [213, 52, 172], and more. Given significant challenges in obtaining labeled data, one particularly exciting recent direction is the advent of graph self-supervised learning (SSL), which aims to learn representations useful for various downstream tasks without using explicit supervision besides available graph structure and node features [236, 94, 183, 15].

One prominent class of graph SSL approaches are contrastive methods [93]. These methods typically utilize contrastive losses such as InfoNCE [138] or margin-based losses [211]

between node and negative sample representations. However, such methods usually require either many negative samples [79] or carefully chosen ones [211, 207], where the first one results with quadratic number of in-batch comparisons, and the latter is especially expensive on graphs since we often store the sparse adjacency matrix instead of its dense complement [183, 15]. These drawbacks motivated the development of non-contrastive methods [183, 15, 222, 100], based on advances in the image domain [66, 33, 32], which do not require negative samples and solely rely on augmentations. This allows for a large speedup compared to their contrastive counterparts with strong performance [15, 222].

However, non-contrastive SSL methods are typically evaluated on node-level tasks, which is a more direct analog of image classification in the graph domain. In comparison, the link-level task (link prediction), which focuses on predicting link existence between pairs of nodes, is largely overlooked. This presents a critical gap in understanding: *Are non-contrastive methods suitable for link prediction tasks? When do they (not) work, and why?* This gap presents a huge opportunity, since link prediction is a cornerstone in the recommendation systems community [81, 225, 14].

Present Work. To this end, our work first performs an extensive evaluation of non-contrastive SSL methods in link prediction contexts to discover the impact of different augmentations, architectures, and non-contrastive losses. We evaluate all of the (to the best of our knowledge) currently existing non-contrastive methods: CCA-SSG [222], Graph Barlow Twins (GBT) [15], and Bootstrapped Graph Latents (BGRL) [183] (which has the same design as the independently proposed SelfGNN [100]). We also compare these methods against a baseline end-to-end GCN [107] with cross-entropy loss, and two contrastive baselines:

GRACE [236], and a GCN trained with max-margin loss [210]. We evaluate the methods in the transductive setting and find that BGRL [183] greatly outperforms not only the other non-contrastive methods, but also GRACE—a strong augmentation-based contrastive model for node classification. Surprisingly, BGRL even performs on-par with a margin-loss GCN (with the exception of 2/6 datasets). However, in the more realistic inductive setting, which considers prediction between new edges and nodes at inference time, we observe a huge gap in performance between BGRL and a margin-loss GCN (ML-GCN). Upon investigation, we find that BGRL is unable to sufficiently push apart the representations of negative links from positive links when new nodes are introduced, owing to a form of overfitting. To address this, we propose TenGAN, a novel non-contrastive method which uses a corruption function to generate cheap "negative" samples—without performing the expensive negative sampling step of contrastive methods. We show that it greatly reduces overfitting tendencies, and outperforms existing non-contrastive methods across 5/6 datasets on the inductive setting. We also show that it maintains comparable speed with BGRL, and is 14× faster than the margin-loss GCN on the `Coauthor-Physics` dataset.

Main Contributions. In short, our main contributions are as follows:

- To the best of our knowledge, this is the first work to explore link prediction with non-contrastive SSL methods.
- We show that, perhaps surprisingly, BGRL (an existing non-contrastive model) works well in the transductive link prediction, with performance at par with contrastive baselines, implicitly behaving similarly to other contrastive models in pushing apart positive and negative node pairs.

- We show that non-contrastive SSL models underperform their contrastive counterparts in the inductive setting, and notice that they generalize poorly due to a lack of negative examples.
- Equipped with this understanding, we propose TENGAN, a novel non-contrastive method that uses cheap "negative" samples to improve generalization. TENGANis simple to implement, very efficient when compared to contrastive methods, and improves on BGRL's inductive performance in 5/6 datasets, making it at or above par with the best contrastive baselines.

4.2 Preliminaries

Notation. We denote a graph as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of n nodes (i.e., $n = |\mathcal{V}|$) and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ be the set of edges. Let the node-wise feature matrix be denoted by $\boldsymbol{X} \in \mathbb{R}^{n \times f}$, where f is the number of raw features, and its i-th row $\boldsymbol{x}_i$ is the feature vector for the i-th node. Let $\boldsymbol{A} \in \{0,1\}^{n \times n}$ denote the binary adjacency matrix. We denote the graph's learned node representations as $\boldsymbol{H} \in \mathbb{R}^{n \times d}$, where d is the size of latent dimension, and $\boldsymbol{h}_i$ is the representation for the i-th node. Let $\boldsymbol{Y} \in \{0,1\}^{n \times n}$ be the desired output for link prediction, as $\mathcal{E}$ and $\boldsymbol{A}$ may have validation and test edges masked off. Similarly, let $\hat{\boldsymbol{Y}} \in \{0,1\}^{n \times n}$ be the output predicted by the decoder for link prediction. Let ORC be a perfect oracle function for our link prediction task, i.e., $\text{ORC}(\boldsymbol{A}, \boldsymbol{X}) = \boldsymbol{Y}$. Let $\mathcal{N}(u) = \{v \mid (u,v) \in \mathcal{E} \vee (v,u) \in \mathcal{E}\}$. Note that we use the terms "embedding" and "representation" interchangeably in this work.

GNNs for Link Prediction. Many new approaches have also been developed with the recent advent of graph neural networks (GNNs). A predominant paradigm is the use of node-embedding-based methods [75, 14, 211, 230]. Node-embedding-based methods typically consist of an encoder $\boldsymbol{H} = \textsc{Enc}(\boldsymbol{A}, \boldsymbol{X})$ and a decoder $\textsc{Dec}(\boldsymbol{H})$. The encoder model is typically a message-passing based Graph Neural Network (GNN) [107, 75, 226]. The message-passing iterations of a GNN for a node u can be described as follows:

$$\boldsymbol{h}_u^{(k+1)} = \textsc{Update}^{(k)}\left(\boldsymbol{h}_u^{(k)}, \textsc{Aggregate}^{(k)}(\{\boldsymbol{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\})\right) \tag{4.1}$$

where $\textsc{Update}$ and $\textsc{Aggregate}$ are differentiable functions, and $\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u$. The decoder model is usually an inner product or MLP applied on a concatenation of Hadamard product of the source and target learned node representations [155, 198].

Graph SSL. Below, we define a few terms used throughout our work which helps set the context for our discussion.

Definition 4.2.1 (Augmentation) *An augmentation* $\textsc{Aug}^+$ *is a label-preserving random transformation function* $\textsc{Aug}^+ : (\boldsymbol{A}, \boldsymbol{X}) \rightarrow (\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{X}})$ *that does not change the oracle's expected value:* $\mathbb{E}[\textsc{Orc}(\textsc{Aug}^+(\boldsymbol{A}, \boldsymbol{X}))] = \boldsymbol{Y}$.

Definition 4.2.2 (Corruption) *A corruption* $\textsc{Aug}^-$ *is a label-altering random transformation* $\textsc{Aug}^- : (\boldsymbol{A}, \boldsymbol{X}) \rightarrow (\check{\boldsymbol{A}}, \check{\boldsymbol{X}})$ *that changes the oracle's expected value:* $\mathbb{E}[\textsc{Orc}(\textsc{Aug}^-(\boldsymbol{A}, \boldsymbol{X}))] \neq \boldsymbol{Y}$.[1]

[1] Note that the definition of these functions are different from the corruption functions in Zhu et al. [236] (which we define as *augmentations*) and are instead similar to the corruption functions in Veličković et al. [186].

Definition 4.2.3 (Contrastive Learning) *Contrastive methods select anchor samples (e.g. nodes) and then compare those samples to both positive samples (e.g. neighbors) and negative samples (e.g. non-neighbors) relative to those anchor samples.*

Definition 4.2.4 (Non-Contrastive Learning) *Non-contrastive methods select anchor samples, but only compare those samples to variants of themselves, without leveraging other samples in the dataset.*

BGRL. While we examine the performance of all of the non-contrastive graph models, we focus our detailed analysis exclusively on BGRL[2] [183] due to its superior performance in link prediction when compared to GBT [15] and CCA-SSG [222]. BGRL consists of two encoders, one of which is referred to as the *online* encoder $\textsc{Enc}_\theta$; the other is referred to as the *target* encoder $\textsc{Enc}_\phi$. BGRL also incorporates a predictor $\textsc{Pred}$ (typically a MLP) and two sets of augmentations: $\mathcal{A}_1^+, \mathcal{A}_2^+$. A single training step for BGRL is as follows: (a) we apply these augmentations: $(\tilde{\boldsymbol{A}}^{(1)}, \tilde{\boldsymbol{X}}^{(1)}) = \textsc{Aug}_1^+(\boldsymbol{A}, \boldsymbol{X})$; $(\tilde{\boldsymbol{A}}^{(2)}, \tilde{\boldsymbol{X}}^{(2)}) = \textsc{Aug}_2^+(\boldsymbol{A}, \boldsymbol{X})$. (b) we perform forward propagation $\boldsymbol{H} = \textsc{Enc}(\tilde{\boldsymbol{A}}^{(1)}, \tilde{\boldsymbol{X}}^{(1)}); \boldsymbol{H}_2 = \textsc{Enc}(\tilde{\boldsymbol{A}}^{(2)}, \tilde{\boldsymbol{X}}^{(2)})$. (c) we pass the output through the predictor $\boldsymbol{Z} = \textsc{Pred}(\boldsymbol{H}_1)$. (d) we use the mean pairwise cosine distance of $\boldsymbol{Z}$ and $\boldsymbol{H_2}$ as the loss (see Eqn. 4.2). (e) $\textsc{Enc}_\theta$ is updated via backpropagation and $\textsc{Enc}_\phi$ is updated via exponential moving average (EMA) from $\textsc{Enc}_\theta$. The BGRL loss is as follows:

$$\mathcal{L}_{\text{BGRL}} = -\frac{2}{n}\sum_{i=0}^{n-1}\frac{\tilde{\boldsymbol{z}}_i \cdot \boldsymbol{h}_i^{(2)}}{||\tilde{\boldsymbol{z}}_i||\ ||\boldsymbol{h}_i^{(2)}||} \tag{4.2}$$

In the next section, we evaluate BGRL and other non-contrastive link prediction methods against contrastive baselines.

[2]Self-GNN [100], which was published independently, also shares the same architecture. As such, we refer to these two methods as BGRL.

4.3 Do Non-Contrastive Learning Methods Perform Well on Link Prediction Tasks?

Several non-contrastive methods have been proposed and have shown effectiveness in node classification [100, 183, 222, 15]. However, none of these methods evaluate or target link prediction tasks. We thus aim to answer the following questions: First, how well do these methods work for link prediction compared to existing contrastive/end-to-end baselines? Second, do they work equally well in both transductive and inductive settings? Finally, if they do work, why; if not, why not?

Differences from Node Classification. Link prediction differs from node classification in several key aspects. First, we must consider the embedding of both the source and destination nodes. Second, we have a much larger set of candidates for the same graph—$O(n^2)$ instead of $O(n)$. Finally, in real applications, link prediction is usually treated as a ranking problem, where we want positive links to be ranked higher than negative links, rather than as a classification problem, e.g. in recommendation systems, where we want to retrieve the top-k most likely links [37, 90]. We discuss this in more detail in Section 4.3.1 below. Given these differences, it is unclear if methods performing well on node classification naturally perform well on link prediction tasks.

Ideal Link Prediction. What does it mean to perform well on link prediction? We clarify this point here. For some nodes $u, v, w \in \mathcal{V}$, let $(u, v) \in \mathcal{E}$ and $(u, w) \notin \mathcal{E}$. Then, an ideal encoder for link prediction would have $\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) < \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_w)$ for some distance function $\textsc{Dist}$. This idea is the core motivation behind margin-loss-based models [210, 75].

4.3.1 Evaluation

Datasets. We use datasets from three different domains: citation networks, co-authorship networks, and co-purchase networks. We use the `Cora` and `Citeseer` citation networks [166], the `Coauthor-CS` and `Coauthor-Physics` co-authorship networks, and the `Amazon-Computers` and `Amazon-Photos` co-purchase networks [168]. We include dataset statistics in Section 4.6.1.

Metric. Following work in the heterogeneous information network [30], knowledge-graph [120], and recommendation systems [37, 90] communities, we choose to use Hits@k over AUC-ROC metrics, since we often empirically prioritize ranking candidate links from a selected node context (e.g. ranking the probability that user A will buy item B, C, or D), as opposed to arbitrarily ranking a randomly chosen positive over negative link (e.g. ranking whether the probability that user A buys item B is more likely than user C does not buy item D). We report Hits@50 ($k = 50$) to strike a balance between the smaller datasets like `Cora` and the larger datasets like `Coauthor-Physics`. However, for completeness of the evaluation, we also include AUC-ROC results in Section 4.6.8.

Decoder. Since our goal is to evaluate the performance of the encoder, we use the same decoder for all of our experiments across all of the methods. The choice of decoder has also been previously studied [198, 196], so we use the best-performing decoder - a Hadamard product MLP. For a candidate link (u, v), we have $\hat{\boldsymbol{Y}} = \textsc{Dec}(\boldsymbol{h}_u * \boldsymbol{h}_v)$ where $*$ represents the Hadamard product, and Dec is a two-layer MLP (with 256 hidden units) followed by a sigmoid. For the self-supervised methods, we first train the encoder and freeze its weights before training the decoder. As a contextual baseline, we also report results on an end-to-end

Table 4.1 Transductive performance of different link prediction methods. We **bold** the best-performing method and underline the second-best method for each dataset. BGRL consistently outperforms other non-contrastive methods and GRACE, and also outperforms ML-GCN, on 3/6 datasets.

	End-To-End	**Contrastive**		**Non-Contrastive**		
Dataset	**E2E-GCN**	**ML-GCN**	**GRACE**	**CCA-SSG**	**GBT**	**BGRL**
`Cora`	**0.816**$_{\pm 0.013}$	0.815$_{\pm 0.002}$	0.686$_{\pm 0.056}$	0.348$_{\pm 0.091}$	0.460$_{\pm 0.149}$	0.792$_{\pm 0.015}$
`Citeseer`	0.822$_{\pm 0.017}$	0.771$_{\pm 0.020}$	0.707$_{\pm 0.068}$	0.249$_{\pm 0.168}$	0.472$_{\pm 0.196}$	**0.858**$_{\pm 0.020}$
`Amazon-Photos`	**0.642**$_{\pm 0.029}$	0.430$_{\pm 0.032}$	0.486$_{\pm 0.025}$	0.369$_{\pm 0.013}$	0.434$_{\pm 0.038}$	0.562$_{\pm 0.013}$
`Amazon-Computers`	**0.426**$_{\pm 0.036}$	0.320$_{\pm 0.060}$	0.240$_{\pm 0.027}$	0.201$_{\pm 0.032}$	0.258$_{\pm 0.008}$	0.346$_{\pm 0.018}$
`Coauthor-CS`	0.762$_{\pm 0.010}$	**0.787**$_{\pm 0.011}$	0.456$_{\pm 0.066}$	0.229$_{\pm 0.018}$	0.298$_{\pm 0.033}$	0.515$_{\pm 0.016}$
`Coauthor-Physics`	0.798$_{\pm 0.018}$	**0.810**$_{\pm 0.003}$	OOM	0.157$_{\pm 0.009}$	0.187$_{\pm 0.011}$	0.476$_{\pm 0.015}$

GCN (E2E-GCN), for which we train the encoder and decoder jointly, backpropagating a binary cross-entropy loss on link existence.

Transductive Evaluation

Transductive Setting. We first evaluate the performance of the methods in the transductive setting, where we train on $G_{train} = (\mathcal{V}, \mathcal{E}_{train})$ for $\mathcal{E}_{train} \subset \mathcal{E}$, validate our method on $G_{val} = (\mathcal{V}, \mathcal{E}_{val})$ for $\mathcal{E}_{val} \subset (\mathcal{E} - \mathcal{E}_{train})$, and test on $G_{test} = (\mathcal{V}, \mathcal{E}_{test})$ for $\mathcal{E}_{test} = \mathcal{E} - \mathcal{E}_{train} - \mathcal{E}_{val}$. Note that the same nodes are present in training, validation, and testing. We also do not introduce any new edges during inference time—inference is performed on $\mathcal{E}_{train}$.

Results. The results of our evaluation are shown in Table 4.1. As expected, the end-to-end GCN generally performs the best across all of the datasets. We also find that CCA-SSG and GBT similarly perform poorly relative to the other methods. This is

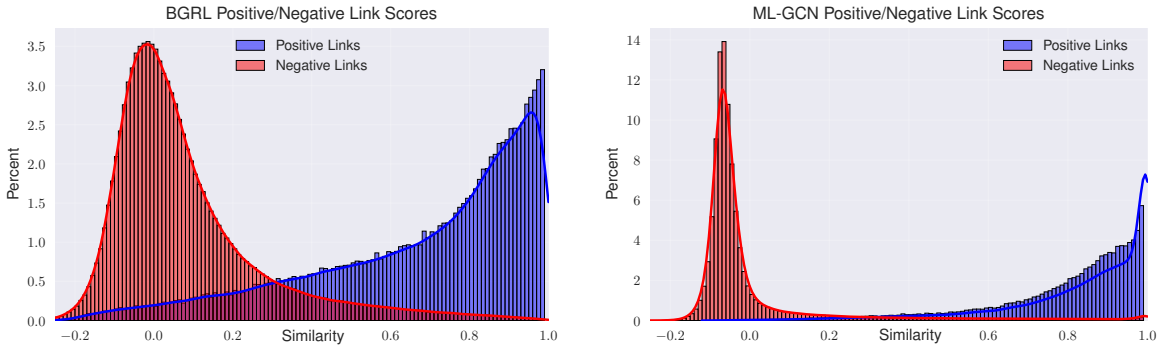


Figure 4.1 These plots show similarities between node embeddings. **Left:** distribution of positive/negative link similarities for BGRL. **Right:** distribution of positive/negative link similarities for ML-GCN. We can see that while they behave similarly, the ML-GCN does a better job of ensuring that positive/negative links are well separated. These scores are computed on `Amazon-Photos`.

intuitive, as neither method was designed for link prediction and were only evaluated for node classification in their respective papers. Surprisingly, however, BGRL outperforms the ML-GCN (the strongest contrastive baseline) on 3/6 of the datasets and performs similarly on 1 other (`Cora`). It also outperforms GRACE across all of the datasets.

Understanding BGRL Performance. Interestingly, we find that BGRL exhibits similar behavior to the ML-GCN on many datasets, despite the BGRL loss function (see Equation (4.2)) not explicitly optimizing for this. Relative to an anchor node u, we can express the max-margin loss of the ML-GCN as follows:

$$L(u) = \mathbb{E}_{v \sim \mathcal{N}(u)} \left[\mathbb{E}_{w \sim \mathcal{E} - \mathcal{N}(u)} J(u, v, w) \right] \tag{4.3}$$

where $J(u, v, w)$ is the margin ranking loss for an anchor u, positive sample v, and negative w:

$$J(u, v, w) = \max\{0, \boldsymbol{h}_u \cdot \boldsymbol{h}_v - \boldsymbol{h}_u \cdot \boldsymbol{h}_w + \Delta\} \tag{4.4}$$

and Δ is a hyperparameter for the size of the margin. This seemingly explicitly optimizes for the aforementioned ideal link prediction behavior (anchor-aware ranking of positive over

negative links). Despite these circumstances, Figure 4.1 shows that both BGRL and ML-GCN both clearly separate positive and negative samples, although ML-GCN pushes them further apart. We provide some intuition on why this may occur in Section 4.6.10 below.

Why Does BGRL Not Collapse? The loss function for BGRL (see Equation (4.2)) is 0 when $h_i^{(2)} = 0$ or $\tilde{\boldsymbol{z}}_i = 0$, i.e., the loss is minimized when the model produces all-zero outputs. While theoretically possible, this is clearly undesirable behavior since this does not result in useful embeddings. We refer to this case as model collapse. It is not fully understood why non-contrastive models do not collapse, but there have been several reasons proposed in the image domain with both theoretical and empirical grounding. We discuss this more in Section 4.6.9. Consistent with the findings from Thakoor et al. [183], we find that collapse does not occur in practice (with reasonable hyperparameter selection).

Conclusion. We find that CCA-SSG and GBT generally perform poorly compared to contrastive baselines. Surprisingly, we find that BGRL generally performs well in the transductive setting by successfully separating positive and negative link distance distributions. However, this setting may not be representative of real-world problems. In the next section, we evaluate the methods in the more realistic inductive setting to see if this performance holds.

Inductive Evaluation

Inductive Setting. While we observe some promising results in favor of non-contrastive methods (namely, BGRL) in the transductive setting, we note that this setting is not entirely realistic. In practice, we often have both new nodes and edges introduced at inference time after our model is trained. For example, consider a social network upon which

a model is trained at some time t_1 but is used for inference (for a GNN, this refers to the message-passing step) at time t_2, where new users and friendships have been added to the network in the interim. Then, the goal of a model run at time t_2 would be to predict any new links at new network state t_3 (although we assume there are no new nodes introduced at that step since we cannot compute the embedding of nodes without performing inference on them first). To simulate this setting, we first partition the graph into two sets of nodes: "observed" nodes (that we see during training) and "unobserved nodes" (that are only used for inference and testing). We then withhold a portion of the edges at each of the time steps t_3, t_2, t_1 to serve as testing-only, inference-only, and training-only edges, respectively. We describe this process in more detail in Section 4.6.4.

Results. Table 4.2 shows that in the inductive setting, BGRL is outperformed by the contrastive ML-GCN on *all* datasets. It still outperforms CCA-SSG and GBT, but it is *much* less competitive in the inductive setting. We next ask: what accounts for this large difference in performance?

Why Does BGRL Not Work Well in the Inductive Setting? One possible reason for the poor performance of BGRL in the inductive setting is that it is unable to correctly differentiate unseen positives from unseen negatives, i.e., it is overfitting on the training graph. Intuitively, this could happen due to a lack of negative samples—BGRL never pushes samples away from each other. We show that this is indeed the case in Figure 4.2, where BGRL's negative link score distribution has heavy overlap with its positive link score distribution. We can also see this behavior in Figure 4.1 where the ML-GCN does a clearly better job of pushing positive/negative samples far apart, despite BGRL's surprising success.

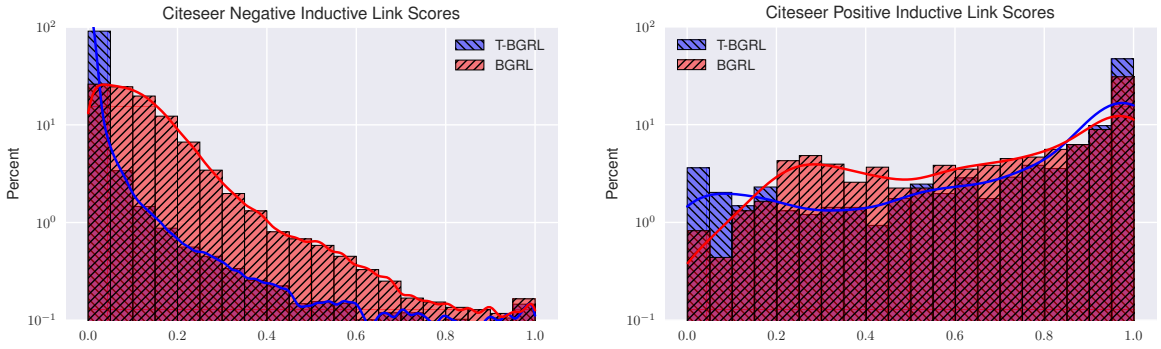


Figure 4.2 These plots show similarities between node embeddings on `Citeseer`. **Left:** distribution of similarity to *non-neighbors* for TenGAN and BGRL. **Right:** distribution of similarity to *neighbors* for TenGAN and BGRL. Note that the y-axis is on a logarithmic scale. TenGAN clearly does a better job of ensuring that negative link representations are pushed far apart from those of positive links.

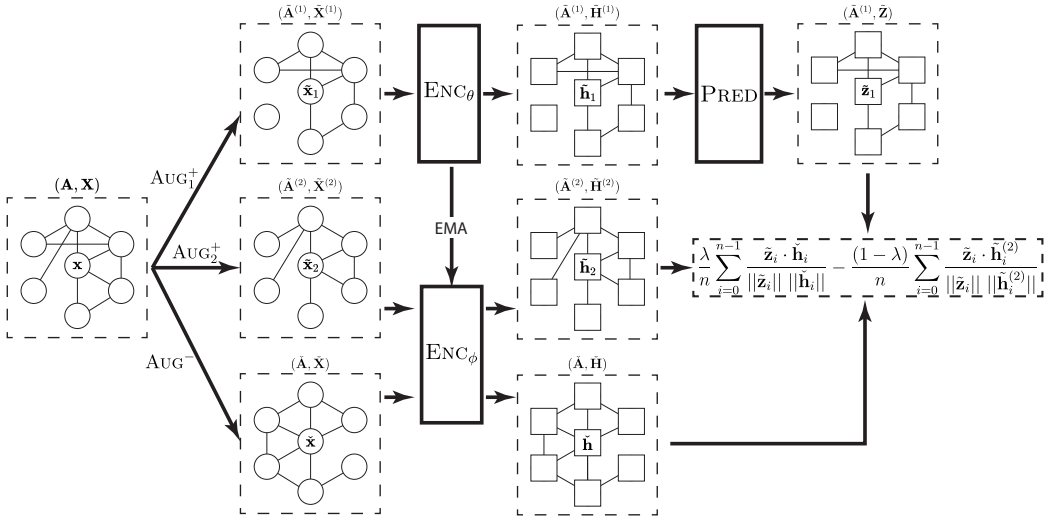


Figure 4.3 TenGAN architecture diagram. The loss function is also shown in Equation (4.5).

Naturally, improving the separation between these distributions increases the chance of a correct prediction. We investigate this hypothesis in Section 4.4 below and propose TenGAN (Figure 4.3), a novel method to help alleviate this issue.

4.4 Improving Inductive Performance in a Non-Contrastive Framework

In order to reduce this systematic gap in performance between ML-GCN (the best-performing contrastive model) and BGRL (the best-performing non-contrastive model), we observe that we need to push negative and positive node pair representations further apart. This way, pairs between new nodes—introduced at inference time—have a higher chance of being classified correctly. Contrastive methods utilize negative sampling for this purpose, but we wish to avoid negative sampling owing to high computational cost. In lieu of this, we propose a simple, yet powerfully effective idea below.

Model Intuition. To emulate the effect of negative sampling without actually performing it, we propose Triplet-BGRL (TenGAN). In addition to the two augmentations performed during standard non-contrastive SSL training, we add a corruption to function as a cheap negative sample. For each node, like BGRL, we minimize the distance between its representations across two augmentations. However, taking inspiration from triplet-style losses [83], we also maximize the distance between the augmentation and corruption representations.

Model Design. Ideally, this model should not only perform better than BGRL in the inductive setting, but should also have the same time complexity as BGRL. In order to meet these expectations, we design efficient, linear-time corruptions (same asymptotic runtime as the augmentations). We also choose to use the online encoder Enc_ϕ to generate embeddings for the corrupted graph so that TenGAN does not have any additional parameters. Figure 4.3

Algorithm 1: PyTorch-style pseudocode for TenGAN

```
# Enc_o: online encoder network
# Enc_t: target encoder network
# Pred: predictor network
# lam: trade-off
# decay: EMA decay parameter
# g: input graph
# feat: node features

g1, feat1 = augment(g, feat) # augmentation #1
g2, feat2 = augment(g, feat) # augmentation #2
c_g, c_feat = corrupt(g, feat) # corruption

h1 = Enc_o(g1, feat1)
h2 = Enc_t(g2, feat2)
c_z = Enc_t(c_g, c_feat)
z1 = Pred(z1)

loss = lam*cosine_similarity(z1, c_z) \
       - (1-lam)*cosine_similarity(z1, h2)
loss.backward() # backprop

# Update Enc_t with EMA
Enc_t.params = decay * Enc_t.params \
               + (1-decay) * Enc_o.params
```

illustrates the overall architecture of the proposed TenGAN, and Algorithm 1 presents PyTorch-style pseudocode. Our new proposed loss function is as follows:

$$\mathcal{L}_{\text{TenGAN}} = \underbrace{\frac{\lambda}{n}\sum_{i=0}^{n-1}\frac{\tilde{\boldsymbol{z}}_i \cdot \check{\boldsymbol{h}}_i}{||\tilde{\boldsymbol{z}}_i||\ ||\check{\boldsymbol{h}}_i||}}_{\text{TenGAN Loss Term}} \underbrace{-\frac{(1-\lambda)}{n}\sum_{i=0}^{n-1}\frac{\tilde{\boldsymbol{z}}_i \cdot \boldsymbol{h}_i^{(2)}}{||\tilde{\boldsymbol{z}}_i||\ ||\boldsymbol{h}_i^{(2)}||}}_{\text{BGRL Loss}} \tag{4.5}$$

where λ is a hyperparameter controlling the repulsive forces between augmentation and corruption.

Corruption Choice. We experiment with several different corruptions methods, but limit ourselves to linear-time corruptions in order to maintain the efficiency of BGRL. We find that $\textsc{ShuffleFeatRandomEdge}(\boldsymbol{A}, \boldsymbol{X}) = (\check{\boldsymbol{A}}, \check{\boldsymbol{X}})$, where $\check{\boldsymbol{A}} \sim \{0,1\}^{n \times n}$ and $\check{\boldsymbol{X}} = \textsc{ShuffleRows}(\boldsymbol{X})$ works the best. We describe each of the different corruptions we experimented with in Section 4.6.7.

Inductive Results. Table 4.2 shows that TenGAN improves inductive performance over BGRL in 5/6 datasets, with very large improvements in the `Cora` and `Citeseer` datasets. The only dataset where BGRL outperformed TenGAN is the `Amazon-Photos` dataset. However, this gap is much smaller (0.01 difference in Hits@50) than the improvements on the other datasets. We plot the scores output by the decoder for unseen negative pairs compared to those for unseen positive pairs in Figure 4.2. We can see that TenGAN pushes apart unseen negative and positive pairs much better than BGRL.

Transductive Results. We also evaluate the performance of TenGAN in the transductive setting to ensure that it does not significantly reduce performance when compared to BGRL. See Table 4.3 on the right for the results.

Difference from Contrastive Methods. While our method shares some similarities with contrastive methods, we believe TenGAN is strictly non-contrastive because it does

not require the $O(n^2)$ sampling from the complement of the edge index used by contrastive methods. This is clearly shown in Figure 4.4, where T-BGRL and BGRL have similar runtimes and are much faster than GRACE and ML-GCN. The corruption can be viewed as a "negative" augmentation—with the only difference being that it changes the expected label for each link. In fact, one of the corruptions that we consider, SparsifyFeatSparsifyEdge, is essentially the same as the augmentations using by BGRL (except with much higher drop probability). We discuss other corruptions below in Section 4.6.7.

Scalability. We evaluate the runtime of our model on different datasets. Figure 4.4 shows the running times to fully train a model for different contrastive and non-contrastive methods over 5 different runs. Note that we use a fixed 10,000 epochs for GRACE, CCA-SSG, GBT, BGRL, and TenGAN, but use early stopping on the ML-GCN with a maximum of 1,000 epochs. We find that (i) TenGAN is comparable to BGRL in runtime owing to efficient choices of corruptions, (ii) it is about 4.3× faster than GRACE on `Amazon-Computers` (the largest dataset which GRACE can run on), and (ii) it is 14× faster than ML-GCN. CCA-SSG is the fastest of all the methods but performs the worst. As mentioned above, we do not compare with SEAL [224] or other subgraph-based methods due to how slow they are during inference. SUREL [209] is ~250× slower, and SEAL [224] is about ~3900× slower according to [209]. In conclusion, we find that TenGAN is roughly as

Table 4.3 Transductive performance of TenGAN compared to ML-GCN and BGRL (same numbers as Table 4.1 above; full figure in Table 4.5).

Dataset	ML-GCN	BGRL	TenGAN
Cora	**0.815**	0.792	0.773±0.020
Citeseer	0.771	0.858	**0.868**±0.023
Coauthor-Cs	**0.787**	0.515	0.555±0.009
Coauthor-Physics	**0.810**	0.476	0.471±0.021
Amazon-Computers	0.320	**0.346**	0.315±0.015
Amazon-Photos	0.430	**0.562**	0.517±0.016

scalable as other non-contrastive methods, and much more scalable than existing contrastive methods.

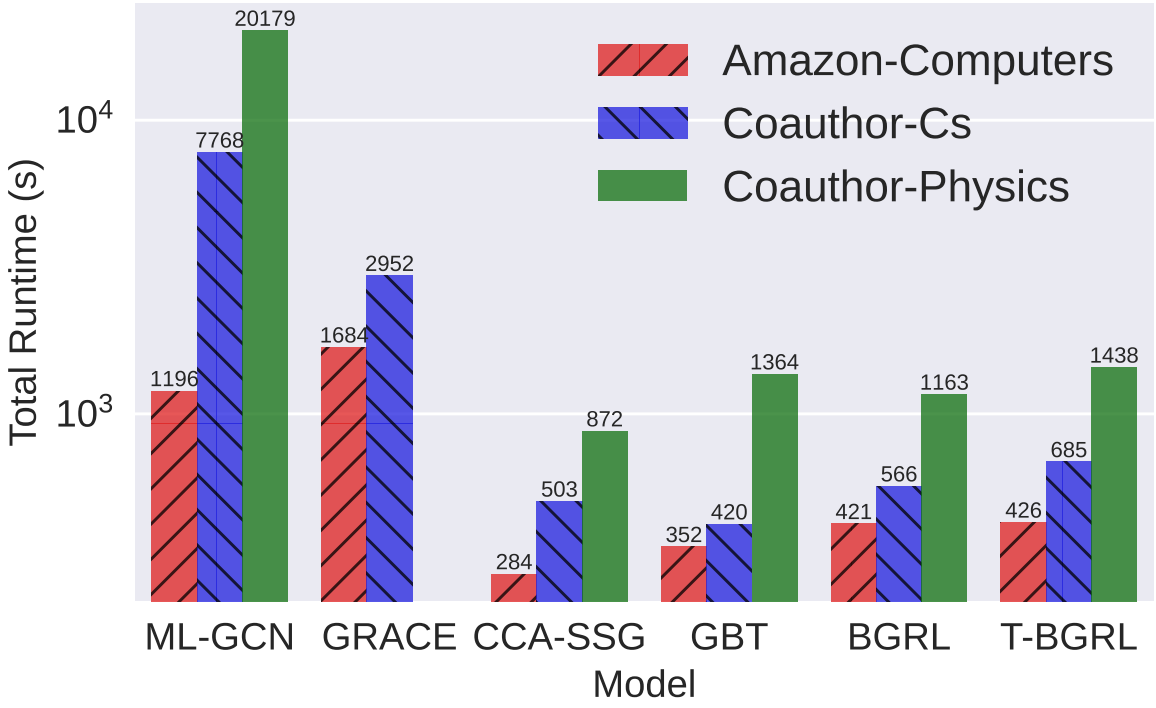


Figure 4.4 Total runtime comparison of different contrastive and non-contrastive methods. T-BGRL and BGRL have relatively similar runtimes and are significantly faster than the contrastive methods (GRACE and ML-GCN).

4.5 Other Related Work

Link Prediction. Link prediction is a long-standing graph machine learning task. Some traditional methods include (i) matrix [130, 193] or tensor factorization [2, 46] methods which factor the adjacency and/or feature matrices to derive node representations which can predict links equipped with inner products, and (ii) heuristic methods which score node pairs based on neighborhood and overlap [216, 218, 151]. Several shallow graph embedding methods [67, 149] which train node embeddings by random-walk strategies have also been

used for link prediction. In addition to the node-embedding-based GNN methods mentioned in Section 4.2, several works [209, 224, 77] propose subgraph-based methods for this task, which aim to classify subgraphs around each candidate link. Few works focus on scalable link prediction with distillation [73], decoder [196], and sketching designs [28].

Graph SSL Methods. Most graph SSL methods can be put categorized into contrastive and non-contrastive methods. Contrastive learning has been applied to link prediction with margin-loss-based methods such as PinSAGE [210], and GraphSAGE [75], where negative sample representations are pushed apart from positive sample representations. GRACE [236] uses augmentation [228] during this negative sampling process to further increase the performance of the model. DGI [186] leverages mutual information maximization between local patch and global graph representations. Some works [97, 94] also explore using multiple contrastive pretext tasks for SSL. Several works [215, 119] also focus on graph-level contrastive learning, via graph-level augmentations and careful negative selection. Recently, non-contrastive methods have been applied to graph representation learning. Self-GNN [100] and BGRL [183] use ideas from BYOL [66] and SimSiam [33] to propose a graph framework that does not require negative sampling. We describe BGRL in depth in Section 4.2 above. Graph Barlow Twins (GBT) [15] is adapted from the Barlow Twins model in the image domain [219] and uses cross-correlation to learn node representations with a shared encoder. CCA-SSG [222] uses ideas from Canonical Correlation Analysis (CCA) [85] and Deep CCA [5] for their loss function. These models are somewhat similar in that it has also been shown that Barlow Twins is equivalent to Kernel CCA [10].

4.6 Conclusion

To our knowledge, this is the first work to study non-contrastive SSL methods and their performance on link prediction. We first evaluate several contrastive and non-contrastive graph SSL methods on link prediction tasks, and find that surprisingly, one popular non-contrastive method (BGRL) is able to perform well in the transductive setting. We also observe that BGRL struggles in the inductive setting, and identify that it has a tendency to overfit the training graph, indicating it fails to push positive and negative node pair representations far apart from each other. Armed with these insights, we propose TENGAN, a simple but effective non-contrastive strategy which works by generating extremely cheap "negatives" by corrupting the original inputs. TENGANsidesteps the expensive negative sampling step evident in contrastive learning, while enjoying strong performance benefits. TENGANimproves on BGRL's inductive performance in 5/6 datasets while achieving similar transductive performance, making it comparable to the best contrastive baselines, but with a 14× speedup over the best contrastive methods.

Reproducibility Statement

To ensure reproducibility, our source code is available online at `https://github.com/snap-research/non-contrastive-link-prediction`. The hyperparameters and instructions for reproducing all experiments are provided in the `README.md` file.

Table 4.2 Performance of various methods in the inductive setting. See Section 4.3.1 for an explanation of our inductive setting. Although we do not introduce TenGAN until Section 4.4, we include the results here to save space.

	End-To-End	**Contrastive**		**Non-Contrastive**			
Dataset	**E2E-GCN**	**ML-GCN**	**GRACE**	**GBT**	**CCA-SSG**	**BGRL**	**T-BGRL**
			Overall				
Cora	0.523±0.019	0.490±0.028	0.448±0.043	0.135±0.077	0.120±0.018	0.324±0.184	**0.568**±0.033
Citeseer	0.621±0.034	0.661±0.036	0.514±0.053	0.305±0.026	0.170±0.071	0.526±0.055	**0.727**±0.027
Coauthor-Cs	0.484±0.048	**0.572**±0.037	0.313±0.017	0.182±0.025	0.176±0.013	0.438±0.025	0.534±0.026
Coauthor-Physics	0.386±0.016	**0.550**±0.059	OOM	0.112±0.014	0.037±0.051	0.439±0.013	0.463±0.023
Amazon-Computers	0.179±0.010	0.279±0.044	0.212±0.057	0.172±0.015	0.155±0.013	0.270±0.034	**0.312**±0.027
Amazon-Photos	0.420±0.123	**0.478**±0.008	0.262±0.010	0.289±0.032	0.182±0.072	0.460±0.023	0.450±0.017
			Performance on Observed-Observed Node Edges				
Cora	0.574±0.020	0.490±0.029	0.557±0.038	0.149±0.084	0.124±0.026	0.345±0.196	**0.624**±0.027
Citeseer	0.610±0.023	0.621±0.021	0.602±0.050	0.358±0.031	0.197±0.082	0.605±0.045	**0.768**±0.021
Coauthor-Cs	0.504±0.047	**0.591**±0.034	0.332±0.018	0.187±0.023	0.177±0.013	0.462±0.025	0.535±0.026
Coauthor-Physics	0.390±0.015	**0.566**±0.058	OOM	0.117±0.014	0.039±0.054	0.445±0.012	0.469±0.023
Amazon-Computers	0.177±0.009	0.278±0.044	0.212±0.059	0.169±0.016	0.155±0.014	0.270±0.034	**0.313**±0.027
Amazon-Photos	0.418±0.123	**0.483**±0.009	0.265±0.011	0.295±0.031	0.185±0.070	0.467±0.023	0.457±0.015
			Performance on Observed-Unobserved Node Edges				
Cora	0.462±0.023	0.487±0.021	0.367±0.045	0.128±0.075	0.115±0.014	0.309±0.175	**0.528**±0.037
Citeseer	0.645±0.055	0.705±0.039	0.458±0.063	0.280±0.024	0.148±0.067	0.487±0.064	**0.708**±0.034
Coauthor-Cs	0.459±0.049	**0.545**±0.042	0.284±0.017	0.175±0.026	0.177±0.013	0.402±0.025	0.536±0.027
Coauthor-Physics	0.379±0.019	**0.525**±0.058	OOM	0.106±0.013	0.035±0.048	0.429±0.013	0.455±0.022
Amazon-Computers	0.183±0.010	0.281±0.045	0.213±0.056	0.177±0.014	0.155±0.011	0.270±0.034	**0.312**±0.027
Amazon-Photos	0.424±0.123	**0.470**±0.007	0.258±0.011	0.279±0.032	0.178±0.076	0.449±0.022	0.439±0.021
			Performance on Unobserved-Unobserved Node Edges				
Cora	0.239±0.027	**0.507**±0.063	0.252±0.066	0.100±0.076	0.125±0.020	0.287±0.164	0.463±0.065
Citeseer	0.595±0.073	**0.681**±0.101	0.287±0.039	0.137±0.019	0.126±0.043	0.271±0.078	0.595±0.045
Coauthor-Cs	0.372±0.043	0.483±0.046	0.230±0.019	0.159±0.037	0.157±0.011	0.341±0.032	**0.517**±0.032
Coauthor-Physics	0.365±0.024	**0.505**±0.065	OOM	0.098±0.013	0.034±0.047	0.424±0.014	0.445±0.026
Amazon-Computers	0.183±0.008	0.275±0.046	0.214±0.052	0.181±0.015	0.155±0.012	0.265±0.032	**0.305**±0.029
Amazon-Photos	0.419±0.126	**0.461**±0.014	0.251±0.010	0.265±0.044	0.172±0.084	0.442±0.028	0.416±0.027

4.6.1 Dataset Statistics

Table 4.4 Statistics for the datasets used in our work.

Dataset	Nodes	Edges	Features
`Cora`	2,708	5,278	1,433
`Citeseer`	3,327	4,552	3,703
`Coauthor-Cs`	18,333	163,788	6,805
`Coauthor-Physics`	34,493	495,924	8,415
`Amazon-Computers`	13,752	491,722	767
`Amazon-Photos`	7,650	238,162	745

4.6.2 Machine Details

We run all of our experiments on either NVIDIA P100 or V100 GPUs. We use machines with 12 virtual CPU cores and 24 GB of RAM for the majority of our experiments. We exclusively use V100s for our timing experiments. We ran our experiments on Google Cloud Platform.

4.6.3 Transductive Setting Details

We use an 85/5/10 split for training/validation/testing data—following Zhang and Chen [224], Cai et al. [22].

4.6.4 Inductive Setting Details

The inductive setting represents a more realistic setting than the transductive setting. For example, consider a social network upon which a model is trained at some time

t_1 but is used for inference (for a GNN, this refers to the message-passing step) at time t_2, where new users and friendships have been added to the network in the interim. Then, the goal of a model run at time t_2 would be to predict any new links at new network state t_3 (although we assume there are no new nodes introduced at that step since we cannot compute the embedding of nodes without performing inference on them first). To simulate this setting, we first perform the following steps:

1. We withhold a portion of the edges (and the same number of disconnected node pairs) to use as testing-only edges.
2. We partition the graph into two sets of nodes: "observed" nodes (that we see during training) and "unobserved nodes" (that can only be seen during testing).
3. We mask out some edges to use as testing-only edges.
4. We mask out some edges to use as inference-only edges.
5. We mask out some edges to use as validation-only edges.
6. We mask out some edges to use as training-only edges.

As the test edges are sampled before the node split, there will be three kinds of them after the splitting. Specifically: edges within observed nodes, edges between observed nodes and unobserved nodes, and edges within unobserved nodes. For ease of data preparation, we use the same percentages for the test edge splitting, unobserved node splitting, and validation edge splitting. Specifically, we mask out 30% of the edges (at each of the above stages) on the small datasets (`Cora` and `Citeseer`), and 10% on all the other datasets. We use a 30%

split on the small datasets to ensure that we have a sufficient number of edges for testing and validation purposes.

4.6.5 Experimental Setup

To ensure that we fairly evaluate each model, we run a Bayesian hyperparameter sweep for 25 runs across each model-dataset combination with the target metric being the validation Hits@50. Each run is the result of the mean averaged over 5 runs (retraining both the encoder and decoder). We used the Weights and Biases [16] Bayesian optimizer for our experiments. We provide a sample configuration file to reproduce our sweeps, as well as the exact parameters used for the top TenGAN runs shown in our tables.

We used the reference GRACE implementation and BGRL implementation but modified them for link prediction instead of node classification. We based our E2E-GCN off of the OGB [87] implementation. We re-implemented CCA-SSG and GBT. The code for all of our implementations and modifications can be found in the link in our paper above.

4.6.6 Full Results

Table 4.5 shows the results of all the methods (including TenGAN) on transductive setting.

Table 4.5 Full transductive performance table (combination of Tables 4.1 and 4.3).

	End-To-End	Contrastive		Non-Contrastive			
Dataset	**E2E-GCN**	**ML-GCN**	**GRACE**	**CCA-SSG**	**GBT**	**BGRL**	**T-BGRL**
Cora	**0.816**$_{\pm 0.013}$	<u>0.815</u>$_{\pm 0.002}$	0.686$_{\pm 0.056}$	0.348$_{\pm 0.091}$	0.460$_{\pm 0.149}$	0.792$_{\pm 0.015}$	0.773$_{\pm 0.020}$
Citeseer	0.822$_{\pm 0.017}$	0.771$_{\pm 0.020}$	0.707$_{\pm 0.068}$	0.249$_{\pm 0.168}$	0.472$_{\pm 0.196}$	<u>0.858</u>$_{\pm 0.020}$	**0.868**$_{\pm 0.023}$
Amazon-Photos	0.642$_{\pm 0.029}$	0.430$_{\pm 0.032}$	0.486$_{\pm 0.025}$	0.369$_{\pm 0.013}$	0.434$_{\pm 0.038}$	0.562$_{\pm 0.013}$	0.517$_{\pm 0.016}$
Amazon-Computers	0.426$_{\pm 0.036}$	0.320$_{\pm 0.060}$	0.240$_{\pm 0.027}$	0.201$_{\pm 0.032}$	0.258$_{\pm 0.008}$	0.346$_{\pm 0.018}$	0.315$_{\pm 0.015}$
Coauthor-Cs	0.762$_{\pm 0.010}$	0.787$_{\pm 0.011}$	0.456$_{\pm 0.066}$	0.229$_{\pm 0.018}$	0.298$_{\pm 0.033}$	0.515$_{\pm 0.016}$	0.555$_{\pm 0.009}$
Coauthor-Physics	0.798$_{\pm 0.018}$	0.810$_{\pm 0.003}$	OOM	0.157$_{\pm 0.009}$	0.187$_{\pm 0.011}$	0.476$_{\pm 0.015}$	0.471$_{\pm 0.021}$

4.6.7 Corruptions

In this work, we experiment with the following corruptions:

1. RandomFeatRandomEdge: Randomly generate an adjacency matrix $\tilde{\boldsymbol{A}}$ and $\tilde{\boldsymbol{X}}$ with the same sizes as $\boldsymbol{A}$ and $\boldsymbol{X}$, respectively. Note that $\tilde{\boldsymbol{A}}$ and $\boldsymbol{A}$ also have the same number of non-zero entries, i.e., the same number of edges.

2. ShuffleFeatRandomEdge: Randomly shuffle the rows of $\boldsymbol{X}$, and generate a random $\tilde{\boldsymbol{A}}$ with the same size as $\boldsymbol{A}$. Note that $\tilde{\boldsymbol{A}}$ and $\boldsymbol{A}$ also have the same number of non-zero entries, i.e., the same number of edges.

3. SparsifyFeatSparsifyEdge: Mask out a large percentage (we chose 95%) of the entries in $\boldsymbol{X}$ and $\boldsymbol{A}$.

Of these corruptions, we find that ShuffleFeatRandomEdge works the best across our experiments.

Table 4.6 Area under the ROC curve for the methods in the transductive setting.

	End-To-End	Contrastive		Non-Contrastive			
Dataset	**E2E-GCN**	**ML-GCN**	**GRACE**	**CCA-SSG**	**GBT**	**BGRL**	**T-BGRL**
Cora	**0.911**$_{\pm 0.004}$	0.893$_{\pm 0.007}$	0.883$_{\pm 0.020}$	0.647$_{\pm 0.076}$	0.736$_{\pm 0.109}$	**0.911**$_{\pm 0.008}$	0.910$_{\pm 0.005}$
Citeseer	0.922$_{\pm 0.006}$	0.891$_{\pm 0.006}$	0.863$_{\pm 0.042}$	0.661$_{\pm 0.050}$	0.755$_{\pm 0.120}$	0.934$_{\pm 0.009}$	**0.953**$_{\pm 0.003}$
Coauthor-Cs	0.964$_{\pm 0.005}$	**0.966**$_{\pm 0.001}$	0.961$_{\pm 0.003}$	0.758$_{\pm 0.047}$	0.894$_{\pm 0.017}$	0.959$_{\pm 0.002}$	0.956$_{\pm 0.002}$
Coauthor-Physics	0.978$_{\pm 0.001}$	**0.986**$_{\pm 0.000}$	OOM	0.821$_{\pm 0.051}$	0.834$_{\pm 0.084}$	0.961$_{\pm 0.002}$	0.963$_{\pm 0.001}$
Amazon-Computers	**0.985**$_{\pm 0.001}$	0.983$_{\pm 0.001}$	0.951$_{\pm 0.011}$	0.907$_{\pm 0.025}$	0.946$_{\pm 0.007}$	0.969$_{\pm 0.002}$	0.976$_{\pm 0.001}$
Amazon-Photos	**0.989**$_{\pm 0.000}$	0.983$_{\pm 0.002}$	0.981$_{\pm 0.001}$	0.939$_{\pm 0.008}$	0.956$_{\pm 0.015}$	0.980$_{\pm 0.000}$	0.982$_{\pm 0.000}$

4.6.8 AUC-ROC Results

Here we include the area under the ROC curve for each of the different models under both the inductive and transductive settings. Note that we perform early stopping on the validation Hits@50 when training the link prediction model, not on the validation AUC-ROC.

4.6.9 Why Does BGRL Not Collapse?

The loss function for BGRL (see Equation (4.2)) is 0 when $h_i^{(2)} = 0$ or $\tilde{\boldsymbol{z}}_i = 0$. While theoretically possible, this is clearly undesirable behavior since this does not result in useful embeddings. We refer to this case as the model collapsing. It is not fully understood why non-contrastive models do not collapse, but there have been several reasons proposed in the image domain with both theoretical and empirical grounding. Chen and He [33] showed that the SimSiam architecture requires both the predictor and the stop gradient. This has also been shown to be true for BGRL. Tian et al. [184] claim that the eigenspace of predictor weights will align with the correlation matrix of the online network under the assumption

Table 4.7 AUC-ROC of various methods in the inductive setting. See Section 4.3.1 for an explanation of our inductive setting.

	End-To-End	**Contrastive**		**Non-Contrastive**			
Dataset	**E2E-GCN**	**ML-GCN**	**GRACE**	**GBT**	**CCA-SSG**	**BGRL**	**T-BGRL**
			Overall				
Cora	0.788±0.015	0.842±0.008	0.858±0.012	0.704±0.032	0.595±0.035	0.814±0.022	**0.920**±0.008
Citeseer	0.810±0.016	0.873±0.004	0.886±0.010	0.691±0.007	0.621±0.070	0.891±0.006	**0.954**±0.003
Coauthor-Cs	0.881±0.040	0.956±0.001	0.944±0.001	0.875±0.036	0.831±0.068	**0.968**±0.001	0.958±0.001
Coauthor-Physics	0.957±0.004	**0.976**±0.001	OOM	0.818±0.092	0.614±0.050	0.974±0.001	**0.976**±0.001
Amazon-Computers	0.974±0.009	0.981±0.001	0.972±0.012	0.919±0.023	0.910±0.031	0.980±0.002	**0.982**±0.002
Amazon-Photos	0.976±0.003	0.982±0.001	0.977±0.002	0.962±0.011	0.885±0.057	**0.984**±0.000	0.981±0.001
			Performance on Observed-Observed Node Edges				
Cora	0.827±0.011	0.834±0.010	0.883±0.010	0.714±0.034	0.584±0.047	0.800±0.025	**0.929**±0.005
Citeseer	0.792±0.014	0.840±0.013	0.905±0.012	0.705±0.019	0.635±0.078	0.875±0.006	**0.956**±0.005
Coauthor-Cs	0.886±0.037	0.951±0.001	0.947±0.002	0.874±0.037	0.828±0.070	**0.967**±0.001	0.955±0.002
Coauthor-Physics	0.959±0.004	**0.976**±0.001	OOM	0.819±0.091	0.615±0.049	0.974±0.001	0.975±0.001
Amazon-Computers	0.974±0.010	**0.981**±0.001	0.971±0.012	0.918±0.024	0.910±0.030	0.979±0.002	**0.981**±0.002
Amazon-Photos	0.976±0.003	0.981±0.001	0.977±0.002	0.962±0.011	0.885±0.055	**0.983**±0.000	0.981±0.001
			Performance on Observed-Unobserved Node Edges				
Cora	0.741±0.022	0.844±0.010	0.840±0.017	0.696±0.030	0.602±0.024	0.818±0.023	**0.912**±0.010
Citeseer	0.841±0.019	0.901±0.005	0.877±0.012	0.687±0.016	0.610±0.069	0.904±0.006	**0.955**±0.004
Coauthor-Cs	0.877±0.045	0.964±0.001	0.940±0.001	0.876±0.036	0.836±0.067	**0.969**±0.001	0.964±0.001
Coauthor-Physics	0.953±0.004	0.975±0.001	OOM	0.817±0.093	0.612±0.052	0.975±0.001	**0.976**±0.000
Amazon-Computers	0.974±0.009	0.981±0.001	0.973±0.011	0.921±0.022	0.909±0.032	0.980±0.002	**0.982**±0.002
Amazon-Photos	0.977±0.003	0.983±0.001	0.977±0.002	0.963±0.012	0.884±0.059	**0.986**±0.000	0.981±0.002
			Performance on Unobserved-Unobserved Node Edges				
Cora	0.571±0.043	0.879±0.016	0.810±0.019	0.693±0.039	0.626±0.040	0.866±0.024	**0.911**±0.011
Citeseer	0.852±0.047	0.917±0.021	0.827±0.029	0.637±0.052	0.599±0.062	0.916±0.012	**0.941**±0.011
Coauthor-Cs	0.850±0.059	0.964±0.001	0.928±0.004	0.877±0.034	0.839±0.061	**0.967**±0.002	0.966±0.001
Coauthor-Physics	0.949±0.006	0.978±0.001	OOM	0.818±0.091	0.613±0.056	0.978±0.001	**0.981**±0.001
Amazon-Computers	0.970±0.010	**0.979**±0.001	0.969±0.011	0.914±0.022	0.899±0.035	0.977±0.002	**0.979**±0.003
Amazon-Photos	0.978±0.004	0.982±0.002	0.977±0.002	0.965±0.011	0.886±0.063	**0.986**±0.001	0.982±0.003

of a one-layer linear encoder and a one-layer linear predictor. Wen and Li [201] looked at the case of a two-layer non-linear encoder with output normalization and found that the predictor is often only useful during the learning process and often converges to the identity function. We did not observe this behavior on BGRL—the predictor is usually significantly different from that of the identity function.

4.6.10 How Does BGRL Pull Representations Closer Together?

Here we clarify the intuition behind BGRL pulling similar points together. To simplify this analysis, we assume that the predictor is the identity function, which Wen and Li [201] found is true in the image representation learning setting. Although we have not observed this in the graph setting, this assumption greatly simplifies our analysis, and we argue it is sufficient to understand why BGRL works.

Suppose we have three nodes: an anchor node u, a neighbor v, and a non-neighbor w. That is, we have $(u, v) \in \mathcal{E}$, $(u, w) \notin \mathcal{E}$, and $(v, w) \notin \mathcal{E}$. Let $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}$ be the embeddings for u, v, w, respectively (e.g. $\boldsymbol{u} = \textsc{Enc}(u)$).

Assuming homophily between the nodes, we have $\boldsymbol{u} \cdot \boldsymbol{v} < \boldsymbol{u} \cdot \boldsymbol{w}$.

We then apply the two augmentations on u, producing $\tilde{u}_1 = \textsc{Aug}_1(u)$ and $\tilde{u}_2 = \textsc{Aug}_2(u)$. For the sake of simplicity, let us assume that we perform edge dropping and feature dropping with the same probability p (in practice, they may be different from each other). We represent the space of possible values for $\tilde{u}_1$ and $\tilde{u}_2$ as a circle with radius r centered at u, where r is controlled by p.

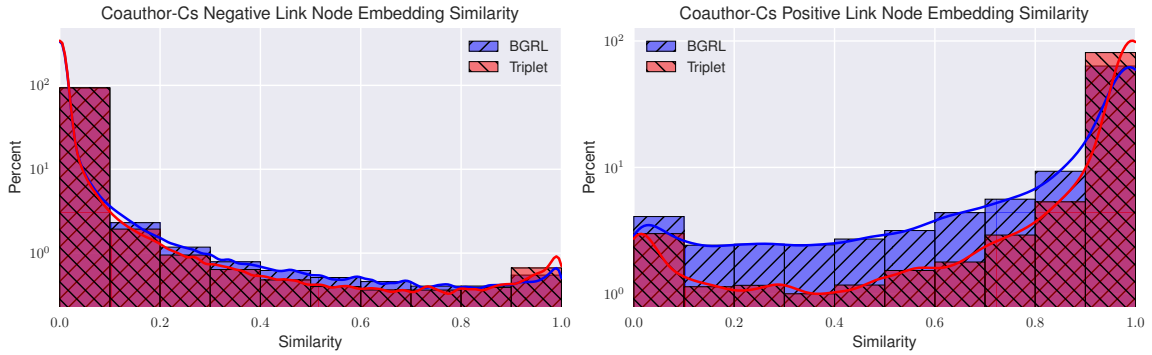


Figure 4.5 These plots show similarities between node embeddings on `Coauthor-Cs`. **Left:** distribution of similarity to *non-neighbors* for TENGAN and BGRL (closer to 0 is better). **Right:** distribution of similarity to *neighbors* for TENGAN and BGRL (closer to 1 is better). Note that the y-axis is on a logarithmic scale. TENGAN clearly does a better job of ensuring that negative link representations are pushed far apart from those of positive links.

The BGRL loss is stated in Equation (4.2) above, but we rewrite it relative to our anchor u and with our assumption about the predictor:

$$\mathcal{L}_u = -\frac{\tilde{\boldsymbol{u}}_1 \cdot \tilde{\boldsymbol{u}}_2}{||\tilde{\boldsymbol{u}}_1||\ ||\tilde{\boldsymbol{u}}_2||} \tag{4.6}$$

Minimizing this loss pushes $\tilde{\boldsymbol{u}}_1$ and $\tilde{\boldsymbol{u}}_2$ closer. Let us denote the encoder after one round of optimization as $\textsc{Enc}'$. Then:

$$\mathbb{E}\left[||\textsc{Enc}'(\textsc{Aug}(u)) - \textsc{Enc}'(\textsc{Aug}(u))||\right] < \mathbb{E}\left[||\textsc{Enc}(\textsc{Aug}(u)) - \textsc{Enc}(\textsc{Aug}(u))||\right] \tag{4.7}$$

Note that $\boldsymbol{v}$ in this example lies within the space of possible augmentations — that is, $v \in \mathcal{A}$, where $\mathcal{A}$ is the set of all possible values of $\textsc{Aug}(u)$. This means, as we repeat this process, we implicitly push $\boldsymbol{u}$ and $\boldsymbol{v}$ closer together — leading to distributions like those shown in Figure 4.1.

4.6.11 Additional Plots

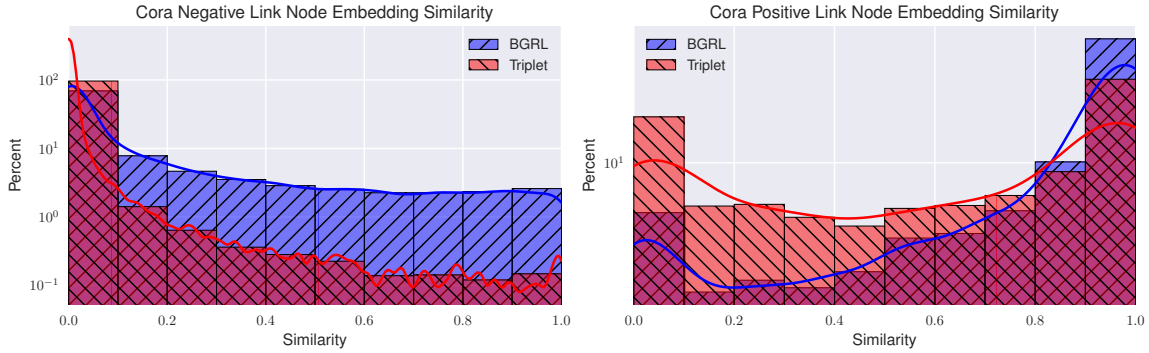


Figure 4.6 These plots show similarities between node embeddings on `Cora`. **Left:** distribution of similarity to *non-neighbors* for TenGAN and BGRL (closer to 0 is better). **Right:** distribution of similarity to *neighbors* for TenGAN and BGRL (closer to 1 is better). Note that the y-axis is on a logarithmic scale. TenGAN clearly does a better job of ensuring that negative link representations are pushed far apart from those of positive links, but does not do as well at differentiating between positive links.

Chapter 5

Clustering-Accelerated Representation Learning on Graphs

Self-supervised learning on graphs has made large strides in achieving great performance in various downstream tasks. However, many state-of-the-art methods suffer from a number of impediments, which prevent them from realizing their full potential. For instance, contrastive methods typically require negative sampling, which is often computationally costly. While non-contrastive methods avoid this expensive step, most existing methods either rely on overly complex architectures or dataset-specific augmentations. In this paper, we ask: *Can we borrow from classical unsupervised machine learning literature in order to overcome those obstacles?* Guided by our key insight that the goal of distance-based clustering closely resembles that of contrastive learning: both attempt to pull representations of similar items together and dissimilar items apart. As a result, we propose CARL-G — a novel *clustering-based* framework for graph representation learning that uses a loss inspired

by Cluster Validation Indices (CVIs), i.e., internal measures of cluster quality (no ground truth required). CARL-G is adaptable to different clustering methods and CVIs, and we show that with the right choice of clustering method and CVI, CARL-G outperforms node classification baselines on 4/5 datasets with up to a **79**× training speedup compared to the best-performing baseline. CARL-G also performs at par or better than baselines in node clustering and similarity search tasks, training up to **1,500**× faster than the best-performing baseline. Finally, we also provide theoretical foundations for the use of CVI-inspired losses in graph representation learning.

5.1 Introduction

Graphs can be used to represent many different types of relational data, including chemistry graphs [29, 71], social networks [135, 166], and traffic networks [42, 181]. Graph Neural Networks (GNNs) have been effective in modeling graphs across a variety of tasks, such as for recommendation systems [210, 81, 159, 182, 53, 230], graph generation [213, 52, 172], and node classification [236, 183, 224, 227, 95, 76]. These GNNs are traditionally [107] trained with a supervised loss, which requires labeled data that is often expensive to obtain in real-world scenarios. Graph self-supervised learning (SSL), a recent area of research, attempts to solve this by learning multi-task representations without labeled data [236, 183, 94, 15, 98].

Most of these existing graph SSL methods can be grouped into either contrastive or non-contrastive SSL. Contrastive learning pulls the representations of similar ("positive") samples together and pushes the representations of dissimilar ("negative") samples apart. In the case of graphs, this often means either pulling the representations of a node and its

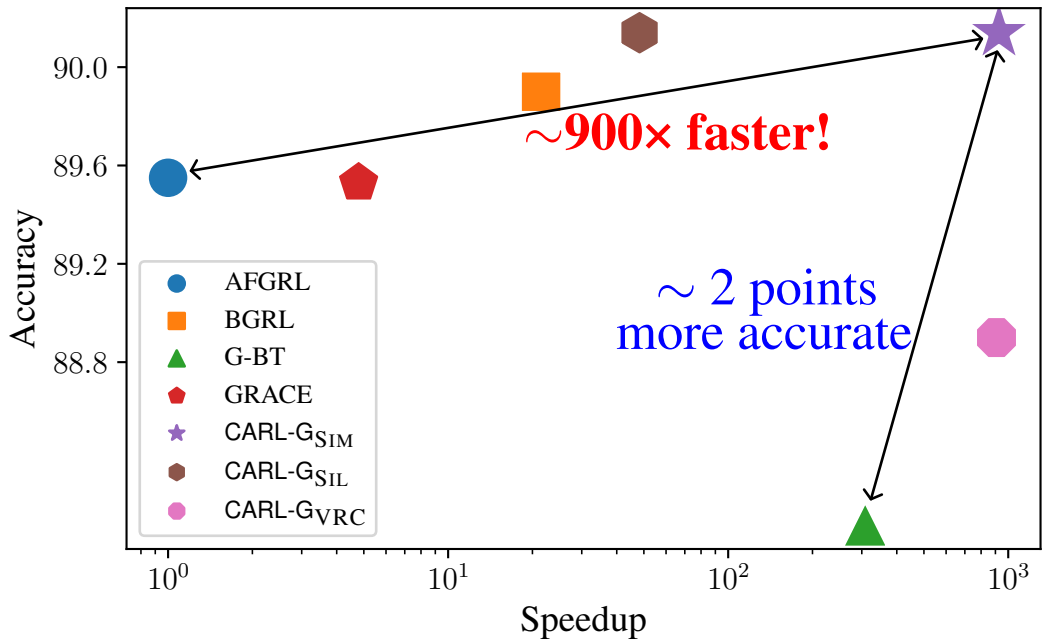


Figure 5.1 Comparison of our proposed methods with other baselines with respect to node classification accuracy and speedup on the `Amazon-Photos` dataset. See Figure 5.3 for results on the other datasets.

neighbors together [75] or pulling the representations of the same node across two different augmentations together [215]. Graph contrastive learning methods typically use non-neighbors as negative samples [236, 210], which can be costly. Non-contrastive learning [174, 183, 15, 113] avoids this step by only pulling positive samples together while employing strategies to avoid collapse.

However, all of these methods have some key limitations. Contrastive methods rely on a negative sampling step, which has an expensive quadratic runtime [183] and requires careful tuning [208]. Non-contrastive methods often have complex architectures (ex. extra encoder with exponentially updated weights [183, 113, 100]) and/or rely heavily on augmentations [15, 222, 183, 229]. Lee et al. [113] shows that augmentations can change the

	Proposed	Baseline Methods				
	CARL-G*	**AFGRL** [113]	**BGRL** [183]	**G-BT** [15]	**GRACE** [236]	**MVGRL** [79]
Avoids Negative Sampling	✓	✓	✓	✓	✗	✗
Augmentation-Free	✓	✓	✗	✗	✗	✗
Single Encoder	✓	✗	✗	✓	✗	✗
Single Forward Pass per Epoch	✓	✗	✗	✗	✗	✗

Table 5.1 Comparison of different self-supervised graph learning methods. *: We use CARL-G_{SIM} as the representative method since it is the best-performing across all of the criteria.

semantics of underlying graphs, especially in the case of molecular graphs (where perturbing a single edge can create an invalid molecule).

Upon further inspection, we observe that the behavior of contrastive and non-contrastive methods is somewhat similar to that of distance-based clustering [203]—both attempt to pull together similar nodes/samples and push apart dissimilar ones. The primary advantage of using clustering over negative sampling is that we can work directly in the smaller embedding space, preventing expensive negative sampling over the graph. Furthermore, there have been decades of research exploring the theoretical foundations of clustering methods and many different metrics have been proposed to evaluate the quality of clusters [158, 24, 103, 47]. These metrics have been dubbed Cluster Validation Indices (CVIs) [6]. In this work, we ask the following question: *Can we leverage well-established clustering methods and CVIs to create a flexible, fast, and effective GNN framework to learn node representations without any labels?*

It is worth emphasizing that our goal is *not node clustering* directly—it is self-supervised graph representation learning. The goal is to develop a general framework that is capable of learning node embeddings for various tasks, including node classification, node

clustering, and node similarity search. There exists some similar work. DeepCluster [27] trains a Convolutional Neural Network (CNN) with a supervised loss on pseudo-labels generated by a clustering method to learn image embeddings.

AFGRL [113] uses clustering to select positive samples in lieu of augmentations for graph representation learning and applies the general BGRL [183] framework to push those representations together. SCD [175] searches over different hyperparameters to obtain a clustering where the silhouette score is maximized. However, to the best of our knowledge, there is no existing work in self-supervised representation learning that directly optimizes for CVIs, which, as we elaborate below, presents us with tremendous potential in advancing and accelerating the state of the art.

We fill this gap by proposing the novel idea of using Cluster Validation Indices (CVIs) directly as our loss function for a neural network. In conjunction with advances in clustering methods [164, 165, 114, 146], CVIs have been improved over the years as measures of cluster quality after performing clustering and have been shown for almost 5 decades to be effective for this purpose [158, 163, 6, 24].

Our proposed method, CARL-G, has several advantages over existing graph SSL methods by virtue of CVI-inspired losses. First, CARL-G generally outperforms other graph SSL methods in node clustering, clustering, and node similarity tasks (see Tables 5.2, 5.5 and 5.6). Second, CARL-G does not require the use of graph augmentations — which are required by many existing graph contrastive and non-contrastive methods [236, 100, 183, 113, 15, 222] and can inadvertently alter graph semantics [113]. Third, CARL-G has a relatively simple architecture compared to the dual encoder architecture of leading non-contrastive

methods [183, 100, 113], drastically reducing the memory cost of our framework. Fourth, we provide theoretical analysis that shows the equivalence of some CVI-based losses and a well-established (albeit expensive) contrastive loss. Finally, CARL-G is sub-quadratic with respect to the size of the graph and much faster than the baselines, with up to a 79× speedup on `Coauthor-CS` over BGRL [183] (the best-performing baseline).

Our contributions can be summarized as follows:

- We propose CARL-G, the first (to the best of our knowledge) framework to use a Cluster Validation Index (CVI) as a neural network loss function.
- We propose 3 variants of CARL-G based on different CVIs, each with its own advantages and drawbacks.
- We extensively evaluate CARL-G_{SIM} — the best all-around performer — across 5 datasets and 3 downstream tasks, where it generally outperforms baselines.
- We provide theoretical insight on CARL-G_{SIM}'s success.
- We benchmark CARL-G_{SIM} against 4 state-of-the-art models and show that it is up to 79 × faster with half the memory consumption (with the same encoder size) compared to the best-performing node classification baseline.

5.2 Preliminaries

Notation We denote a graph as $G = (\mathcal{V}, \mathcal{E})$. $\mathcal{V}$ is the set of n nodes (i.e., $n = |\mathcal{V}|$) and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. We denote the node-wise feature matrix as $\boldsymbol{X} \in \mathbb{R}^{n \times f}$, where f is the dimension of raw node features, and its i-th row $\boldsymbol{x}_i$ is the feature vector for the i-th node. We denote the binary adjacency matrix as $\boldsymbol{A} \in \{0, 1\}^{n \times n}$ and the learned

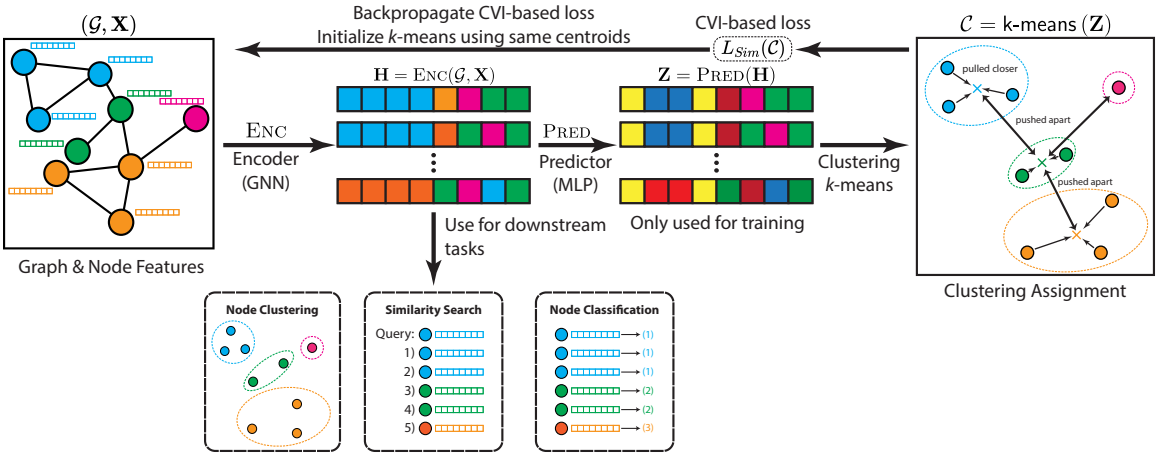


Figure 5.2 CARL-G architecture diagram. We describe the method in detail in Section 5.3.

node representations as $\boldsymbol{H} \in \mathbb{R}^{n \times d}$, where d is the size of latent dimension, and $\boldsymbol{h}_i$ is the representation for the i-th node. Let $\mathcal{N}(u)$ be a function that returns the set of neighbors for a given node u (i.e., $\mathcal{N}(u) = \{v | (u, v) \in \mathcal{E}\}$).

Let $\mathcal{C}$ be the set of clusters, $\mathcal{C}_i \subseteq \mathcal{V}$ be the set of nodes in the i-th cluster, and $c = |\mathcal{C}|$ be the number of clusters. For ease of notation, let $\mathcal{U} \in [1, c]^n$ be the set of cluster assignments, where $\mathcal{U}_u$ is the cluster assignment for node u. Let $\boldsymbol{\mu} = \frac{1}{c} \sum_{u \in \mathcal{V}} \boldsymbol{h}_u$ be the global centroid and $\boldsymbol{\mu}_i = \frac{1}{|\mathcal{C}_i|} \sum_{u \in \mathcal{C}_i} \boldsymbol{h}_u$ be the centroid for the i-th cluster.

5.2.1 Graph Neural Networks

A Graph Neural Network (GNN) [107, 75, 226] typically performs message-passing along the edges of the graph. Each iteration of the GNN can be described as follows [160]:

$$\boldsymbol{h}_u^{(k+1)} = \textsc{Update}^{(k)} \left(\boldsymbol{h}_u^{(k)}, \textsc{Aggregate}^{(k)}(\{\boldsymbol{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\}) \right), \tag{5.1}$$

where Update and Aggregate are differentiable functions, and $\boldsymbol{h}_u^{(0)} = \boldsymbol{x}_u$. In this work, we opt for simplicity and use Graph Convolutional Networks (GCNs) [107] as the default GNN. These are GNNs where Update consists of a single MLP layer, and Aggregate is

the mean of a node's representation with its neighbors. Formally, each iteration of the GCN can be written as:

$$\boldsymbol{h}_u^{(k+1)} = \sigma\left(\boldsymbol{W}^{(k+1)} \sum_{v \in \mathcal{N}(u) \cup \{u\}} \frac{\boldsymbol{h}_v}{\sqrt{|\mathcal{N}(u)| \cdot |\mathcal{N}(v)|}}\right). \tag{5.2}$$

5.2.2 Cluster Validation Indices

Clustering is a class of unsupervised methods that aims to partition the input space into multiple groups, known as clusters. The goal of clustering is generally to maximize the similarity of points within each cluster while minimizing the similarity of points between clusters [203]. In this work, we focus on centroid-based clustering algorithms like k-means [129] and k-medoids [141].

Cluster Validation Indices (CVIs) [6] estimate the quality of a partition (i.e., clustering) by measuring the compactness and separation of the clusters without knowledge of the ground truth clustering. Note that these are different from metrics like Normalized Mutual Information (NMI) [38] or the Rand Index [153], which require ground truth information of cluster labels. Many different CVIs have been proposed over the years and extensively evaluated [6, 163].

Arbelaitz et al. [6] extensively evaluated 30 different CVIs over a wide variety of datasets and found that the Silhouette [158], Davies-Bouldin* [103], and Calinski-Harabasz (also known as the VRC: Variance Ratio Criterion) [24] indices perform best across 720 different synthetic datasets. The VRC has also been shown to be effective in determining the number of clusters for clustering methods [163, 24, 132, 50]. As such, we focus on the

silhouette index (the best-performing CVI) and VRC (an effective CVI — especially for choosing the number of clusters) in this work.

Silhouette

The silhouette index computes the ratio of intra-cluster distance with respect to the inter-cluster distance of itself with its nearest neighboring cluster. It returns a value in [-1, 1], where a value closer to 1 signifies more desired and better distinguishable clustering. The silhouette index [158] is defined as $\textsc{Sil}(\mathcal{C}) = \frac{1}{n}\sum_{u\in\mathcal{V}} s(u)$, where:

$$s(u) = \frac{b(u) - a(u)}{\max\{a(u), b(u)\}}\,, \tag{5.3}$$

and

$$a(u) = \frac{1}{|\mathcal{C}_{\mathcal{U}_u}| - 1} \sum_{v\in(\mathcal{C}_{\mathcal{U}_u}-\{u\})} \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v)\,, \tag{5.4}$$

$$b(u) = \min_{i\neq\mathcal{U}_u} \frac{1}{|\mathcal{C}_i|} \sum_{v\in\mathcal{C}_i} \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v)\,. \tag{5.5}$$

The runtime of computing the silhouette index for a given node is $O(n)$, which can be expensive if calculated over all nodes. We discuss this issue and a modified solution later in Section 5.3.1.

Variance Ratio Criterion

The VRC [24] computes a ratio between its intra-cluster variance and its inter-cluster variance. Its intra-cluster variance is based on the distances of each point to its

centroid, and its inter-cluster variance is based on the distance from each cluster centroid to the global centroid. Formally,

$$\mathrm{VRC}(\mathcal{C}) = \frac{n-c}{c-1} \frac{\sum_{\mathcal{C}_k \in \mathcal{C}} |\mathcal{C}_k| \textsc{Dist}(\boldsymbol{\mu}_k, \boldsymbol{\mu})}{\sum_{\mathcal{C}_k \in \mathcal{C}} \sum_{u \in \mathcal{C}_k} \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{\mu})} . \tag{5.6}$$

For the purposes of this paper, we use Euclidean distance, i.e., $\textsc{Dist}(\boldsymbol{a}, \boldsymbol{b}) = \|\boldsymbol{a} - \boldsymbol{b}\|_2$.

5.3 Proposed Method

Problem Formulation

Given a graph G and its node-wise feature matrix $\boldsymbol{X} \in \mathbb{R}^{n \times f}$, learn node embeddings $\boldsymbol{h}_u \in \mathbb{R}^d$ for each node $u \in \mathcal{V}$ without any additional information (e.g. node class labels). The learned embeddings should be suitable for various downstream tasks, such as node classification and node clustering.

CARL-G

We propose CARL-G, which consists of three main steps (Figure 5.2). First, a GNN encoder $\textsc{Enc}(\cdot)$ takes the graph as input and produce node embeddings $\boldsymbol{H} = \textsc{Enc}(\boldsymbol{X}, \boldsymbol{A})$. Next, a multi-layer perceptron (MLP) predictor network $\textsc{Pred}(\cdot)$ takes the embeddings by GNN and produces a second set of node embeddings $\boldsymbol{Z} = \textsc{Pred}(\boldsymbol{H})$. We then perform a clustering algorithm (e.g., k-means) on $\boldsymbol{Z}$ to produce a set of clusterings $\mathcal{C}$. It is worth noting that the clustering algorithm does not have to be differentiable. Finally, we compute a cluster validation index (CVI) on the cluster assignments and backpropagate to update the encoder's and predictor's parameters. After training, only the GNN encoder $\textsc{Enc}(\cdot)$ and its produced embeddings $\boldsymbol{H}$ are used to perform downstream tasks, and the predictor

network $\textsc{Pred}(\cdot)$ is discarded (similar to the prediction heads in non-contrastive learning work [183, 113, 100]).

5.3.1 Training CARL-G

As aforementioned in Section 5.2.2, we evaluate the silhouette index [158] and the VRC [24] as learning objectives. In order to use them effectively, we must make slight modifications to the loss functions. First, we must negate the functions since a higher score is better for both CVIs, and we typically want to minimize a loss function. Second, while $\textsc{Sil}(\mathcal{C}) = 1$ and $\text{VRC}(\mathcal{C}) \to \infty$ are theoretically ideal, we find this is generally not true in practice. This is because the clustering method may miscluster some nodes and fully maximizing the CVIs will push the misclustered representations too close together, negatively impacting a classifier's ability to distinguish them. To bound the maximum values of our loss, we add τ_{SIL} and τ_{VRC} — the target silhouette and VRC indices, respectively. The silhouette-based and VRC-based losses are then defined as follows:

$$L_{\text{SIL}} = |\tau_{\text{SIL}} - \textsc{Sil}(\mathcal{C})| \ , \qquad L_{\text{VRC}} = |\tau_{\text{VRC}} - \text{VRC}(\mathcal{C})| \ , \tag{5.7}$$

where $\tau_{\text{SIL}} \in [-1, 1]$ is the target silhouette index and $\tau_{\text{VRC}} \in [0, \infty)$ is the target VRC.

Upon careful inspection of Equation (5.3), we can observe that the computational complexity for the silhouette is $O(n^2)$, while the complexity of VRC is only $O(nc)$, where $c \ll n$. This is a critical weakness in using the silhouette, especially when the goal is to avoid a quadratic runtime (the typical drawback of contrastive methods). Backpropagating on this loss function would also result in quadratic memory usage because we have to store the gradients for each operation. To resolve this issue, we leverage the simplified silhouette [86],

which instead uses the centroid distance. The simplified silhouette has been shown to have competitive performance with the original silhouette [189] while being much faster — running in $O(nc)$ time. As such, we also try the simplified silhouette, which can be written as:

$$s'(u) = \frac{b'(u) - a'(u)}{\max\{a'(u), b'(u)\}}\,, \tag{5.8}$$

where $i = \mathcal{U}_u$ is the cluster assignment for u and

$$a'(u) = \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{\mu}_i)\,, \qquad b'(u) = \min_{\mathcal{C}_k \neq \mathcal{C}_i} \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{\mu}_i)\,. \tag{5.9}$$

We use the same loss function as L_{SIL} (Equation (5.7)), simply substituting $s'(u)$ for $s(u)$ (see Section 5.2.2), and name it L_{SIM}.

5.3.2 Clustering Method

k-means We primarily focus on k-means clustering for this framework due to its fast linear runtime (although we do briefly explore using k-medoids in Section 5.4.3 below). The goal of k-means is to minimize the sum of squared errors—also known as the inertia or within-cluster sum of squares. Formally, this can be written as:

$$\arg\min_{\mathcal{C}} \sum_{i=1}^{c} \sum_{x \in \mathcal{C}_i} \textsc{Dist}(\boldsymbol{x}, \boldsymbol{\mu}_i)\,. \tag{5.10}$$

Finding the optimal solution to this problem is NP-hard [39], but efficient approximation algorithms [165, 126] have been developed that return an approximate solution in linear time (see Section 5.5). While k-means is fast, it is known to be heavily dependent on its initial centroid locations [21, 8], which can be partially solved via repeated re-initialization and picking the clustering that minimizes the inertia.

Poor initialization is typically not a large issue in k-means use cases since the end goal is usually to compute a single clustering so we can simply repeat and re-initialize until we are satisfied. However, since we generate a new clustering once per epoch in CARL-G, poor initialization can result in a large amount of variance between epochs.

To minimize the chance of poor centroid initialization occurring during training, we carry the cluster centroids over between epochs. The centroids will naturally update after running k-means since the embeddings $\boldsymbol{Z}$ changes each epoch (after backpropagation with CVI-based loss).

5.3.3 Theoretical Analysis

To gain a theoretical understanding of why our framework works, we compare it to Margin loss — a fundamental contrastive loss function that has been shown to work well for self-supervised representation learning [75, 210]. We show that CVI-based loss (especially silhouette loss) has some similarity to Margin based loss, which intuitively explains the success of CVI-based loss. In addition, we show that CVI-based loss has the advantages of (a) lower sensitivity to graph structure, and (b) no negative sampling required.

Similarity analysis of CVI-based loss and Margin loss

Both Margin loss and CVI-based loss fundamentally consist of two terms: one measuring the distance between neighbors/inter-cluster points and one measuring the distance between non-neighbors/inter-cluster points. This similarity allows us to analyze basic versions of our proposed silhouette loss and margin loss in the context of node classification and show that they are identical in both of their ideal cases. We further analyze the sensitivity of

these losses with respect to various parameters of the graph to examine the advantages and disadvantages of our proposed method. To do this, we first define the mean silhouette and margin loss functions:

Definition 1 (Mean Silhouette) *We define the mean silhouette loss (removing the hyper-parameter τ_{SIL}, focusing on the numerator (un-normalizing the index) and replacing* min *with the mean) as follows:*

$$L_{MS}(u) = -(b_{MS}(u) - a(u)) = a(u) - b_{MS}(u)\,, \tag{5.11}$$

where

$$b_{MS}(u) = \frac{1}{c-1}\sum_{j\neq i}\frac{1}{|\mathcal{C}_j|}\sum_{v\in\mathcal{C}_j} \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v)\,. \tag{5.12}$$

Definition 2 (Margin Loss) *We define margin loss as follows:*

$$\begin{aligned} ML(u) =& \frac{1}{|\mathcal{N}(u)|}\sum_{v\in\mathcal{N}(u)} \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) \\ &- \frac{1}{|\mathcal{V} - \mathcal{N}(u) - \{u\}|}\sum_{t\notin\mathcal{N}(u)} \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_t)\,. \end{aligned} \tag{5.13}$$

It is worth noting that this margin loss differs from the max-margin loss traditionally used in graph SSL [210]. We simplify it above in Theorem 2 by removing the max function for ease of analysis.

Let $\mathcal{L}$ be the set of true class labels, and $\mathcal{L}_u$ be the class label for a node $u \in \mathcal{V}$. We define the expected inter-class and intra-class distances as follows:

$$\mathbb{E}\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v)\right] = \begin{cases} \alpha, & \text{if } \mathcal{L}_u = \mathcal{L}_v\,; \\ \beta, & \text{otherwise,} \end{cases} \tag{5.14}$$

where $\alpha, \beta \in \mathbb{R}^+$. Next, let

$$P\left((u,v) \in \mathcal{E}\right) = \begin{cases} p, & \text{if } \mathcal{L}_u = \mathcal{L}_v\,; \\ q, & \text{otherwise,} \end{cases} \tag{5.15}$$

i.e., $\mathcal{G}$ follows a stochastic block model with a probability matrix $P \in [0,1]^{c \times c}$ of the form:

$$P = \begin{bmatrix} p & q & q & q \\ q & \ddots & \ddots & \ddots \\ q & \ddots & \ddots & q \\ q & \ddots & q & p \end{bmatrix}. \tag{5.16}$$

Note that q does not necessarily equal $1 - p$. Finally, we define the inter-class clustering error rate ϵ and intra-class clustering error rate δ as follows:

$$P(\mathcal{C}_u \neq \mathcal{C}_v | \mathcal{L}_u = \mathcal{L}_v) = \epsilon\,; \tag{5.17}$$

$$P(\mathcal{C}_u = \mathcal{C}_v | \mathcal{L}_u \neq \mathcal{L}_v) = \delta\,. \tag{5.18}$$

Claim 3 *Given the above assumptions, the expected value of the simplified silhouette loss approaches that of the margin loss as $p \to 1, q \to 0$, and $\epsilon, \delta \to 0$.*

Proof. To find $\mathbb{E}\left[L_s(u)\right]$, we first find $\mathbb{E}\left[a(u)\right]$ and $\mathbb{E}\left[b_s(u)\right]$:

$$\mathbb{E}\left[a(u)\right] = \frac{\alpha}{c} - \frac{\epsilon\alpha}{c} + \delta\beta - \frac{\delta\beta}{c}\,; \tag{5.19}$$

$$\mathbb{E}\left[b_s(u)\right] = \frac{\epsilon\alpha}{c} + \beta - \beta\delta - \frac{\beta}{c} + \frac{\delta\beta}{c}\,; \tag{5.20}$$

$$\begin{aligned} \mathbb{E}\left[L_s(u)\right] &= \mathbb{E}\left[a(u)\right] - \mathbb{E}\left[b_s(u)\right] \\ &= -\frac{2\epsilon\alpha}{c} - \frac{2\delta\beta}{c} + \frac{\beta}{c} + \frac{\alpha}{c} - \beta + 2\delta\beta\,. \end{aligned} \tag{5.21}$$

Next, we take its limit as $\epsilon, \delta \to 0$:

$$\lim_{\epsilon,\delta\to 0}\left(-\frac{2\epsilon\alpha}{c}-\frac{2\delta\beta}{c}+\frac{\beta}{c}+\frac{\alpha}{c}-\beta+2\delta\beta\right)=\frac{\beta}{c}+\frac{\alpha}{c}-\beta\,. \tag{5.22}$$

To find $\mathbb{E}\left[\textsc{ml}(u)\right]$, we first find the expected value of its left and right sides:

$$\mathbb{E}\left[\frac{1}{\mathcal{N}(u)}\sum_{v\in\mathcal{N}(u)}\textsc{Dist}(\boldsymbol{h}_u,\boldsymbol{h}_v)\right]=\frac{\alpha p}{c}+\beta q-\frac{\beta q}{c}\,; \tag{5.23}$$

$$\mathbb{E}\left[\frac{1}{|\mathcal{V}-\mathcal{N}(u)-\{u\}|}\sum_{t\notin\mathcal{N}(u)}\textsc{Dist}(\boldsymbol{h}_u,\boldsymbol{h}_t)\right] \tag{5.24}$$

$$=\frac{\alpha}{c}-\frac{\alpha p}{c}+\beta-\beta q-\frac{\beta}{c}+\frac{\beta q}{c}\,. \tag{5.25}$$

Substituting them back in, we get

$$\mathbb{E}\left[\textsc{ml}(u)\right]=\frac{2\alpha p}{c}+2\beta q-\frac{2\beta q}{c}-\frac{\alpha}{c}+\frac{\beta}{c}-\beta\,. \tag{5.26}$$

Taking its limit as $p \to 1, q \to 0$, we find

$$\lim_{p\to 1,q\to 0}\left(\frac{2\alpha p}{c}+2\beta q-\frac{2\beta q}{c}-\frac{\alpha}{c}+\frac{\beta}{c}-\beta\right) \tag{5.27}$$

$$=\frac{2\alpha}{c}-\frac{\alpha}{c}+\frac{\beta}{c}-\beta=\frac{\alpha}{c}+\frac{\beta}{c}-\beta\,. \tag{5.28}$$

$$\therefore \lim_{p\to 1,q\to 0}\mathbb{E}\left[\textsc{ml}(u)\right]=\lim_{\epsilon,\delta\to 0}\mathbb{E}\left[L_s(u)\right]\,. \tag{5.29}$$

■ Since the two loss functions are identical in their ideal cases, one may wonder: *Why not use margin loss instead?* Well, the silhouette-based loss has two key advantages:

Lower sensitivity to graph structure.

The margin loss is minimized as $p \to 1$ and $q \to 0$. However, p and q are attributes of the graph itself, making it difficult for a user to directly improve the performance of a model using that loss function. On the other hand, the mean silhouette depends on ϵ and δ, the

inter/intra-class clustering error rates, instead. Even on the same graph, a silhouette-based loss can likely be improved by either choosing a more suitable clustering method or distance metric. This greatly increases the flexibility of this loss function.

No negative sampling.

Negative sampling is required for most graph contrastive methods and often requires either many samples [79] or carefully chosen samples [212, 207]. This is costly, often costing quadratic time [183]. The primary advantage of non-contrastive methods is that they avoid this step [15, 183]. The simplified silhouette avoids this issue by only working in the $n \times d$ embedding space instead of the $n \times n$ graph. It also contrasts node representations against centroid representations instead of against other nodes directly.

5.4 Experimental Evaluation

We evaluate 3 variants of CARL-G: (a) **CARL-G$_{\mathbf{SIL}}$** — based on the silhouette loss in Equation (5.7), (b) **CARL-G$_{\mathbf{VRC}}$** — based on the VRC loss in Equation (5.7), and (c) **CARL-G$_{\mathbf{SIM}}$** — based on the simplified silhouette loss in Equation (5.8). We evaluate these variants on 5 datasets on node classification and thoroughly benchmark their memory usage and runtime. We then select the best-performing variant, CARL-G$_{\text{SIM}}$, and evaluate its performance across 2 additional tasks: node clustering, and embedding similarity search.

Node Classification

A common task for GNNs is to classify each node into one of several different classes. In the supervised setting, this is often explicitly optimized for during the training process

since the GNN is typically trained with cross-entropy loss over the labels [107, 75] but this is not possible for graph SSL methods where we do not have the labels during the training of the GNN. As such, the convention [186, 236, 15, 183, 113] is to train a logistic regression classifier on the frozen embeddings produced by the GNN.

Following previous works, we train a logistic regression model with ℓ_2 regularization on the frozen embeddings produced by our encoder model. We compare against a variety of self-supervised baselines, including both GNN-based and non-GNN-based: DeepWalk [149], RandomInit [186], DGI [186], GMI [147], MVGRL [79], GRACE [236], G-BT [15], AFGRL [113], and BGRL [183]. We also evaluate our method against two supervised models: GCA [237] and a GCN [107]. We follow [183, 113] and use an 80/10/10 train/validation/test, early stopping on the validation accuracy. We re-run AFGRL and BGRL using their published code and weights (where possible) on that split[1]. Finally, we use node2vec results from [113] and the reported results of the other baseline methods from their respective papers. See Section 5.4.4 for implementation details.

Node Clustering

Following previous graph representation learning work [79, 142, 113], we also evaluate CARL-G on the task of node clustering. We fit a k-means model on the generated embeddings $\boldsymbol{H}$ using the evaluation criteria from [113] — NMI and cluster homogeneity. Following [113], we re-run our model with different hyperparameters (the embeddings are not the same as node classification) and report the highest clustering scores. Due to computational resource

[1]The official BGRL implementation online uses an 80/0/20 split compared to the 80/10/10 split mentioned in [183], so we re-run their trained models on an 80/10/10 split. Most results are similar but we get slightly different results on `Amazon-Computers`.

constraints, we choose to only evaluate CARL-G_{SIM}, the overall best-performing model. We report the scores of the baselines models from [113].

Similarity Search

Following [113], we evaluate our model on node similarity search. The goal of similarity search is to, given a query node u, return the k nearest neighbors. In our setting, the goal is to return other nodes belonging to the same class as the query node. We evaluate the performance of each method by computing its Hits@k — the percentage of the top k neighbors that belong to the same class. Similar to [113], we evaluate our model every epoch and report the highest similarity search scores. We use $k \in \{5, 10\}$ and use the scores from [113] as baseline results.

5.4.1 Evaluation Results

Node Classification Performance

We show the node classification accuracy of our three proposed methods along with the baseline results in Table 5.5. CARL-G_{SIL} generally performs the best of all the evaluated methods, with the highest accuracy on 4/5 of the datasets (all except `Wiki-CS`). CARL-G_{SIM} generally performs similarly to CARL-G_{SIL}, with similar performance on all datasets except `Wiki-CS` and `Amazon-Photos`, and still outperforms the baselines on 4/5 datasets. CARL-G_{VRC} is the weakest-performing method of the three methods. It only outperforms baselines on 2/5 datasets. Since CARL-G_{SIM} is much faster than CARL-G_{SIL} (see Section 5.3.1 and Figure 5.4a) without sacrificing much performance, we focus on CARL-G_{SIM} for the remainder of the evaluation tasks.

		GRACE	GCA	BGRL	AFGRL	CARL-G_{SIM}
WikiCS	NMI	0.428	0.337	0.397	0.413	**0.471**
	Hom.	0.442	0.353	0.416	0.430	**0.491**
Computers	NMI	0.479	0.528	0.536	0.552	**0.558**
	Hom.	0.522	0.582	0.587	0.604	**0.607**
Photo	NMI	0.651	0.644	0.684	0.656	**0.701**
	Hom.	0.666	0.658	0.700	0.674	**0.718**
Co.CS	NMI	0.756	0.762	0.773	0.786	**0.790**
	Hom.	0.791	0.797	0.804	**0.816**	0.815
Co.Physics	NMI	OOM	OOM	0.557	0.729	**0.771**
	Hom.	OOM	OOM	0.602	0.735	**0.776**

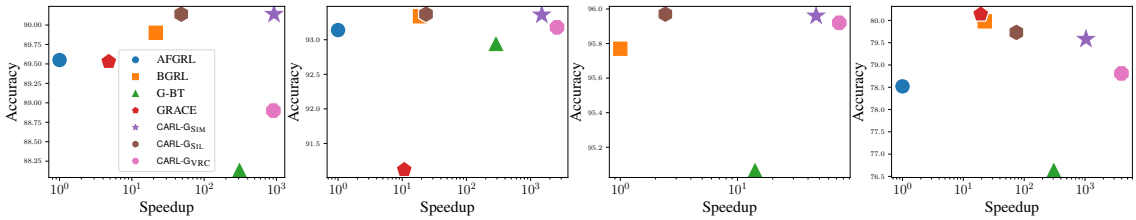
Table 5.2 Node clustering performance in terms of cluster NMI and homogeneity. CARL-G_{SIM} outperforms the baselines on 4/5 datasets.

Node Clustering Performance

We evaluate CARL-G_{SIM} on node clustering and display the results in Table 5.2. We find that it generally outperforms its baselines on 5/5 datasets in terms of NMI and 4/5 datasets in terms of homogeneity. CARL-G_{SIM} and AFGRL [113] both encourage a clusterable representations by utilizing k-means clustering as part of their respective training pipelines.

Similarity Search Performance

We evaluate CARL-G_{SIM} on similarity search in Table 5.6, where it roughly performs on par with AFGRL, the best-performing baseline. This is surprising, as AFGRL specifically



(a) `Amazon-Comp` (b) `Coauthor-CS` (c) `Coauthor-Physics` (d) `Wiki-CS`

Figure 5.3 Runtime v.s. accuracy plots. CARL-G$_{SIM}$, CARL-G$_{SIL}$, and CARL-G$_{VRC}$ are our proposed methods. Speedup is relative to the slowest baseline (AFGRL). AFGRL and GRACE run out of memory on `Coauthor-Physics`.

optimizes for the similarity search task by using k-NN as one of the criteria to sample neighbors.

5.4.2 Resource Benchmarking

We benchmark the 3 variants of our proposed method against BGRL [183] (the best-performing baseline), AFGRL [113] (the most recent baseline), G-BT [15] (the fastest baseline), and GRACE [236] (a strong contrastive baseline). We time the amount of time it takes to train each of the best-performing node classification models. We remove all evaluation code and purely measure the amount of time it takes to train each method, taking care to synchronize all asynchronous GPU operations. We use the default values in the respective papers for AFGRL and BGRL: 5,000 epochs for AFGRL and 10,000 epochs for BGRL. We use 50 epochs for CARL-G, although our method converges much faster in practice.

We also measure the GPU memory usage of each method. We use the hyperparameters by the respective paper authors for each dataset, which is why the methods use different encoder sizes. Note that the encoder sizes greatly affect the runtime and memory usage of

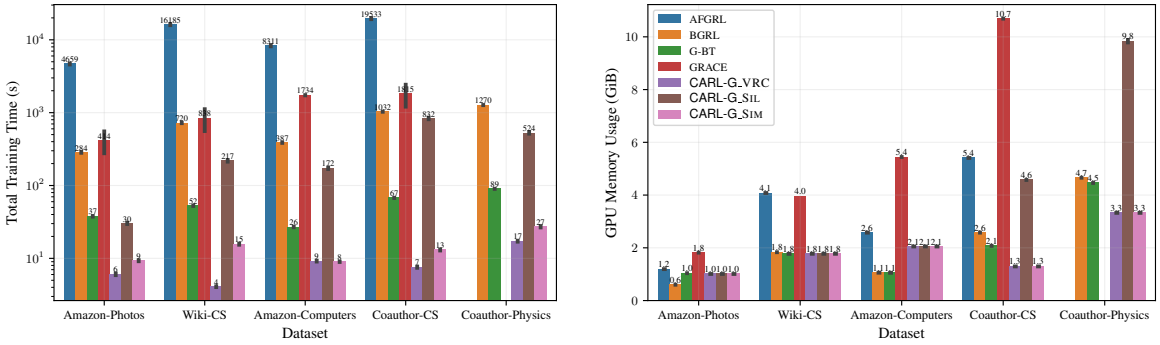
each model, so we report the layer sizes used in Table 5.3. Our benchmarking results can be found in Figures 5.4a and 5.4b.

	Computers	Photos	Co-CS	Co-Phy	Wiki
AFGRL	[512]	[512]	[1024]	OOM	[1024]
BGRL	[256,128]	[256,128]	[512,256]	[256,128]	[512,256]
G-BT	[256,128]	[512,256]	[512,256]	[256,128]	[512,256]
GRACE	[256,128]	[256,128]	[512,256]	[256,128]	[512,256]
CARL-G_{SIM}	[512,256]	[512,256]	[512,256]	[512,256]	[512,256]
CARL-G_{SIL}	[512,256]	[512,256]	[512,256]	[512,256]	[512,256]
CARL-G_{VRC}	[512,256]	[512,256]	[512,256]	[512,256]	[512,256]

Table 5.3 GCN layer sizes used by the encoder for each method. The layer sizes greatly affect the amount of memory used by each model (shown in Figure 5.4b).

CARL-G is fast.

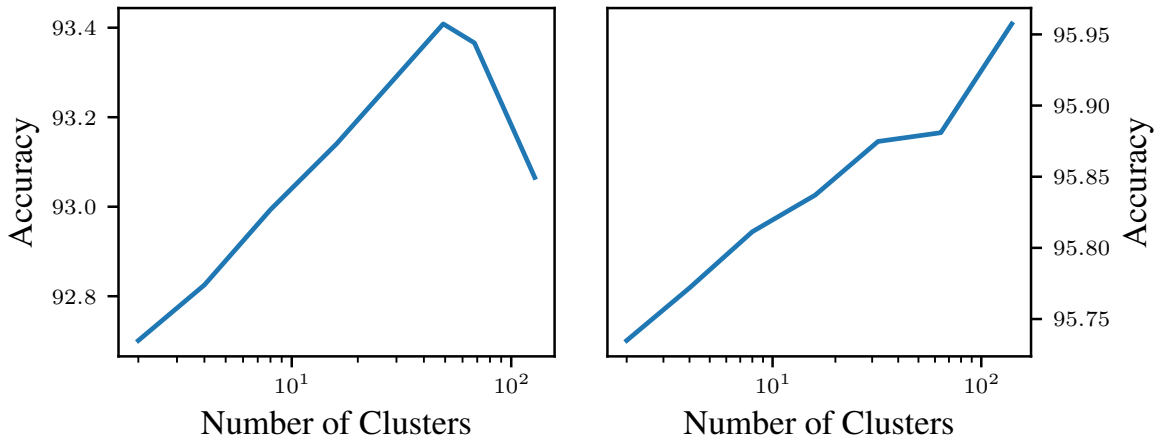
In Figure 5.4a, we show that CARL-G_{SIM} is much faster than competing baselines, even in cases where the encoder is larger (see Table 5.3). BGRL is the best-performing node classification baseline, and CARL-G_{SIM} is about 79× faster on `Coauthor-CS`, and 57× faster on `Coauthor-Physics`. AFGRL is by far the slowest method, requiring much longer to train.



(a) Mean total training time. The y-axis is on a log. scale.

(b) Max GPU memory usage.

Figure 5.4 Mean total training time (left) and max GPU usage (right) for each model. CARL-G_{VRC} is the fastest with generally the least amount of memory used. CARL-G_{SIM} uses the same amount of memory but is slightly slower. Note that not all of the baselines use the same encoder size—see Table 5.3 for encoder sizes.



(a) Acc. on `Amazon-Photos`.

(b) Acc. on `Coauthor-Physics`.

Figure 5.5 Node classification accuracy of CARL-G_{SIM} on `Amazon-Photos` and `Coauthor-Physics` with a different number of clusters.

CARL-G works with a fixed encoder size

We find that CARL-G works well with a fixed encoder size (see Table 5.3). Unlike AFGRL, BGRL, GRACE, and G-BT, we fix the encoder size for CARL-G across all datasets. This has practical advantages by allowing a user to fix the model size across datasets, thereby reducing the number of hyperparameters in the model. We observed that increasing the embedding size also increases the performance of our model across all datasets. This is not true for all of our baselines — for example, [113] found that BGRL, GRACE, and GCA performance will often decrease in performance as embedding sizes increase. We limited our model embedding size to 256 for a fair comparison with other models.

CARL-$\mathsf{G}_{\mathrm{SIM}}$ uses much less memory for the same encoder size.

When the encoder sizes of baseline methods are the same, CARL-$\mathsf{G}_{\mathrm{SIM}}$ uses much less memory than the baselines. The GPU memory usage of CARL-$\mathsf{G}_{\mathrm{SIM}}$ is also much lower than (about half) the memory usage of a BGRL model of the same size. This is because BGRL stores two copies of the encoder with different weights. The second encoder's weights gradually approach that of the first encoder during training but still takes up twice the space compared to single-encoder models like CARL-$\mathsf{G}_{\mathrm{SIM}}$ or G-BT [15].

CARL-$\mathsf{G}_{\mathrm{SIM}}$'s runtime is linear with respect to the number of neighbors.

In Section 5.3.1, we mention that the runtime of CARL-$\mathsf{G}_{\mathrm{SIL}}$, the silhouette-based loss, is $O(n^2)$. This was the motivation for us to propose CARL-$\mathsf{G}_{\mathrm{SIM}}$ — the simplified-silhouette-based loss which has an $O(nc)$ runtime instead.

5.4.3 Ablation Studies

We perform an ablation and sensitivity analysis on several aspects of our model. First, we examine the sensitivity of our model with respect to c—the number of clusters. Second, we examine the effect of using k-medoids instead of k-means. Finally, we try to inject more graph structural information during the clustering stage to see if we are losing any information.

Effect of the number of clusters.

We perform sensitivity analysis on c — the number of clusters (see Figures 5.5a and 5.5b). We find that, generally, the accuracy of our method goes up as the number of clusters increases. As the number of clusters continues to increase, the accuracy begins to drop. This implies that, much like traditional clustering [163], there is some "sweet spot" for c. However, it is worth noting that this number does not directly correspond to the number of classes in the data and is much higher than c for all of the datasets. DeepCluster [27] also makes similar observations, where they find 10,000 clusters is ideal for ImageNet, despite there only being 1,000 labeled classes.

Table 5.4 k-medoids w/ CARL-G_{SIM}.

Dataset	**Accuracy Δ**
Computers	-1.41
Co.CS	-0.33
Co.Phy	-0.07
Photos	-0.11

k-medoids v.s. k-means.

We study the effect of using k-medoids instead of k-means as our clustering algorithm. Both algorithms are partition-based clustering methods [203] and have seen optimizations in recent years [164, 165]. We find that the k-means-based CARL-G_{SIM} generally performs better across all 4 of the evaluated datasets. The differences in node classification accuracy are shown in Table 5.4.

Does additional information help?

It may appear as if we are losing graph information by working only with the embeddings. If this is the case, we should be able to improve the performance of our method by injecting additional information into the clustering step. We can do this by modifying the distance function of our clustering algorithm to the following:

$$\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) = \lambda \left\| \boldsymbol{h}_u - \boldsymbol{h}_v \right\|_2 + (1 - \lambda)\boldsymbol{D}_{u,v}\,, \tag{5.30}$$

where $\boldsymbol{D}$ is the all-pairs shortest path (APSP) length matrix of $\mathcal{G}$. This allows us to inject node neighborhood information into the clustering algorithm on top of the aggregation performed by the GNN. However, we find there is no significant change in performance for low λ and performance decreases for high λ. This helps confirm the hypothesis that the GNN encoder is able to successfully embed a sufficient amount of structural data in the embedding.

	Method	Wiki-CS	Amazon-Computers	Amazon-Photos	Coauthor-CS	Coauthor-Physics
Traditional	Raw Features	71.98 ± 0.00	73.81 ± 0.00	78.53 ± 0.00	90.37 ± 0.00	93.58 ± 0.00
	node2vec [67]	71.79 ± 0.05	84.39 ± 0.08	89.67 ± 0.12	85.08 ± 0.03	91.19 ± 0.04
	DeepWalk [149]	74.35 ± 0.06	85.68 ± 0.06	89.44 ± 0.11	84.61 ± 0.22	91.77 ± 0.15
	DeepWalk [149] + Feat.	77.21 ± 0.03	86.28 ± 0.07	90.05 ± 0.08	87.70 ± 0.04	94.90 ± 0.09
GNN SSL	Random-Init [186]	78.95 ± 0.58	86.46 ± 0.38	92.08 ± 0.48	91.64 ± 0.29	93.71 ± 0.29
	DGI [186]	75.35 ± 0.14	83.95 ± 0.47	91.61 ± 0.22	92.15 ± 0.63	94.51 ± 0.09
	GMI [147]	74.85 ± 0.08	82.21 ± 0.31	90.68 ± 0.17	OOM	OOM
	MVGRL [79]	77.52 ± 0.08	87.52 ± 0.11	91.74 ± 0.07	92.11 ± 0.12	95.33 ± 0.03
	GRACE [236]	**80.14** ± 0.48	89.53 ± 0.35	92.78 ± 0.45	91.12 ± 0.20	OOM
	G-BT [15]	76.65 ± 0.62	88.14 ± 0.33	92.63 ± 0.44	92.95 ± 0.17	95.07 ± 0.17
	AFGRL [113]	78.52 ± 0.72	89.55 ± 0.36	92.91 ± 0.26	93.14 ± 0.23	OOM
	BGRL [183]	<u>79.98</u> ± 0.10	<u>89.90</u> ± 0.19	93.17 ± 0.30	93.34 ± 0.13	95.77 ± 0.05
Proposed	CARL-G_{VRC}	78.81 ± 0.49	88.90 ± 0.39	93.31 ± 0.36	93.18 ± 0.31	95.92 ± 0.14
	CARL-G_{SIM}	79.58 ± 0.60	**90.14** ± 0.33	<u>93.37</u> ± 0.37	<u>93.36</u> ± 0.39	<u>95.96</u> ± 0.09
	CARL-$G_{S_{IL}}$	79.73 ± 0.44	**90.14** ± 0.34	**93.44** ± 0.32	**93.37** ± 0.33	**95.97** ± 0.14
Supervised	GCA [237]	78.35 ± 0.05	88.94 ± 0.15	92.53 ± 0.16	93.10 ± 0.01	95.73 ± 0.03
	Supervised GCN [107]	77.19 ± 0.12	86.51 ± 0.54	92.42 ± 0.22	93.03 ± 0.31	95.65 ± 0.16

Table 5.5 Table of node classification accuracy. Bolded entries indicate the highest accuracy for that dataset. Underlined entries indicate the second-highest accuracy. OOM indicates out-of-memory.

		GRACE	GCA	BGRL	AFGRL	CARL-G$_{\text{SIM}}$
WikiCS	Hits@5	0.775	0.779	0.774	0.781	**0.789**
	Hits@10	0.765	0.767	0.762	0.766	**0.775**
Computers	Hits@5	0.874	0.883	0.895	**0.897**	0.881
	Hits@10	0.864	0.874	0.886	**0.889**	0.871
Photo	Hits@5	0.916	0.911	**0.925**	0.924	0.922
	Hits@10	0.911	0.905	**0.920**	0.917	0.917
Co.CS	Hits@5	0.910	0.913	0.911	**0.918**	0.916
	Hits@10	0.906	0.910	0.909	**0.914**	**0.914**
Co.Physics	Hits@5	OOM	OOM	0.950	**0.953**	**0.953**
	Hits@10	OOM	OOM	0.946	0.949	**0.950**

Table 5.6 Performance on similarity search. Surprisingly, CARL-G performs fairly well on this task, despite not being explicitly optimized for this task (unlike AFGRL, which uses KNN during training).

5.4.4 Implementation Details

For fair evaluation with other baselines, we elect to use a standard GCN [107] encoder. Our focus is on the overall framework rather than the architecture of the encoder. All of our baselines also use GCN layers. Following [183, 113], we use two-layer GCNs for all datasets and use a two-layer MLP for the predictor network.We implement our model with PyTorch [143] and PyTorch Geometric [54]. A copy of our code is publically available at `https://github.com/willshiao/carl-g`. We adapt the code from [173], which contains implementations of BGRL [183], GRACE [236], and GBT [15] to use the split and downstream tasks from [113]. We also use the official implementation of AFGRL [113]. We perform 50 runs of Bayesian hyperparameter optimization on each dataset and task for each of our 3

methods. The hyperparameters for our results are available at that link. All of our timing experiments were conducted on Google Cloud Platform using 16 GB NVIDIA V100 GPUs.

5.4.5 Limitations & Future Work

While our proposed framework has been shown to be highly effective in terms of both training speed and performance across the 3 tasks, there are also some limitations to our approach. One such limitation is that we use hard clustering assignments, i.e., each node is assigned to exactly one cluster. This can pose issues for multi-label datasets like the Protein-Protein Interaction (PPI) [238] graph dataset. One possible solution to this problem is to perform soft clustering and use a weighted average of CVIs for second/tertiary cluster assignments, but this would require major modifications to our method, and we reserve an exploration of this for future work.

5.5 Additional Related Work

Deep Clustering. A related, but distinct, area of work is deep clustering, which uses a neural network to directly aid in the clustering of data points [4]. However, the fundamental goal of deep clustering differs from graph representation learning in that the goal is to produce a clustering of the graph nodes rather than just representations of them. An example of this is DEC [202], which uses a deep autoencoder with KL divergence to learn cluster centers, which are then used to cluster points with k-means.

Clustering for Representation Learning. There exists work that uses clustering to learn embeddings [231, 27, 204]. Notably, DeepCluster [27] trains a CNN with standard

cross-entropy loss on pseudo-labels produced by k-means. Similarly, [204] simultaneously performs clustering and dimensionality reduction with a deep neural network. The key difference between those models and our proposed framework is that we use graph data and CVI-based losses instead of traditional supervised losses.

Clustering for Efficient GNNs. There also exists work that uses clustering to speed up GNN training and inference. Cluster-GCN [35] samples node blocks produced by graph clustering algorithms and speeds up GCN layers by limiting convolutions within each block for training and inference. However, it is worth noting that it computes a fixed clustering, rather than updating the clustering jointly with our model (unlike CARL-G). FastGCN [31] does not explicitly cluster nodes but uses Monte Carlo importance sampling to similarly reduce neighborhood size and improve the speed of GCNs.

Efficient k-means. Over the years, many variants and improvements to k-means have been proposed. The original method proposed to solve the k-means assignment problem was Lloyd's algorithm [126]. Since then, several more efficient algorithms have been developed. Bottou and Bengio [20] propose using stochastic gradient descent for finding a solution. Sculley [165] further builds on this work by proposing a k-means variant that uses mini-batching to dramatically speed up training. Finally, approximate nearest-neighbor search libraries like FAISS [96] allow for efficient querying of nearest neighbors, further speeding up training.

5.6 Conclusion

In this chapter, we are the first to introduce Cluster Validation Indexes in the context of graph representation learning. We propose a novel CVI-based framework and investigate trade-offs between different CVI variants. We find that the loss function based on the simplified silhouette achieves the best overall performance to runtime ratio. It outperforms all baselines across 4/5 datasets in node classification and node clustering tasks, training up to 79× faster than the best-performing baseline. It also performs on-par with the best performing node similarity search baseline while training 1,500× faster. Moreover, to more comprehensively understand the effectiveness of CARL-G, we establish a theoretical connection between the silhouette and the well-established margin loss.

5.7 Appendix

5.7.1 Full Proof of Equivalency to Margin Loss

Proof. For ease of analysis, we work with the simplified silhouette loss (Theorem 1) and the non-max margin loss (Theorem 2). Let $\mathcal{L}$ be the set of class labels, and $\mathcal{L}_u$ be the class label for node u. Let $\mathcal{C}_u$ be the cluster assignment for node u, and $c = |C|$ be the number of clusters/classes. We define the expected inter-class and intra-class distances as follows:

$$\mathbb{E}\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v)\right] = \begin{cases} \alpha & \text{if } \mathcal{L}_u = \mathcal{L}_v \\ \\ \beta & \text{otherwise} \end{cases}, \tag{5.31}$$

where $\alpha, \beta \in \mathbb{R}^+$. Next, let

$$P\left((u,v) \in \mathcal{E}\right) = \begin{cases} p & \text{if } \mathcal{L}_u = \mathcal{L}_v \\ \\ q & \text{otherwise} \end{cases}, \tag{5.32}$$

i.e., $\mathcal{G}$ follows a stochastic block model with a probability matrix $P \in [0,1]^{c \times c}$ of the form:

$$P = \begin{bmatrix} p & q & q & q \\ q & \ddots & \ddots & \ddots \\ q & \ddots & \ddots & \ddots \\ q & \ddots & \ddots & p \end{bmatrix}. \tag{5.33}$$

Note that q does not necessarily equal $1 - p$. We define the inter-class clustering error rate ϵ and intra-class clustering error rate δ as follows:

$$P(\mathcal{C}_u \neq \mathcal{C}_v | \mathcal{L}_u = \mathcal{L}_v) = \epsilon \tag{5.34}$$

$$P(\mathcal{C}_u = \mathcal{C}_v | \mathcal{L}_u \neq \mathcal{L}_v) = \delta\,. \tag{5.35}$$

To find $\mathbb{E}[s_s(u)]$, we first find $\mathbb{E}[a(u)]$ and $\mathbb{E}[b_s(u)]$:

$$\mathbb{E}[a(u)] = \mathbb{E}\left[\frac{1}{|\mathcal{C}_i| - 1} \sum_{v \in (\mathcal{C}_i - \{u\})} \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v)\right] \tag{5.36}$$

$$= \mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) | \mathcal{C}_u = \mathcal{C}_v\right] \tag{5.37}$$

$$= P(\mathcal{L}_u = \mathcal{L}_v) \cdot P(\mathcal{C}_u = \mathcal{C}_v | \mathcal{L}_u = \mathcal{L}_v) \tag{5.38}$$

$$\cdot \mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) | \mathcal{C}_u = \mathcal{C}_v \wedge \mathcal{L}_u = \mathcal{L}_v\right]$$

$$+ P(\mathcal{L}_u \neq \mathcal{L}_v) \cdot P(\mathcal{C}_u = \mathcal{C}_v | \mathcal{L}_u \neq \mathcal{L}_v)$$

$$\cdot \mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) | \mathcal{C}_u = \mathcal{C}_v \wedge \mathcal{L}_u \neq \mathcal{L}_v\right]$$

$$= \left(\frac{1}{c}\right)(1 - \epsilon)\alpha + \left(1 - \frac{1}{c}\right)\delta\beta \tag{5.39}$$

$$= \frac{\alpha}{c} - \frac{\epsilon\alpha}{c} + \delta\beta - \frac{\delta\beta}{c} \tag{5.40}$$

and

$$\mathbb{E}[b_s(u)] = \mathbb{E}\left[\frac{1}{c - 1} \sum_{j \neq i} \frac{1}{|C_j|} \sum_{v \in \mathcal{C}_j} \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v)\right] \tag{5.41}$$

$$= \mathbb{E}_{j \neq i}\left[\frac{1}{|C_j|} \sum_{v \in \mathcal{C}_j} \textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v)\right] \tag{5.42}$$

$$= \mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v | \mathcal{C}_u \neq \mathcal{C}_v)\right] \tag{5.43}$$

$$= P(\mathcal{L}_u = \mathcal{L}_v) \cdot P(\mathcal{C}_u \neq \mathcal{C}_v | \mathcal{L}_u = \mathcal{L}_v) \cdot \tag{5.44}$$

$$\mathbb{E}_v \left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) | \mathcal{C}_u \neq \mathcal{C}_v \wedge \mathcal{L}_u = \mathcal{L}_v \right]$$

$$+ P(\mathcal{L}_u \neq \mathcal{L}_v) \cdot P(\mathcal{C}_u \neq \mathcal{C}_v | \mathcal{L}_u \neq \mathcal{L}_v)$$

$$\cdot \mathbb{E}_v \left[\textsc{Dist}(\boldsymbol{h}_u, \boldsymbol{h}_v) | \mathcal{C}_u \neq \mathcal{C}_v \wedge \mathcal{L}_u \neq \mathcal{L}_v \right]$$

$$= \left(\frac{1}{c}\right) \epsilon\alpha + \left(1 - \frac{1}{c}\right)(1-\delta)\beta \tag{5.45}$$

$$= \frac{\epsilon\alpha}{c} + \beta\left(1 - \delta - \frac{1}{c} + \frac{\delta}{c}\right) \tag{5.46}$$

$$= \frac{\epsilon\alpha}{c} + \beta - \beta\delta - \frac{\beta}{c} + \frac{\delta\beta}{c}\,. \tag{5.47}$$

Now, we can find $\mathbb{E}\left[s_s(u)\right]$:

$$\mathbb{E}\left[s_s(u)\right] = \mathbb{E}\left[b_s(u)\right] - \mathbb{E}\left[a(u)\right] \tag{5.48}$$

$$= \frac{\epsilon\alpha}{c} + \beta - \beta\delta - \frac{\beta}{c} + \frac{\delta\beta}{c} - \left(\frac{\alpha}{c} - \frac{\epsilon\alpha}{c} + \delta\beta - \frac{\delta\beta}{c}\right) \tag{5.49}$$

$$= \frac{\epsilon\alpha}{c} + \beta - \beta\delta - \frac{\beta}{c} + \frac{\delta\beta}{c} - \frac{\alpha}{c} + \frac{\epsilon\alpha}{c} - \delta\beta + \frac{\delta\beta}{c} \tag{5.50}$$

$$= \frac{2\epsilon\alpha}{c} + \frac{2\delta\beta}{c} - \frac{\beta}{c} - \frac{\alpha}{c} + \beta - 2\delta\beta\,. \tag{5.51}$$

Taking its limit as $\epsilon, \delta \to 0$, we find

$$\lim_{\epsilon,\delta\to 0}\left(-\frac{2\epsilon\alpha}{c} - \frac{2\delta\beta}{c} + \frac{\beta}{c} + \frac{\alpha}{c} - \beta + 2\delta\beta\right) = \frac{\beta}{c} + \frac{\alpha}{c} - \beta\,. \tag{5.52}$$

We similarly break down the margin loss into two terms:

$$\mathbb{E}\left[\frac{1}{\mathcal{N}(u)}\sum_{v\in\mathcal{N}(u)}\textsc{Dist}(\boldsymbol{h}_u,\boldsymbol{h}_v)\right] \tag{5.53}$$

$$= \mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u,\boldsymbol{h}_v)|(u,v)\in\mathcal{E}\right] \tag{5.54}$$

$$= P(\mathcal{L}_u=\mathcal{L}_v)\cdot P((u,v)\in\mathcal{E}|\mathcal{L}_u=\mathcal{L}_v) \tag{5.55}$$

$$\cdot\,\mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u,\boldsymbol{h}_v)|(u,v)\in\mathcal{E}\wedge\mathcal{L}_u=\mathcal{L}_v\right]$$

$$+\,P(\mathcal{L}_u\neq\mathcal{L}_v)\cdot P((u,v)\in\mathcal{E}|\mathcal{L}_u\neq\mathcal{L}_v)$$

$$\cdot\,\mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u,\boldsymbol{h}_v)|(u,v)\in\mathcal{E}\wedge\mathcal{L}_u\neq\mathcal{L}_v\right]$$

$$= \left(\frac{1}{c}\right)\alpha p + \left(1-\frac{1}{c}\right)\beta q \tag{5.56}$$

$$= \frac{\alpha p}{c} + \beta q - \frac{\beta q}{c} \tag{5.57}$$

and

$$\mathbb{E}\left[\frac{1}{|\mathcal{V}-\mathcal{N}(u)-\{u\}|}\sum_{t\notin\mathcal{N}(u)}\textsc{Dist}(\boldsymbol{h}_u,\boldsymbol{h}_t)\right] \tag{5.58}$$

$$= \mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u,\boldsymbol{h}_v)|(u,v)\notin\mathcal{E}\right] \tag{5.59}$$

$$= P(\mathcal{L}_u=\mathcal{L}_v)\cdot P((u,v)\notin\mathcal{E}|\mathcal{L}_u=\mathcal{L}_v) \tag{5.60}$$

$$\cdot\,\mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u,\boldsymbol{h}_v)|(u,v)\notin\mathcal{E}\wedge\mathcal{L}_u=\mathcal{L}_v\right]$$

$$+\,P(\mathcal{L}_u\neq\mathcal{L}_v)\cdot P((u,v)\notin\mathcal{E}|\mathcal{L}_u\neq\mathcal{L}_v)$$

$$\cdot\,\mathbb{E}_v\left[\textsc{Dist}(\boldsymbol{h}_u,\boldsymbol{h}_v)|(u,v)\notin\mathcal{E}\wedge\mathcal{L}_u\neq\mathcal{L}_v\right]$$

$$= \left(\frac{1}{c}\right)(1-p)\alpha + \left(1-\frac{1}{c}\right)(1-q)\beta \tag{5.61}$$

$$= \frac{\alpha}{c} - \frac{\alpha p}{c} + \beta - \beta q - \frac{\beta}{c} + \frac{\beta q}{c}\,. \tag{5.62}$$

Re-substituting the terms into the margin loss equation, we get

$$\mathbb{E}\left[\frac{1}{|\mathcal{N}(u)|}\sum_{v\in\mathcal{N}(u)}\textsc{Dist}(\boldsymbol{h}_u,\boldsymbol{h}_v)\right. \tag{5.63}$$
$$\left.-\frac{1}{|\mathcal{V}-\mathcal{N}(u)-\{u\}|}\sum_{t\notin\mathcal{N}(u)}\textsc{Dist}(\boldsymbol{h}_u,\boldsymbol{h}_t)\right]$$

$$=\mathbb{E}\left[\frac{1}{|\mathcal{N}(u)|}\sum_{v\in\mathcal{N}(u)}\textsc{Dist}(\boldsymbol{h}_u,\boldsymbol{h}_v)\right] \tag{5.64}$$
$$-\mathbb{E}\left[\frac{1}{|\mathcal{V}-\mathcal{N}(u)-\{u\}|}\sum_{t\notin\mathcal{N}(u)}\textsc{Dist}(\boldsymbol{h}_u,\boldsymbol{h}_t)\right]$$

$$=\frac{\alpha p}{c}+\beta q-\frac{\beta q}{c}-\left(\frac{\alpha}{c}-\frac{\alpha p}{c}+\beta-\beta q-\frac{\beta}{c}+\frac{\beta q}{c}\right) \tag{5.65}$$

$$=\frac{\alpha p}{c}+\beta q-\frac{\beta q}{c}-\frac{\alpha}{c}+\frac{\alpha p}{c}-\beta+\beta q+\frac{\beta}{c}-\frac{\beta q}{c} \tag{5.66}$$

$$=\frac{2\alpha p}{c}+2\beta q-\frac{2\beta q}{c}-\frac{\alpha}{c}+\frac{\beta}{c}-\beta\,. \tag{5.67}$$

Taking its limit as $p\to 1, q\to 0$:

$$\lim_{p\to1,q\to0}\left(\frac{2\alpha p}{c}+2\beta q-\frac{2\beta q}{c}-\frac{\alpha}{c}+\frac{\beta}{c}-\beta\right) \tag{5.68}$$

$$=\frac{2\alpha}{c}-\frac{\alpha}{c}+\frac{\beta}{c}-\beta=\frac{\alpha}{c}+\frac{\beta}{c}-\beta \tag{5.69}$$

$$\therefore\lim_{p\to1,q\to0}\mathbb{E}\left[\textsc{ml}(u)\right]=\lim_{\epsilon,\delta\to0}\mathbb{E}\left[L_s(u)\right]\,. \tag{5.70}$$

■

5.7.2 Meaning of Ideal Conditions

Our analysis in Section 5.7.1 aims to show that the more traditional margin-based losses and silhouette-based losses are sensitive to different parameters and their equivalence in the best-case scenario. Here, we briefly summarize what each of those ideal conditions means:

- $p \to 1$: We approach the case where an edge exists between each node of the same class.
- $q \to 0$: We approach the case where an edge never exists between nodes of different classes.
- $\epsilon \to 0$: We approach the case where we always place two nodes in the same cluster if they are the same class.
- $\delta \to 0$: We approach the case where we never place two nodes in the same cluster if they are in different classes.

Essentially, the ideal case for a margin-loss GNN is $p \to 1$ and $q \to 0$. Conversely, the ideal case for CARL-G is $\epsilon \to 0$, $\delta \to 0$. As we mentioned in Section 5.3.3, silhouette-based loss relies on the clustering error rate rather than the inherent properties of the graph. We show that a margin-loss GNN is exactly equivalent to a mean-silhouette-loss GNN under the above conditions; however, it also follows that some equivalence can also be drawn between them for different non-ideal values of p, q, ϵ, and δ, but we feel such analysis is out of the scope of this work.

5.7.3 Additional Experiment Details

We ran our experiments on a combination of local and cloud resources. All non-timing experiments were run on an NVIDIA RTX A4000 or V100 GPU, both with 16 GB of VRAM. All timing experiments were conducted on a Google Cloud Platform (GCP) instance with 12 Intel Skylake CPU cores, 64 GB of RAM, and a 16 GB V100 GPU.

Dataset	Nodes	Edges	Features	Classes
`Wiki-CS`	11,701	216,123	300	10
`Coauthor-CS`	18,333	163,788	6,805	15
`Coauthor-Physics`	34,493	495,924	8,415	5
`Amazon-Computers`	13,752	491,722	767	10
`Amazon-Photos`	7,650	238,162	745	8

Table 5.7 Statistics for the datasets used in our work.

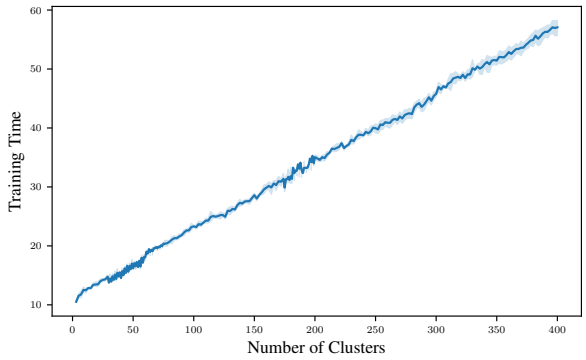


Figure 5.6 Training time versus number of clusters for CARL-G_{SIM} on `Coauthor-Physics`. As expected (see Section 5.3.1), the training time is linear with respect to the number of clusters.

Accuracy means and standard deviations are computed by re-training the classifier on 5 different splits. The code and exact hyperparameters for this paper can be found online at `https://github.com/willshiao/carl-g`.

Method	Dataset	Max GPU Memory	Mean CPU Memory	Training Time	Layer Sizes
AFGRL	Amazon-Computers	2,637	2,671	8,311.94	[512]
	Amazon-Photos	1,221	2,615	4,659.91	[512]
	Coauthor-CS	5,537	3,038	19,533.74	[1024]
	Wiki-CS	4,177	2,647	16,185.56	[1024]
BGRL	Amazon-Computers	1,081	2,289	387.03	[256,128]
	Amazon-Photos	615	2,267	284.29	[256,128]
	Coauthor-CS	2,637	2,722	1,032.14	[512,256]
	Coauthor-Physics	4,769	3,362	1,270.93	[256,128]
	Wiki-CS	1,877	2,240	720.37	[512,256]
CARL-G_{SIM}	Amazon-Computers	2,100	2,910	8.97	[512,256]
	Amazon-Photos	1,032	2,875	9.29	[512,256]
	Coauthor-CS	1,325	3,352	13.07	[512,256]
	Coauthor-Physics	3,405	3,998	27.11	[512,256]
	Wiki-CS	1,816	2,857	15.64	[512,256]
CARL-G_{SIL}	Amazon-Computers	2,100	3,435	172.26	[512,256]
	Amazon-Photos	1,032	3,320	30.13	[512,256]
	Coauthor-CS	4,682	4,273	832.25	[512,256]
	Coauthor-Physics	10,074	4,882	524.27	[512,256]
	Wiki-CS	1,816	3,392	217.63	[512,256]
CARL-G_{VRC}	Amazon-Computers	2,100	2,875	9.15	[512,256]
	Amazon-Photos	1,032	2,843	6.04	[512,256]
	Coauthor-CS	1,325	3,320	7.57	[512,256]
	Coauthor-Physics	3,405	3,964	17.22	[512,256]
	Wiki-CS	1,816	2,826	4.08	[512,256]

Table 5.8 Performance of various methods.

Chapter 6

Recommendation Systems

Recommendation systems (RS) are an increasingly relevant area for both academic and industry researchers, given their widespread impact on the daily online experiences of billions of users. One common issue in real RS is the *cold-start* problem, where users and items may not contain enough information to produce high-quality recommendations. This work focuses on a complementary problem: recommending new users and items unseen (out-of-vocabulary, or OOV) at training time. This setting is known as the *inductive* setting and is especially problematic for factorization-based models, which rely on encoding only those users/items (and, more generally, other sparse features) seen at training time with fixed parameter vectors. However, despite its practical significance, handling OOV values is often an afterthought in many academic works due to a predominant focus on *transductive* evaluation, where all categorical values for sparse features are observed at training time. As a result, existing solutions applied in practice are often naïve, such as assigning OOV users/items to random buckets. In this work, we tackle this problem and propose approaches that better

leverage available user/item features to improve OOV handling at the embedding table level. We discuss general-purpose plug-and-play approaches that are easily applicable to most RS models and improve inductive performance without negatively impacting transductive model performance. We extensively evaluate 9 OOV embedding methods on 5 models across 4 datasets (spanning different domains). One of these datasets is a proprietary production dataset from a prominent RS employed by a large social platform serving hundreds of millions of daily active users. In our experiments, we find that several proposed methods that exploit feature similarity using LSH consistently outperform alternatives on a majority of model-dataset combinations, with the best method showing a mean improvement of 3.74% over the industry standard baseline in inductive performance. We release our code and hope our work helps practitioners make more informed decisions when handling OOV for their RS and further inspires academic research into improving OOV support in RS.

6.1 Introduction

Recommendation systems (RS) suggest items to users and have found wide adoption across a variety of domains. For example, they have been used to recommend advertisements [191, 192], movies [78], friends [159, 170], and products [44, 121] to users. These methods are studied in both academia and industry, but many aspects often differ between academic and industrial recommendation systems [162]. One such difference is their evaluation methodology.

RS research in academia primarily focuses on the *transductive* setting [188, 178], where a portion of interactions are masked out for validation and testing. Such a setting

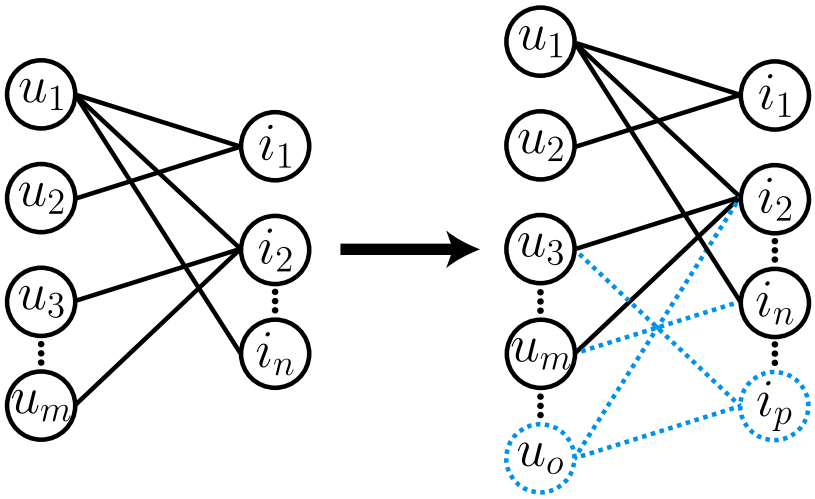


Figure 6.1 Comparison between transductive (left) and inductive (right) settings. In the transductive setting, RS are evaluated on interactions between users and items observed during training time (i.e., bold links). Whereas in the inductive setting, besides transductive interactions, RS are also evaluated on interactions related to users and items unseen during the training (i.e., both bold and dash links).

assumes that all users and items in the dataset are seen during training. However, in industrial RS environments, there is often a constant influx of new, or out-of-vocabulary (OOV), users and items that were not seen at training time, i.e., the *inductive setting* which correspond to new users and/or items showing up at validation and testing. Almost all production models are deployed to be utilized in an (at least partially) inductive setting, but a recent survey [162] found that only about 10%[1] of 88 recent RS papers evaluated their models in the fully inductive setting, in which OOV users and items are considered. Similarly, existing state-of-the-art models [23, 217, 192] also use embedding tables for sparse ID features, which face similar issues when encountering values unseen at training time since a model would not have an existing row in the embedding table for unseen values.

[1]based on an estimate from Figure 1 of [162].

While handling the inductive setting, industrial practitioners often use primitive methods such as random hashing to a fixed number of OOV buckets whose values are updated during training[2]. Such random hashing on OOV values intuitively can unlock the inductive capability for almost any transductive RS without affecting transductive performance and have been incorporated into industrial RS infrastructures as default options [1]. However, while simple to implement, these primitive methods can easily map two very different OOV users/items to the same embedding bucket, which is known as embedding collision and can greatly affect the recommendation performance [125, 220]. In Figure 6.2, we show that with a primitive method like assigning OOV values to random buckets, there exists a clear gap between inductive and transductive performance for all datasets. This demonstrates the importance of properly handling OOV values and leads us to the following question: **Can we re-imagine how we handle OOV users and items to improve the inductive capability of RS?**

While many methods in literature could potentially be used to solve this problem, we constrain our search to methods that meet criteria important to industry practitioners:

- *Efficient*: the OOV embedding method should run in sub-linear time with respect to the total number of users/items.
- *Maintains Transductive Performance*: active users and popular items are often the platform's main income sources. Hence, the OOV embedding method should not sacrifice the base model's performance on non-OOV items.

[2]Some examples of industry usage are `https://engineering.linkedin.com/blog/2023/enhancing-homepage-feed-relevance-by-harnessing-the-power-of-lar`, `https://blog.taboola.com/preparing-for-the-unexpected/` and in the Monolith source code [125].

- *Model-Agnostic*: the OOV embedding method should be applicable to RS with different model architectures.

Given the above criteria, this work explores existing and proposes new OOV embedding methods. These methods range from simply using a zero vector to feature-similarity-based methods. In our experiments, we thoroughly evaluate nine different OOV embedding methods (detailed in section 6.3) to provide a broad empirical understanding of the performance of different OOV strategies. Among the 9 OOV methods, inspired by feature-based cold-start work [111], we propose using several feature-based methods, which utilize feature information to compensate for OOV values. In particular, we propose two locality-sensitive hashing (LSH) [61] based methods that exploit feature-similarity consistently outperform other feature or non-feature-based methods in most models-dataset combinations, with the best method showing a mean improvement of 3.74% over the industry-standard random bucket assignment method.

To properly evaluate OOV methods under inductive settings, we also create appropriate inductive datasets, as existing public datasets are (1) transductive and (2) lack user/item features. Such limitations directly contradict the setting faced in industrial recommendation systems, where we usually have rich feature information for both users and items and many OOV values. Therefore, we augment three existing open-source datasets and perform a time-based split such that unseen items naturally appear during evaluation. Furthermore, we also created a proprietary industrial dataset from a large social media company containing rich feature information to evaluate OOV methods properly under real applications.

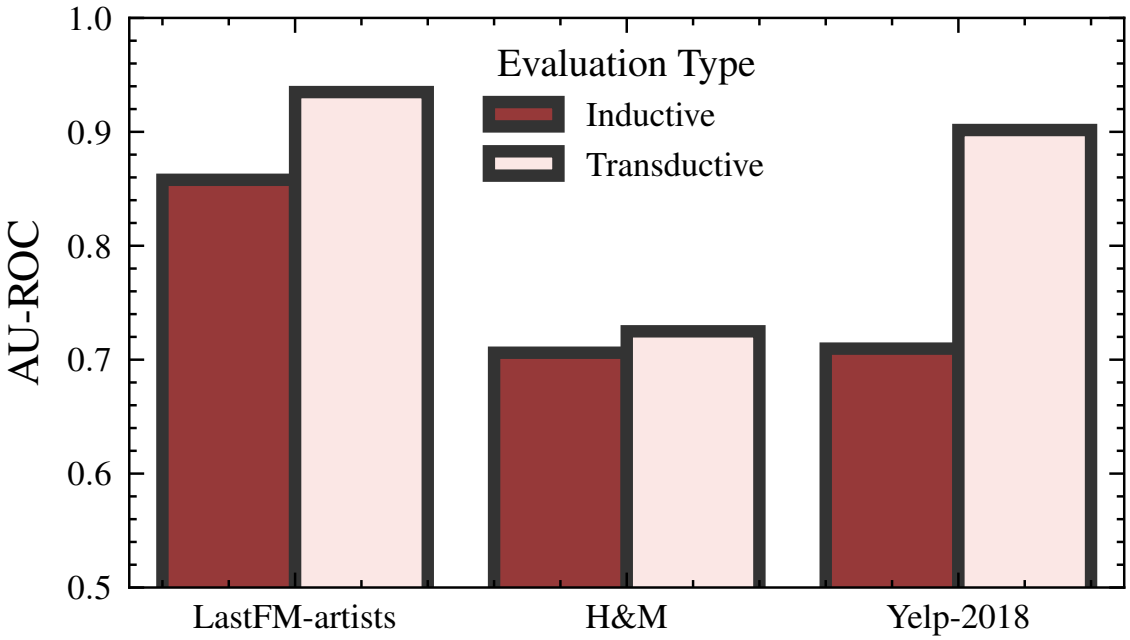


Figure 6.2 Comparison of inductive vs transductive performance with Wide & Deep models, where OOV (inductive) values are handled with trained random buckets. We see a clear gap in inductive performance vs transductive performance, showing the importance of properly handling OOV values.

Our contributions can be summarized as the following,

- To the best of our knowledge, our work is the first to provide a comprehensive empirical understanding of the performances of various OOV methods for RS.
- We demonstrate that a class of proposed feature-aware, locality-sensitive hashing-based OOV embedders that exploit feature-similarity consistently outperform existing approaches in inductive performance.
- We provide realistic inductive datasets by augmenting and splitting three open-source datasets, enabling experiments on inductive performance and OOV methods of RS, which will be publicly available upon the release of this manuscript.

- We will open-source our evaluation framework, a major extension of the popular RecBole [232] RS library that adds inductive and OOV support to encourage future research in this area.

6.2 Preliminaries and Related Work

In this section, we formally define OOV values and the user/item recommendation system problem. We detail the difference between the two classes of RS model setups that we study, delineated by the use of contextual features: context-free vs. context-aware models since OOV handling behaves differently for each class of models.

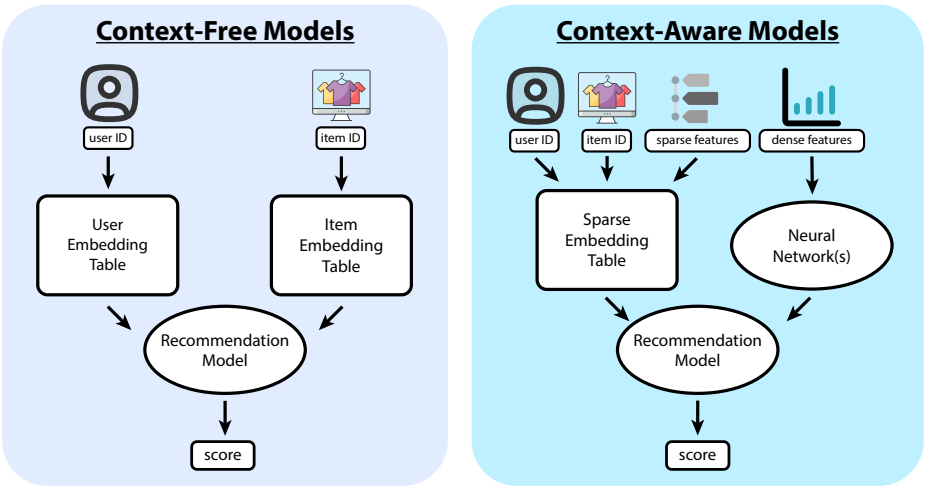


Figure 6.3 Typical structure of context-aware and context-free recommendation models.

Notation We denote the set of users as $\mathcal{U}$ and the set of items as $\mathcal{I}$. We denote the set of interactions as $\mathcal{R} \subseteq \mathcal{U} \times \mathcal{I}$. For flexibility, let $\boldsymbol{R}$ be the interaction matrix such that $\boldsymbol{R}_{u,i} = 1 \iff (u, i) \in \mathcal{R}$. Let $m = |\mathcal{U}|$ and $n = |\mathcal{I}|$ be the number of users and items, respectively. Let $\boldsymbol{U} \in \mathbb{R}^{m \times d}$ and $\boldsymbol{I} \in \mathbb{R}^{n \times d}$ be the user/item feature matrices. We assume[3]

[3]We assume users/item features to have the same dimension d for simplicity, but this can be enforced in practice with a projection layer if user/item features have different dimensions d_u and d_i respectively.

that both feature matrices are of dimension d. For a given user $u \in \mathcal{U}$, we have the associated contextual features $\boldsymbol{U}_u$. Similarly, for a given item $i \in \mathcal{I}$, we have the associated features $\boldsymbol{I}_i$. $\text{cmean}(\cdot) : \mathbb{R}^{n \times d} \rightarrow \mathbb{R}^d$ is the column-wise mean of a matrix.

We split the set of users and items based on a time t. All users/items appearing before time t are considered a part of the training set, users $\mathcal{U}_{\text{train}} \subseteq \mathcal{U}$ and items $\mathcal{I}_{\text{train}} \subseteq \mathcal{I}$. The set of training interactions $\mathcal{R}_{\text{train}} \subseteq \mathcal{R}$ is also similarly created.

OOV Values We consider a value Out-Of-Vocabulary (OOV) if it is a categorical value that does not exist at training time but appears at inference time. Formally, a user u is OOV if and only if $u \notin \mathcal{U}_{\text{train}} \wedge u \in \mathcal{U}$ and an item i is OOV if and only if $i \notin \mathcal{I}_{\text{train}} \wedge i \in \mathcal{I}$. We abbreviate non-OOV values as IV (In-Vocabulary).

Transductive vs. Inductive Settings In the transductive setting, RS models are evaluated on interactions between users and items that are observed during the model training (i.e., $\mathcal{U}_{\text{eval}} \subseteq \mathcal{U}_{\text{train}}$ and $\mathcal{I}_{\text{eval}} \subseteq \mathcal{I}_{\text{train}}$). Whereas in the inductive setting, besides transductive interactions, RS models are also evaluated on interactions between users and items that do not appear during the model training (i.e., $\mathcal{U}_{\text{eval}} \cup \mathcal{U}_{\text{train}} \neq \mathcal{U}_{\text{train}}$ and $\mathcal{I}_{\text{eval}} \cup \mathcal{I}_{\text{train}} \neq \mathcal{I}_{\text{train}}$).

6.2.1 Context-Free Models

Context-free models are the ones that do not use any additional feature information other than the IDs of users or items. They are also known as latent factor models [167] and are typically based on matrix factorization (MF) [109, 108], with the goal of approximating the training interaction matrix $\boldsymbol{R}_{\text{train}} \in \mathbb{R}^{m \times n}$. Typically, they factor $\boldsymbol{R}_{\text{train}}$ into two matrices

$\boldsymbol{A} \in \mathbb{R}^{m \times d}$ and $\boldsymbol{B} \in \mathbb{R}^{n \times d}$ such that $\boldsymbol{R}_{\text{train}} \approx \boldsymbol{A}\boldsymbol{B}^\top$. The rows of $\boldsymbol{A}$ and $\boldsymbol{B}$ are the user and item embeddings, respectively.

These embeddings can be learned in a variety of ways. For example, the Non-negative Matrix Factorization (NMF) [112] of the interaction matrix can be computed via non-negative least squares or gradient descent. In this work, we focus on two popular context-free models: Bayesian Personalized Ranking (BPR) [154] and DirectAU [188].

BPR is a pairwise ranking model that learns user and item embeddings by maximizing the likelihood of observed interactions.

$$\mathcal{L}_{\text{BPR}} = \frac{1}{|\mathcal{R}|} \sum_{(u,i) \in \mathcal{R}} -\log \left(\sigma(\boldsymbol{A}_u \cdot \boldsymbol{B}_i^\top - \boldsymbol{A}_u \cdot \boldsymbol{B}_{i'}^\top) \right), \tag{6.1}$$

where i' is a randomly sampled item from $\mathcal{I}$ such that $(u, i') \notin \mathcal{R}$ and $\sigma(\cdot)$ is the sigmoid function. In Equation (6.1), $\boldsymbol{A}_u \cdot \boldsymbol{B}_i^\top$ could be regarded as the dot-product similarity between user u and item i. One recent work [80] replaces the dot-product similarity by an MLP to infer a similarity score. Unlike BPR that utilizes negative sampling (i.e., $\boldsymbol{A}_u \cdot \boldsymbol{B}_{i'}^\top$ in Equation (6.1)) for training, DirectAU [188] is a loss function that instead directly optimizes for alignment and uniformity — factors that have been shown to be important for representation quality [194]. It is worth noting that these are often used as retrieval models in production [170], which is why we evaluate them as such in our experiments.

6.2.2 Context-Aware Models

Context-aware models utilize complimentary contextual features in addition to the user or item IDs. They are often based on the two-tower architecture [89], where each tower is responsible for embedding the user and item features, respectively. The two towers output

embeddings of the same dimensionality, allowing them to be directly compared to produce a score for each user-item pair.

However, these models are very dependent on the quality of the input contextual features. In production, practitioners often produce cross-features [34] that capture the interactions between features. As such, we focus on 3 context-aware models that incorporate these cross-features: Wide & Deep [34], eXtreme Deep Factorization Machine (xDeepFM) [117], and Deep & Cross Networks V2 (DCN-V2) [192]. We focus on these models as they are three of the most popular context-aware models in practice. The models are often used during the ranking or re-ranking stage in production pipelines, hence we evaluate them using ranking metrics in our experiments.

The features used in context-aware models are typically categorized into two types: sparse and dense. Sparse features are categorical features that are typically one-hot or multi-hot encoded. Dense features are continuous features. For example, in the case of social media content recommendation, a user's country could be a sparse feature and their mean daily app usage could be a dense feature. Sparse features are typically embedded using an embedding table where each row represents the embedding for that feature's ID. These tables are typically randomly initialized and gradually updated during training. Dense features are typically either unmodified or passed through neural network layers. In this work, we focus primarily on the handling of OOV values in sparse features — specifically, the user/item IDs, which are most likely to be OOV in production settings.

6.3 Towards a General OOV Embedder

The motivation for this work stems from how OOV users/items are typically handled in real-world production settings. In practice, OOV users/items are often assigned to a random bucket within which all values share the same embedding or are simply assigned completely random embeddings[2]. This clearly results in poor performance for any pure ID-based models (e.g., factorization-based) that rely on stored embeddings for users/items seen at training time. However, even for models that use features, this can still result in poor performance since poorly-assigned embeddings simply add additional noise. For example, random bucket assignment for OOV users means that two OOV users have the same chance to share an embedding, regardless of how similar/different they are.

Since our goal is to improve OOV support for most general recommendation systems, regardless of specific model architecture, we limit the scope of our modifications to a component that is used in almost all production recommendation systems: the embedding table. In this work, we focus primarily on OOV support for unseen user/item IDs, but the same ideas can also be easily extended to improve support for unseen categorical values in other features. This leads to the following formal definition of an OOV embedder:

OOV Embedders A user OOV embedder $f_{\text{user}} : \mathcal{U} \setminus \mathcal{U}_{\text{train}} \to \mathbb{R}^d$ maps an OOV user to a real-valued embedding. An item OOV embedder does the same: $f_{\text{item}} : \mathcal{I} \setminus \mathcal{I}_{\text{train}} \to \mathbb{R}^d$.

For the sake of simplicity, we describe all the following OOV embedders in terms of OOV users, but we use them for both OOV users and items during evaluation. They can be easily converted to item OOV embedders by substituting the appropriate variables.

Embedder	`zero`	`mean`	`rand`	`r-bucket`	`knn`	`dhe`	`fdhe`	`dnn`	`m-lsh`	`s-lsh`
Requires training	✗	✗	✗	✓	✗	✓	✓	✓	✓	✓
Uses user/item ID	✗	✗	✓	✓	✗	✓	✓	✗	✗	✗
Uses trainble OOV buckets	✗	✗	✗	✓	✗	✗	✗	✗	✓	✓
Uses features	✗	✗	✗	✗	✓	✗	✓	✓	✓	✓
Same features → same embedding	✗	✓	✗	✗	✓	✗	✗	✓	✓	✓
Requires pre-processing	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗
Complexity	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$<O(n)$[1]	$O(\theta)$	$O(\theta)$	$O(\theta)$	$O(b)$	$O(b)$
Potential unique embeddings	1	1	1	b	$> n$	$> n$	$> n$	$> n$	$> n$	b

Table 6.1 Comparison of the different OOV embedders evaluated in this work. For applicable methods, θ refers to the number of parameters in the neural network, b refers to the number of buckets, and n is the number of input items. *Features* refer to non-ID features. We assume the embedding dimensionality is constant for the complexity analysis.

[1] The exact complexity here is difficult to compute since we rely on approximate nearest neighbor search [96, 70].

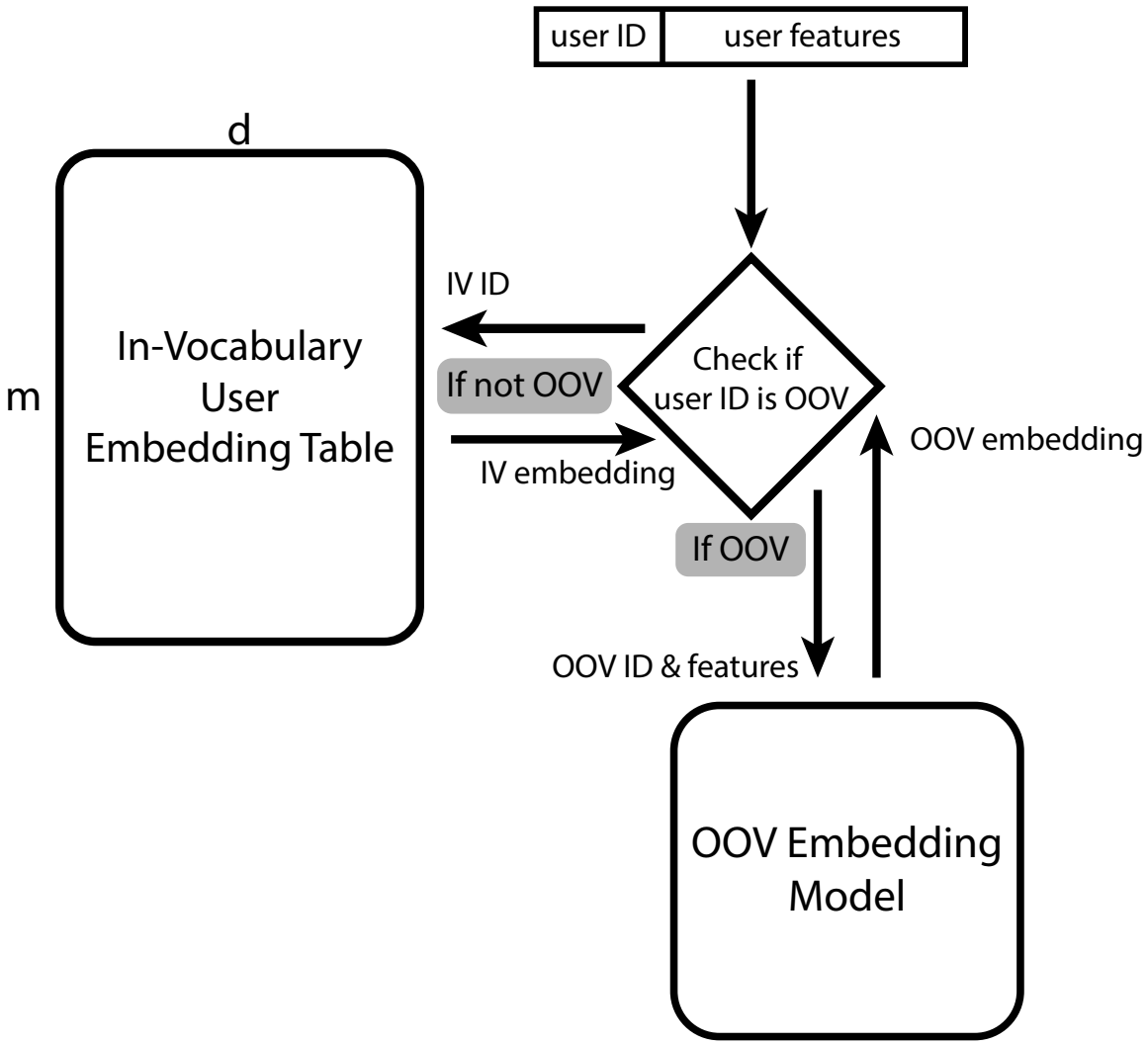


Figure 6.4 How IV/OOV user IDs are handled under our framework. Item IDs are handled the same way.

6.3.1 Heuristic-based Embedders

In this work, we first introduce several heuristic-based OOV embedding models that do not require additional trainable parameters. These are straightforward to apply in practice due to their speed and ease of implementation.

Zero Embedder `zero` simply uses the zero vector for all OOV inputs. This is a simple solution sometimes used in Natural Language Processing (NLP) for OOV words [127, 131]. Formally, $f_{\text{zero}}(\cdot) = \{0\}^c$. For context-free models, all new users will randomly select items (we ensure that items with the same score will be randomly selected without bias towards their ID). For context-aware models, all predictions will depend entirely on a user's/item's contextual information.

Mean Embedder `mean` uses the column-wise mean of the embedding matrix for all OOV IDs. Note that users and items use their respective means. Formally, for users, $f_{\text{mean}}(\cdot) = \text{cmean}(\boldsymbol{U})$. For context-free models, this means that the RS model will recommend the same popular items to all new users and that all new items will have the same probability of being recommended.

Fixed Random Embedder `rand` returns a random floating point vector for all OOV IDs. There are b fixed random vectors for each ID type (e.g., user ID, item ID). This ensures that the model's output is deterministic for users/items. Formally, for a set of random vectors $\mathcal{V} = \{\boldsymbol{v} \in \mathbb{R}^d\}$, we have $f_{\text{rand}}(\cdot) = \mathcal{V}_{g(u)}$ where g is a random hash function[4] $\mathbb{Z} \to \{1, 2, \ldots b\}$.

[4]We use the three-round integer hash function from `https://github.com/skeeto/hash-prospector`.

This approach is similar to generating a random vector, except that (a) the output for a given ID is deterministic, and (b) the maximum amount of memory used is bounded by b.

KNN Embedder `knn` returns the mean of the k nearest neighbors of a given point, as measured by the inner product of the features. Formally, for a user u, we have $f_{\text{knn}}(u) = \frac{1}{k}\sum_{a\in\text{K-NEAREST}(u)} \boldsymbol{U}_a$. With $k = 2$, this is similar to the double-hashing performed by Zhang et al. [220], except that we use feature similarity instead of random hashing to select rows. In order to meet the efficiency criteria mentioned in Section 6.1, we use approximate nearest neighbor search through libraries like FAISS [96] and SCaNN [70]. Each training ID's k-nearest neighbors can optionally be pre-computed and stored to prevent additional overhead during training.

6.3.2 Learning-based Embedders

We also consider a set of trained embedders that are optimized during the training of the base model. As mentioned in Section 6.3.3, we freeze the non-OOV parameters of the base model to avoid affecting its transductive performance. Some of these methods use an embedding table with b rows, where each row corresponds to an OOV *bucket*. This value can be tuned depending on the expected number of OOV values. In the following paragraphs, we introduce several different learning-based OOV embedders. We describe how these embedders are optimized in Section 6.3.3.

Random Buckets `r-bucket` randomly assigns an embedding (denoted as a bucket) to a given OOV ID. This mapping is done with a deterministic hash function[4] to ensure that the bucket mappings remain consistent. The chance of any bucket being selected is uniform.

Given o OOV IDs and b buckets, each bucket's expected number of OOV IDs is o/b. This is similar to `rand`, except that the values in each bucket are optimized during training. This is TensorFlow's [1] default approach for handling OOV values. PyTorch-style pseudocode for this approach can be found in **??** 2.

DHE Deep Hash Embedding (DHE) [99] substitutes a deep neural network for the embedding table. To ensure determinism for a given ID, they first compute many hashes on that ID and use those as inputs to the neural network. We use SipHash [9] with different key values as the hash functions for our implementation. DHE was originally created as a drop-in replacement for the main embedding table in context-free methods, but we use it as an OOV embedded (only on OOV IDs) since it naturally works in this case.

F-DHE Kang et al. [99] mentions that DHE can also incorporate user features. `fdhe` uses the concatenation of the user/item feature vector with the hash inputs (as with DHE) for the input to a deep neural network. This incorporates user/item features into the OOV embedding. However, compared to `dnn`, it also assigns a unique embedding for each user/item ID, even if they share the same features.

DNN `dnn` is a simple feed-forward deep neural network that takes in the user/item features as input and outputs a real-valued vector. This embedder can be viewed as a modification to `fdhe` that omits the hash-related features. As a result, users/items with the same features will share the same embedding.

Mean LSH `m-lsh` is a locality-sensitive hashing (LSH) [40] based OOV embedder. It uses a random projection matrix to map a user/item ID to a binary vector. It then uses this binary vector to index into the OOV embedding table and returns the column-wise mean of the rows where the binary vector is 1. This helps ensure that similar users/items have similar embeddings, even if their LSH vector is not exactly the same. The projection matrix remains constant, but the OOV embedding table values are updated during training. PyTorch-style pseudocode for this embedder can be found below in **??** 3.

Single LSH `s-lsh` is similar to `m-lsh` but instead treats the binary vector as a single index into the OOV embedding table. This means similar users/items with the same LSH vector will have the same embedding. Conversely, users/items with different LSH hashes will have completely different embeddings. As with `m-lsh`, the projection matrix remains constant, but the OOV embedding table values are updated during training.

For all of the embedders, we implement *per-feature* normalization — we normalize each feature vector individually before concatenating them together. This is done to ensure that the distance between two users/items is not dominated by a single feature. Otherwise, long, dense features (like content embeddings) or lists of categorical features (like watch history) could dominate the similarity computations for OOV embedding methods like KNN.

6.3.3 OOV Embedder Training

The training procedure does not need to be modified for the untrained OOV embedders (Section 6.3.1) — they can be applied to a pre-trained model. However, trained embedders (Section 6.3.2) add additional parameters that need to be optimized over OOV

Algorithm 2: PyTorch-style pseudocode for the `r-bucket` OOV embedder.

```
# oov_id: OOV user/item ID
# oov_table: OOV embedding table.
# Each row of the table is an OOV bucket
def rbucket_embed(oov_id, table):
    # b is the number of rows in oov_table
    b = oov_table.size(0)
    # hash_func is a deterministic hash function
    # that returns an integer
    hashed_id = hash_func(row_features)
    # we use the selected bucket's embedding
    return oov_table[hashed_id % b]
    # the bucket is updated via backpropagation
```

users/items. With a time-based inductive dataset split (details in Section 6.4), our training set contains only IV values, and the test set contains OOV values. OOV embedders are only used on OOV values so there is no training data for their parameters if only use the training set. As such, there are two main ways to generate OOV data in training for optimizing our OOV embedders: (1) withhold training data and use it as OOV samples or (2) generate synthetic OOV samples from the training data.

Withholding Data Withholding training data to use as OOV samples is the simplest method, but it also reduces the amount of data available for training. This also complicates

Algorithm 3: PyTorch-style pseudocode for the `m-lsh` OOV embedder.

```
# row_features: vector of the user/item features.
# oov_table: OOV embedding table.
# Each row of the table is an OOV bucket
def lsh_embed(row_features, oov_table):
    # lsh_hash is a binary vector
    lsh_hash = random_projection(row_features)
    # get col-wise mean of rows where vec is 1
    return oov_table[lsh_hash].mean(axis=1)
    # oov_table is updated via backpropagation
```

evaluation when benchmarking trained embedders against untrained embedders since the untrained embedders do not have access to the withheld data. Reducing the amount of data available for transductive training worsens transductive performance, violating the criteria defined in Section 6.1. For this reason, we choose to use synthetic OOV samples. However, the withholding data approach may be useful in production settings where we often cannot afford to maintain a unique embedding table entry for every user/item and may treat low-frequency IDs as OOV values.

Synthetic Data A simple way to train OOV embedders without affecting existing performance is to generate synthetic OOV samples. For each user/item, we create an OOV version of it that has the same interactions. We then select a subset with ratio α of the OOV samples each epoch to use for training. We then perform feature masking, a common

augmentation for self-supervised learning [236, 183], with mask rate β on the features of the OOV samples. This ensures that generated samples do not have the exact same features as the input samples. There are three types of OOV interactions: (IV user) $\rightarrow$ (OOV item), (OOV user) $\rightarrow$ (IV item), and (OOV user) $\rightarrow$ (OOV item). We generate each type with equal probability — although, in practice, this can be tuned to match the expected distribution of OOV interactions in production.

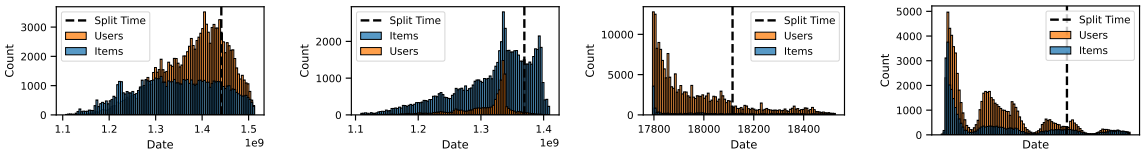
Maintaining Transductive Performance When training our OOV embedder, our aim is to maintain the performance of the transductive portion of the model. For example, with synthetic training, interactions that only involve one OOV user/item will result in undesirable updates to the main embedding table. To avoid this, we split each epoch into two training steps. In the first step, we train the model on the original training data — as we normally would in transductive training. There are no OOV values at this point, so it does not affect any trainable parameters in the OOV embedder. In the second step, we freeze the main embedding table weights and train the model on the synthetic OOV samples. The only parameters that can be updated at this step are those of the OOV embedder. We also checkpoint and restore the optimizer state before and after the second step. This ensures that the OOV training does not affect the transductive portion of the model.

6.4 Datasets

As mentioned in Section 6.2, following suggestions from recent works [179, 92], we split the datasets based on a time t. We select t for each dataset by computing the first time each user/item appeared. We then select a time t such that 20% of the users/items are

Dataset	IV Users / Items	OOV Users / Items	Mean User / Item Deg.	# User / Item Cat. Feat.	# User / Item Float Feat.
`Yelp-2018`	126,379 / 79,238	28,140 / 13,078	13.74 / 21.92	3 / 7	18 / 6
`LastFM-artists`	11,962 / 76,152	342 / 17,190	53.69 / 8.39	1 / 2	45 / 1
`H&M`	200,749 / 18,871	36,961 / 7,024	12.76 / 135.71	7 / 12	0 / 3
`Content`	74,700 / 30,757	6,610 / 3,941	24.47 / 59.42	57 / 339	172 / 899

Table 6.2 Statistics for each of the datasets used in this work. The number of float/dense features counts the number of distinct dense vectors, not the total number of floating point values (e.g., text embeddings count as a single float feature).



(a) `Yelp-2018` (b) `LastFM-artists` (c) `H&M` (d) `Content`

Figure 6.5 Visualization of where the inductive split occurs on the datasets. The x-axis is the time that the user/item first appeared. Everything to the left of the split time is used for training and validation. The remainder is used for evaluation.

OOV. Formally, select t such that $|\mathcal{U}_{\text{train}}| + |\mathcal{I}_{\text{train}}| \approx 0.8(n+m)$. This results in a naturally different distribution of OOV users compared to OOV items for each of the four datasets. Plots of the relative user/item distributions can be seen in Figure 6.5.

We benchmark various transductive recommendation system methods across four different datasets. Below, we briefly describe how we processed each of the datasets. Representative statistics for each dataset can be found in Table 6.2.

Yelp The `Yelp-2018` dataset consists of user reviews of businesses on Yelp from the 2018 Yelp Dataset Challenge[5]. We start with the version of the dataset provided by RecBole [232]. We then sample 75% of the users/items and perform 5-core filtering. We also clean up each feature by removing invalid values, normalizing floating point values, and imputing missing

[5] `https://www.yelp.com/dataset`

values with scikit-learn [145]. We also remove low-frequency values in categorical features and normalize all strings. Finally, we add text vectors for each business name. We use 300-dimensional GloVe [148] vectors for this purpose.

LastFM The `LastFM-artists` dataset [161] consists of user/artist interactions on LastFM gathered in 2014. We start with the LastFM-1b version of the dataset provided by RecBole [232] and sample 10% of the users and items. We perform the feature cleaning as with the `Yelp-2018` dataset and add GloVe vectors for each artist's name.

H&M The `H&M` dataset consists of user/item purchases on the H&M website. The raw dataset is taken from the H&M Kaggle competition[6] and we sample 30% of the users/items. We compute GloVe vectors for each item's name and use a pre-trained Vision Transformer [45] to extract features from each item's image. We also perform the same feature cleaning as with the `Yelp-2018` dataset.

Content The `Content` dataset is a proprietary user-item interaction dataset from a large social platform serving hundreds of millions of daily active users. The data is gathered from 5 days of production traffic over users sampled from a single country. We only collect users who are 18 years old and above. The `Content` dataset has rich user/item features as with many production recommendation systems. Unfortunately, due to the large number of features, we were unable to train any context-aware models with our RecBole-based [232] evaluation framework.

[6] `https://www.kaggle.com/competitions/h-and-m-personalized-fashion-recommendations`

6.5 Experimental Evaluation

	H&M			LastFM-artists			Yelp-2018		
OOV Method	DCNV2	WideDeep	xDeepFM	DCNV2	WideDeep	xDeepFM	DCNV2	WideDeep	xDeepFM
`fdhe`	66.07	68.51	<u>72.1</u>	85.53	83.25	75.88	71.24	75.87	68.86
`dhe`	69.09	68.55	**74.12**	86.15	84.97	59.74	70.57	67.15	71.48
`zero`	<u>71.06</u>	71.21	69.06	81.57	86.57	84.75	71.16	72.43	76.04
`knn`	63.71	63.13	63.61	83.9	84.79	83.2	72.73	73.42	75.04
`rand`	55.17	70.49	66.76	82.14	<u>86.64</u>	85.52	78.95	74.75	73.68
`r-bucket`	65.48	70.61	68.37	**87.13**	85.79	84.24	<u>79.88</u>	70.97	76.04
`dnn`	63.87	<u>71.7</u>	71.09	86.65	84.71	<u>86.19</u>	76.23	<u>77.89</u>	<u>82.26</u>
`s-lsh`	**73.03**	**72.61**	70.64	86.37	79.71	85.86	78.52	76.03	80.68
`mean`	67.72	70.72	66.12	86.49	85.79	85.16	74.9	**79.82**	73.88
`m-lsh`	70.69	71.65	71.3	<u>86.93</u>	**86.67**	**86.78**	**79.96**	76.35	**82.48**

Table 6.3 OOV user AUC of context-aware methods with different OOV embedding methods. Higher is better. The best-performing method in each column is bolded, and the second-best is underlined. Rows are sorted from lowest mean rank to highest mean rank.

6.5.1 Evaluation Details

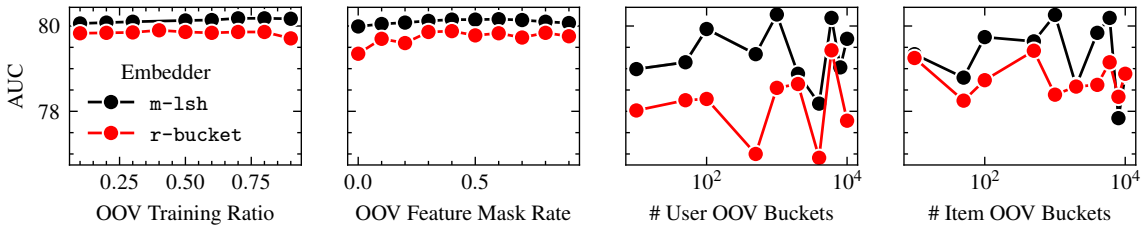


Figure 6.6 Sensitivity analysis of different training hyperparameters for `m-lsh` and `r-bucket` with WideDeep on `Yelp-2018`. Note that the y-axis range is relatively small and that the x-axis for OOV buckets is on a logarithmic scale.

Evaluation Metrics We evaluate the ranking and retrieval models separately. Following conventions from existing work [192, 191, 205, 154], we use ndcg@k (where k=20) for retrieval

models and AUROC for ranking models. In Tables 6.3 and 6.4, we report the inductive performance of OOV users. It is worth noting that the transductive performance of IV users to IV items remains the same due to how we train the OOV embedding models (see Section 6.3.3).

Experimental Details All models utilize a fork of the RecBole [232, 233] framework for experiments, in which we have made extensive modifications to the framework and models to support OOV values and swap between different OOV embedder types. We also added support for filtered evaluation on a subset of users/items. We were very careful to facilitate the easy addition of OOV support to new models. We run all experiments on Google Cloud Platform (GCP). Experiments are conducted on Google Compute Engine instances with NVIDIA Tesla P100 GPUs. The anonymized code, datasets, and hyperparameters for each of our experiments and embedders are available here: `https://github.com/snap-research/improving-inductive-oov-recsys`.

6.5.2 Context-Aware Results

The OOV user evaluation results of context-aware models are displayed in Table 6.3. On average, the best-performing OOV embedding method is `m-lsh` and the worst is `fdhe`. Unfortunately, we were unable to train context-aware models on `Content` (even in the transductive setting) using our RecBole-based framework due to the large number of features and resulting stability issues. We make the following observations:

Context helps OOV embeddings From Table 6.3, we can see that incorporating contextual information generally helps OOV embeddings. 3/4 of the best-performing OOV

Dataset	Yelp-2018		LastFM-artists		H&M		Content
Method	BPR	DAU	BPR	DAU	BPR	DAU	BPR
zero	0.79	0.79	0.93	0.93	1.15	1.15	0.71
fdhe	0.99	1.06	0.34	0.35	1.97	2.02	1.32
dhe	1.12	1.11	0.59	0.40	2.09	1.96	1.39
dnn	2.94	**4.43**	0.38	0.44	2.94	**3.36**	1.87
knn	6.95	1.58	45.69	**4.86**	5.23	1.67	**6.00**
rand	4.05	0.83	15.09	0.71	2.80	1.89	0.94
r-bucket	9.33	1.22	41.78	0.76	6.02	1.38	1.38
s-lsh	9.13	1.05	46.92	0.79	**6.19**	2.02	1.05
mean	<u>9.40</u>	<u>2.89</u>	**48.38**	0.12	6.15	1.94	<u>3.74</u>
m-lsh	**9.49**	1.49	<u>47.85</u>	<u>1.13</u>	<u>6.16</u>	<u>2.15</u>	2.02

Table 6.4 OOV user NDCG@20 of context-free methods with different OOV embedding methods. Higher is better. The best-performing method in each column is bolded and the second-best is underlined.

embedding models utilize context information. This aligns with our intuition: similar users/items should have similar embeddings. This is true for both context-free and context-aware models. In some cases, like with xDeepFM on `Yelp-2018`, the gap in AU-ROC on OOV users is as large as 6 points — showing that incorporating feature information in OOV handling can drastically improve an RS's ability to generalize to OOV users/items.

LSH-based solutions perform well Both `m-lsh` and `s-lsh` work well for the context-aware models, with one of the two methods performing the best on $^6/_9$ model/dataset combinations, as shown in Table 6.3. Across the context-aware model experiments, `m-lsh` and `s-lsh` show a mean improvement of 3.74% and 2.58% over `r-bucket` (a common industry standard[2]), respectively. They also perform well compared to the next-best method, `mean`, showing respective average improvements of 3.45% and 2.25%.

DHE-based solutions perform poorly. `dhe` and `fdhe` are the methods with the lowest average rank across the model/dataset combinations shown in Table 6.3. Surprisingly, we find that `zero` generally outperforms both `dhe` and `fdhe`. This is likely due to the additional noise the multiple hash inputs introduce to DHE-style models — a different ID results in a completely different embedding.

6.5.3 Context-Free Results

Table 6.4 shows the NDCG@20 for OOV users of BPR and DirectAU. `m-lsh` has the highest mean rank of the different OOV embedding methods. Unlike in the context-aware setting, a relatively large gap exists between different base models on the same dataset.

Surprisingly, BPR outperforms DirectAU on OOV users across all three of the datasets. We make the following observations about OOV embedding methods on the context-free models:

Improving context-free OOV performance is difficult Both BPR and DirectAU exhibit poor performance on most of the datasets, regardless of OOV embedder choice. This shows that it is difficult to encode feature information from OOV IDs in a useful manner for context-free models.

OOV Embedder choice is extremely important From Table 6.4, we can observe that there is a large gap between the best-performing models on each dataset and the worst-performing models for context-free models. This is especially true for BPR on `LastFM-artists`, where there is a 48.04 gap between the best-performing `mean` embedder and the worst-performing `fdhe` embedder. `fdhe` and `dhe` exhibit similarly poor performance across the four datasets.

6.5.4 Sensitivity Analysis

As mentioned in Section 6.3.3, there are two key hyperparameters for training the models: α (OOV sampling ratio) and β (feature masking probability). `r-bucket`, `m-lsh`, and `s-lsh` also assign values to buckets in an embedding table. These methods, therefore, have another hyperparameter b, the number of buckets, for each instance of the OOV embedding method. Since we use two instances of each OOV method (one for user IDs and one for item IDs), each model run has two bucket-related hyperparameters: b_u and b_i, the number of user/item buckets, respectively.

We conduct a sensitivity analysis on each of the four hyperparameters on `m-lsh`, the best-performing OOV embedder, and `r-bucket`, a frequently-used approach in practice[2]. The results of this analysis are shown in Figure 6.6. Generally, the performance is not very sensitive to OOV training ratio and feature mask rate hyperparameters; even for the number of user/item OOV buckets, the only performance fluctuates within a relatively small range. We can also observe that `m-lsh` outperforms `r-bucket` even under different low training ratios and high feature mask rates.

6.5.5 Recommendations for Practitioners

Based on the results of our experiments in Tables 6.3 and 6.4, we make the following recommendations for practitioners aiming to improve their performance on OOV users/items:

(1) **If contextual information (features) is available**, try using `m-lsh`. Across our experiments, `m-lsh` generally performs the best. An advantage of `m-lsh` over `s-lsh` is that it results in fewer collisions (see Table 6.1). It can also be trivially computed directly on the GPU and efficiently implemented through data structures like PyTorch's [143] `EmbeddingBag`.

(2) **If no features are available and collisions are not important**, consider using `mean`. It is extremely cheap to compute and, based on our experiments, is the best-performing untrained OOV embedder. However, all IDs will receive the same embedding, making it particularly problematic for context-free models.

(3) **If only users *or* items have features**, OOV embedding methods can be mixed. For example, in a dataset with user features but no item features, `m-lsh` could be

used for users and `mean` for items. This approach can also be used in any case where the user/item ID distributions are significantly different.

(4) **Careful caching can greatly speed up OOV training/inference**. Many of the OOV embedding methods described in this work can benefit from caching — usually for the mapping from ID to bucket(s). For example, caching the LSH vector in `m-lsh` and `s-lsh` can save a vector-matrix multiplication for each ID. Similarly, caching the k-nearest-neighbors for `knn` can save an ANN index query. This is especially true in cases where features are static or during training, during which the full set of examples is known.

6.6 Additional Related Work

Hashing for Scalable RS While this work primarily utilizes hashing to support unseen IDs, hashing is often used to improve the scalability of practical RS. One method is via the "hashing trick" [133, 200], which reduces the feature space required by categorical features. Zhang et al. [220] uses two hash functions to select and combine the embeddings of high-frequency items to form the embeddings for low-frequency items. Liu et al. [125] uses cuckoo hashing [139] on high-frequency IDs to maintain a collision-less hash table. This allows for the continuous eviction of old IDs during online training. Zhang et al. [223] improves the speed of recommendations by using LSH to limit candidate pairs. Ghaemmaghami et al. [60] proposes a novel hierarchical-clustering-based approach to hash users/items to encourage collisions between similar IDs, resulting in better performance with fewer buckets. Zhang et al. [221] formulates collaborative filtering as a binary code hashing problem, allowing users/items to be represented with binary embeddings. This allows for improvements in

speed and storage efficiency. Tan et al. [180] instead utilizes a Graph Neural Network (GNN) to learn the binary hashing function.

Cold-Start RS Methods A known issue in recommendation systems is the cold-start problem [118], which is when low-degree users and items receive poorer quality recommendations. In this work, we look specifically at the problem of OOV users/items, which means they occur exactly *zero* times in the training examples. However, cold-start methods often focus on the transductive setting where all the users/items appear at train time (although some may have very few interactions). Vartak et al. [185] focuses on the case of OOV items and proposes a meta-learning approach that uses a classifier based on user history to adjust model parameters. Wang et al. [190] extends Model-Agnostic Meta-Learning (MAML) [55] for improving cold-start recommendation performance. DropoutNet [187] uses input dropout during RS model training to improve the model's generalization to missing features. Lam et al. [111] proposes a probabilistic approach to handling OOV users on MovieLens [78].

Cold-Start Graph Methods Recommendation systems can be formulated as a *link prediction* problem on a bipartite graph, where edges represent interactions between users and items. As such, we also briefly discuss existing literature focused on improving cold-start performance on graph-related tasks. These include both training-based [88, 234] and augmentation-based [156, 230] approaches. However, due to model architecture and training differences, these approaches are not straightforward to apply to RS.

NLP OOV Handling There has also been extensive study of handling OOV values for text tokenization in the field of Natural Language Processing (NLP) [131, 17]. Modern DNN

models typically use sub-word (e.g., character or byte-level) tokens [150, 43, 131], which reduce or eliminate the chance of OOV values. However, word-level tokenizers often have to deal with OOV values, and various approaches have been proposed, including using mean and random vectors [131, 127, 128].

6.7 Conclusion

In this chapter, we explored the inductive setting in recommendation systems, where we focused on finding the best method to handle previously unseen (OOV) values. We evaluated nine different OOV embedder methods that are efficient, model-agnostic, and guaranteed to maintain transductive performance. To the best of our knowledge, this is the first comprehensive empirical study of the performance of various OOV methods for recommendation systems. Our results show that, of the nine methods, the locality-sensitive-hashing-based methods tend to be the most effective in improving inductive performance. Additionally, we augment and re-release three inductive datasets to facilitate future study of inductive performance and OOV methods in recommendation system problems. Furthermore, we derive a set of four recommendations for industrial practitioners to improve their inductive recommendation systems performance and alleviate pain points in dealing with OOV values. We hope this chapter encourages both academic and industrial researchers to further explore the inductive and OOV settings, considering their immediate practical impact in real-world, production-scale recommendation systems.

Chapter 7

Conclusion

This thesis makes significant contributions to the field of graph learning by addressing four key research questions and exploring three critical frontiers. The thesis demonstrates the importance of bridging ideas from different domains, emphasizing efficiency, and challenging established concepts to advance the state-of-the-art in graph learning.

In the area of multiplex graph generation, we introduce TenGAN, a novel method that combines ideas from tensor decompositions with modern Graph Neural Networks (GNNs) to efficiently generate multiplex graphs. By implicitly compressing the parameters in the network, TenGAN reduces the number of parameters required and preserves the interactions between different modes, overcoming the limitations of existing statistical preferential attachment models.

The thesis also makes substantial progress in non-contrastive link prediction by performing a detailed benchmark of existing methods and identifying a key limitation. By proposing a simple modification to address this limitation, the thesis demonstrates an

improvement of up to 120% in Hits@50 compared to existing non-contrastive methods and a 14× speedup over contrastive methods, highlighting the importance of efficiency in graph learning tasks.

Furthermore, the thesis establishes a theoretical connection between contrastive learning and clustering on graphs, leading to the development of CARL-G, a new framework and loss function that reformulates node representation learning as a clustering problem. By leveraging Clustering Validation Indices (CVIs) as effective substitutes for existing contrastive losses, CARL-G achieves better node representations in significantly less time, with a 79× faster training time than the best-performing node classification baseline and a 1,500× faster training time than the best-performing node clustering and similarity search baseline.

Finally, the thesis challenges the established assumption that simple out-of-vocabulary (OOV) handling methods are sufficient in recommendation systems. By exploring 9 different OOV embedding methods on 5 different models, the thesis identifies locality-sensitive hashing (LSH) based methods as a more effective approach, resulting in a 3.74% mean improvement over the industry-standard baseline. This contribution highlights the importance of improving the inductive capabilities of recommendation systems in a fast and space-efficient manner.

In summary, this thesis makes significant advances in graph learning by introducing novel methods, frameworks, and insights that address key research questions and challenges in the field. Although there remains much work to be done, its contributions have the potential to impact a wide range of applications, from social network analysis to recommendation systems, and lay the foundation for further research and development in graph learning.

7.1 Future Directions

Here we list some potential future directions:

7.1.1 Multi-Stage Recommendation System Training

Typically, recommendation systems in industrial applications consist of multiple stages. These usually consist of the retrieval, ranking, and re-ranking stages. Each stage is typically progressively more expensive than the previous stages. Each stage also usually therefore reduces the number of candidate items considered at that stage. Typically, there is not much information passed between the stages.

One potentially interesting future direction would be to transfer information between the stages. This could be in several forms. For example, information about the certainty of predictions or historical performance on items could be used to either increase or reduce the candidate count at each stage, potentially saving money or increasing ranking accuracy. Information could also be passed backward from the later ranking stages in order to facilitate distillation — each ranking stage could be the teacher for the previous one. This challenges the traditional notion in production recommendation systems that information only needs to flow in a single direction.

However, we need a flexible recommendation system evaluation framework before a large-scale study like this is feasible. Most existing recommendation frameworks like RecBole [232] only support a single model (whether retrieval or ranking). We believe it may be worth creating and open-sourcing a framework that realistically emulates multiple retrieval

and ranking stages (without the large resource requirements of a typical production-grade deployment) before attempting to conduct research on multi-stage recommendation systems.

7.1.2 Non-Contrastive Graph Learning for Other Graph Tasks

Non-contrastive graph learning was originally shown to be effective on node classification tasks [183, 15]. We also showed that it can be effective for link prediction tasks, with some modifications, in Chapter 4 above. However, there are various other graph tasks where these models could potentially be applied to great effect. It would be interesting to see how effective these non-contrastive models are for those tasks since industrial practitioners often would use the same node embeddings for various downstream tasks.

For example, non-contrastive models could potentially be used for the task of graph classification, where we are given a set of input graphs and their classes and the objective is to determine the class of a given unseen graph. This could be done by using a non-contrastive model to obtain embeddings for each graph, obtained via a readout function on the node embeddings, and training a downstream classifier. The goal would be to determine the difference in the effectiveness of using embeddings from a non-contrastive model compared to a contrastive model or a more traditional supervised model.

Another pair of interesting tasks to explore would be the node similarity search (given a node, how likely are its nearest k nearest neighbors to be within the same class) and node clustering (after running clustering on the embeddings, how pure are the clusters) tasks. We briefly evaluate the performance of BGRL on those tasks in Chapter 5 but it would be interesting to perform a more detailed and thorough evaluation across multiple non-contrastive methods.

7.1.3 Diffusion Models for Multi-view Graph Generation

Recent work has led to the development of diffusion models [82, 206] for image generation as a more stable and effective alternative to GANs [63]. Liu et al. [123] has proposed diffusion-based graph generative models. This potentially paves the way for future extensions to multi-view graphs. However, diffusion models are infamous for being costly to train, especially on large inputs. This would make it difficult to scale to larger multi-view graphs. One potential approach to this would be to compress the multi-view graph through a tensor decomposition, much as we showed was possible with TenGAN and may be worth exploring.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] Evrim Acar, Daniel M. Dunlavy, and Tamara G. Kolda. Link prediction on evolving data using matrix and tensor factorizations. In *2009 IEEE International Conference on Data Mining Workshops*, pages 262–269, 2009. doi: 10.1109/ICDMW.2009.54.

[3] Esraa Al-Sharoa, Mahmood Al-khassaweneh, and Selin Aviyente. A tensor based framework for community detection in dynamic networks. In *ICASSP*, pages 2312–

2316, New Orleans, 2017. IEEE. doi: 10.1109/ICASSP.2017.7952569.

[4] Elie Aljalbout, Vladimir Golkov, Yawar Siddiqui, and Daniel Cremers. Clustering with deep learning: Taxonomy and new methods, 2018. URL `http://arxiv.org/abs/1801.07648`.

[5] Galen Andrew, Raman Arora, Jeff Bilmes, and Karen Livescu. Deep canonical correlation analysis. In *International conference on machine learning*, pages 1247–1255. PMLR, 2013.

[6] Olatz Arbelaitz, Ibai Gurrutxaga, Javier Muguerza, Jesús M Pérez, and Iñigo Perona. An extensive comparative study of cluster validity indices. *Pattern recognition*, 46(1): 243–256, 2013.

[7] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*, pages 214–223. PMLR, 2017.

[8] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[9] Jean-Philippe Aumasson and Daniel J Bernstein. Siphash: a fast short-input prf. In *International Conference on Cryptology in India*, pages 489–508. Springer, 2012.

[10] Randall Balestriero and Yann LeCun. Contrastive and non-contrastive self-supervised learning recover global and local spectral embedding methods. *arXiv preprint arXiv:2205.11508*, 2022.

[11] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999. doi: 10.1126/science.286.5439.509. URL https://www.science.org/doi/abs/10.1126/science.286.5439.509.

[12] Muthu M. Baskaran, Thomas Henretty, James Ezick, Richard Lethin, and David Bruns-Smith. Enhancing network visibility and security through tensor analysis. *FGCS*, 96:207–215, 2019. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2019.01.039. URL https://www.sciencedirect.com/science/article/pii/S0167739X18302073.

[13] Prithwish Basu, Matthew Dippel, and Ravi Sundaram. Multiplex networks: A generative model and algorithmic complexity. In *ASONAM*, pages 456–463, 2015. doi: 10.1145/2808797.2808900.

[14] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

[15] Piotr Bielak, Tomasz Kajdanowicz, and Nitesh V Chawla. Graph barlow twins: A self-supervised representation learning framework for graphs. *Knowledge-Based Systems*, 256:109631, 2022. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2022.109631. URL https://www.sciencedirect.com/science/article/pii/S095070512200822X.

[16] Lukas Biewald. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

[17] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017.

[18] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. Netgan: Generating graphs via random walks. In *ICML*, pages 610–619. PMLR, 2018.

[19] Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. Netgan: Generating graphs via random walks. In *International conference on machine learning*, pages 610–619. PMLR, 2018.

[20] Leon Bottou and Yoshua Bengio. Convergence properties of the k-means algorithms. *Advances in neural information processing systems*, 7, 1994.

[21] Paul S Bradley and Usama M Fayyad. Refining initial points for k-means clustering. In *ICML*, volume 98, pages 91–99, San Francisco, CA, USA, 1998. Citeseer, Morgan Kaufmann.

[22] Lei Cai, Jundong Li, Jie Wang, and Shuiwang Ji. Line graph neural networks for link prediction, 2020. URL `https://arxiv.org/abs/2010.10046`.

[23] Xuheng Cai, Chao Huang, Lianghao Xia, and Xubin Ren. Lightgcl: Simple yet effective graph contrastive learning for recommendation. *arXiv preprint arXiv:2302.08191*, 2023.

[24] Tadeusz Caliński and Jerzy Harabasz. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27, 1974.

[25] Alessio Cardillo, Jesús Gómez-Gardeñes, Massimiliano Zanin, Miguel Romance, David Papo, Francisco del Pozo, and Stefano Boccaletti. Emergence of network features from multiplexity. *Scientific Reports*, 3(1):1344, February 2013.

[26] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka, and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, AAAI'10, page 1306–1313, Atlanta, Georgia, 2010.

[27] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European conference on computer vision (ECCV)*, pages 132–149, ECCV 2018, 2018. ECCV.

[28] Benjamin Paul Chamberlain, Sergey Shirobokov, Emanuele Rossi, Fabrizio Frasca, Thomas Markovich, Nils Hammerla, Michael M Bronstein, and Max Hansmire. Graph neural networks for link prediction with subgraph sketching. *arXiv preprint arXiv:2209.15486*, 2022.

[29] Chi Chen, Weike Ye, Yunxing Zuo, Chen Zheng, and Shyue Ping Ong. Graph networks as a universal machine learning framework for molecules and crystals. *Chemistry of Materials*, 31(9):3564–3572, 2019.

[30] Hongxu Chen, Hongzhi Yin, Weiqing Wang, Hao Wang, Quoc Viet Hung Nguyen, and Xue Li. Pme: Projected metric embedding on heterogeneous networks for link prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 1177–1186, New York, NY, USA,

2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3219986. URL https://doi.org/10.1145/3219819.3219986.

[31] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling, 2018.

[32] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, Proceedings of Machine Learning Research, pages 1597–1607, 2020.

[33] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15750–15758, 2021.

[34] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[35] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 257–266, New York, NY, USA, 2019. Association for Computing Machinery.

[36] Fengyu Cong, Qiu-Hua Lin, Li-Dan Kuang, Xiao-Feng Gong, Piia Astikainen, and Tapani Ristaniemi. Tensor decomposition of eeg signals: A brief review. *Journal of*

Neuroscience Methods, 248:59–69, 2015. ISSN 0165-0270. doi: https://doi.org/10.1016/j.jneumeth.2015.03.018. URL https://www.sciencedirect.com/science/article/pii/S0165027015001016.

[37] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, page 39–46, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605589060. doi: 10.1145/1864708.1864721. URL https://doi.org/10.1145/1864708.1864721.

[38] Leon Danon, Albert Diaz-Guilera, Jordi Duch, and Alex Arenas. Comparing community structure identification. *Journal of statistical mechanics: Theory and experiment*, 2005 (09):P09008, 2005.

[39] Sanjoy Dasgupta. *The hardness of k-means clustering*. Department of Computer Science and Engineering, University of California, San Diego, San Diego, CA, USA, 2008.

[40] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.

[41] Nicola De Cao and Thomas Kipf. MolGAN: An implicit generative model for small molecular graphs. 5 2018. URL https://arxiv.org/abs/1805.11973.

[42] Austin Derrow-Pinion, Jennifer She, David Wong, Oliver Lange, Todd Hester, Luis Perez, Marc Nunkesser, Seongjae Lee, Xueying Guo, Brett Wiltshire, et al. Eta

prediction with graph neural networks in google maps. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 3767–3776, Queensland, Australia, 2021. ACM.

[43] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. URL `https://arxiv.org/abs/1810.04805`.

[44] Ruihai Dong, Michael P O'Mahony, Markus Schaal, Kevin McCarthy, and Barry Smyth. Combining similarity and sentiment in opinion mining for product recommendation. *Journal of Intelligent Information Systems*, 46(2):285–312, 2016.

[45] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[46] Daniel M Dunlavy, Tamara G Kolda, and Evrim Acar. Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(2):1–27, 2011.

[47] J. C. Dunn. Some recent investigations of a new fuzzy partitioning algorithm and its application to pattern classification problems. *Journal of Cybernetics*, 4(2):1–15, 1974. doi: 10.1080/01969727408546062. URL `https://doi.org/10.1080/01969727408546062`.

[48] P. Erdös and A. Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6: 290, 1959.

[49] Evangelos E. Papalexakis, Christos Faloutsos, and Nicholas D Sidiropoulos. Parcube: Sparse parallelizable tensor decompositions. In Peter A. Flach, Tijl De Bie, and Nello Cristianini, editors, *ECML-PKDD'12*, pages 521–536, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-33460-3.

[50] Brian. Everitt. *Cluster analysis /*. Heinemann Educational for the Social Science Research Council,, London :, 2001. ISBN 0435822977.

[51] Shuangfei Fan and Bert Huang. Labeled graph generative adversarial networks. *arXiv:1906.03220*, 2019.

[52] Shuangfei Fan and Bert Huang. Labeled graph generative adversarial networks. *CoRR*, abs/1906.03220:1–10, 2019. URL `http://arxiv.org/abs/1906.03220`.

[53] Wenqi Fan, Xiaorui Liu, Wei Jin, Xiangyu Zhao, Jiliang Tang, and Qing Li. Graph trend filtering networks for recommendation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 112–121, Madrid, Spain, 2022. ACM.

[54] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

[55] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[56] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In *CIKM*, page

1797–1806. ACM, 2017. ISBN 9781450349185. doi: 10.1145/3132847.3132953. URL https://doi.org/10.1145/3132847.3132953.

[57] Marzena Fügenschuh, Ralucca Gera, and Tobias Lory. A Synthetic Model for Multilevel Air Transportation Networks. In Natalia Kliewer, Jan Fabian Ehmke, and Ralf Borndörfer, editors, *Operations Research Proceedings 2017*, Operations Research Proceedings, pages 347–353. Springer, March 2018. doi: 10.1007/978-3-319-89920-6. URL https://ideas.repec.org/h/spr/oprchp/978-3-319-89920-6%5F47.html.

[58] Marzena Fügenschuh, Ralucca Gera, and Andrea Tagarelli. Angel: A synthetic model for airline network generation emphasizing layers. *IEEE Transactions on Network Science and Engineering*, 7(3):1977–1987, 2020. doi: 10.1109/TNSE.2020.2965207.

[59] Laetitia Gauvin, André Panisson, and Ciro Cattuto. Detecting the community structure and activity patterns of temporal networks: A non-negative tensor factorization approach. *PLoS ONE*, 9(1):e86028, 01 2014. doi: 10.1371/journal.pone.0086028. URL http://dx.doi.org/10.1371%2Fjournal.pone.0086028.

[60] Benjamin Ghaemmaghami, Mustafa Ozdal, Rakesh Komuravelli, Dmitriy Korchev, Dheevatsa Mudigere, Krishnakumar Nair, and Maxim Naumov. Learning to collide: Recommendation system model compression with learned hash functions. *arXiv preprint arXiv:2203.15837*, 2022.

[61] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Vldb*, volume 99, pages 518–529, 1999.

[62] Minas Gjoka, Carter T Butts, Maciej Kurant, and Athina Markopoulou. Multigraph sampling of online social networks. *IEEE Journal on Selected Areas in Communications*, 29(9):1893–1905, 2011.

[63] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[64] Derek Greene and Pádraig Cunningham. Producing a unified graph representation from multiple social network views. In *Proceedings of the 5th Annual ACM Web Science Conference*, WebSci '13, page 118–121, New York, NY, USA, 2013. ACM. ISBN 9781450318891. doi: 10.1145/2464464.2464471. URL `https://doi.org/10.1145/2464464.2464471`.

[65] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A Kernel Two-Sample Test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012. ISSN ISSN 1533-7928. URL `http://jmlr.csail.mit.edu/papers/v13/gretton12a.html`.

[66] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

[67] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pages 855–864, New York, NY, USA, 2016. ACM.

[68] Ekta Gujral and Evangelos E Papalexakis. Smacd: Semi-supervised multi-aspect community detection. In *SDM*, pages 702–710. SIAM, 2018.

[69] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *NeurIPS*, 30, 2017.

[70] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, pages 3887–3896, Cambridge, MA, USA, 2020. PMLR, PMLR.

[71] Zhichun Guo, Chuxu Zhang, Wenhao Yu, John Herr, Olaf Wiest, Meng Jiang, and Nitesh V Chawla. Few-shot graph learning for molecular property prediction. In *Proceedings of the Web Conference 2021*, pages 2559–2567, New York, NY, USA, 2021. ACM.

[72] Zhichun Guo, Bozhao Nan, Yijun Tian, Olaf Wiest, Chuxu Zhang, and Nitesh V Chawla. Graph-based molecular representation learning. *arXiv preprint arXiv:2207.04869*, 2022.

[73] Zhichun Guo, William Shiao, Shichang Zhang, Yozen Liu, Nitesh Chawla, Neil Shah, and Tong Zhao. Linkless link prediction via relational distillation. *arXiv preprint arXiv:2210.05801*, 2022.

[74] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008. SciPy.

[75] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1024–1034, Long Beach, CA, 2017. NeurIPS.

[76] Xiaotian Han, Tong Zhao, Yozen Liu, Xia Hu, and Neil Shah. Mlpinit: Embarrassingly simple gnn training acceleration with mlp initialization, 2022.

[77] Yu Hao, Xin Cao, Yixiang Fang, Xike Xie, and Sibo Wang. Inductive link prediction for nodes having only attribute information. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, jul 2020. doi: 10.24963/ijcai.2020/168. URL `https://doi.org/10.24963%2Fijcai.2020%2F168`.

[78] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

[79] Kaveh Hassani and Amir Hosein Khas Ahmadi. Contrastive multi-view representation learning on graphs. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 4116–4126, Cambridge, MA, 2020. PMLR.

[80] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[81] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and*

development in Information Retrieval, pages 639–648, New York, NY, USA, 2020. ACM.

[82] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[83] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network, 2014. URL `https://arxiv.org/abs/1412.6622`.

[84] Paul W. Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5(2):109–137, 6 1983. ISSN 0378-8733. doi: 10.1016/0378-8733(83)90021-7. URL `https://www.sciencedirect.com/science/article/abs/pii/0378873383900217`.

[85] Harold Hotelling. Relations between two sets of variates. In *Breakthroughs in statistics*, pages 162–190. Springer, 1992.

[86] Eduardo R Hruschka, Leandro Nunes de Castro, and Ricardo JGB Campello. Evolutionary algorithms for clustering gene-expression data. In *Fourth IEEE International Conference on Data Mining (ICDM'04)*, pages 403–406, New York, 2004. IEEE, IEEE.

[87] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 33:22118–22133, 2020.

[88] Weihua Hu, Kaidi Cao, Kexin Huang, Edward W Huang, Karthik Subbian, and Jure Leskovec. Tuneup: A training strategy for improving generalization of graph neural networks. *arXiv preprint arXiv:2210.14843*, 2022.

[89] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2333–2338, 2013.

[90] Nicolas Hubert, Pierre Monnin, Armelle Brun, and Davy Monticolo. New Strategies for Learning Knowledge Graph Embeddings: the Recommendation Case. In *EKAW 2022 - 23rd International Conference on Knowledge Engineering and Knowledge Management*, Bolzano, Italy, September 2022. URL `https://hal.inria.fr/hal-03722881`.

[91] Roberto Interdonato, Matteo Magnani, Diego Perna, Andrea Tagarelli, and Davide Vega. Multilayer network simplification: Approaches, models and methods. *Computer Science Review*, 36:100246, 2020. ISSN 1574-0137. doi: https://doi.org/10.1016/j.cosrev.2020.100246. URL `https://www.sciencedirect.com/science/article/pii/S1574013719301923`.

[92] Yitong Ji, Aixin Sun, Jie Zhang, and Chenliang Li. A critical study on data leakage in recommender system offline evaluation. *ACM Transactions on Information Systems*, 41(3):1–27, 2023.

[93] Wei Jin, Tyler Derr, Haochen Liu, Yiqi Wang, Suhang Wang, Zitao Liu, and Jiliang Tang. Self-supervised learning on graphs: Deep insights and new direction. *arXiv preprint arXiv:2006.10141*, 2020.

[94] Wei Jin, Xiaorui Liu, Xiangyu Zhao, Yao Ma, Neil Shah, and Jiliang Tang. Automated self-supervised learning for graphs. *CoRR*, abs/2106.05470:1–20, 2021. URL `https:`

//arxiv.org/abs/2106.05470.

[95] Wei Jin, Tong Zhao, Jiayuan Ding, Yozen Liu, Jiliang Tang, and Neil Shah. Empowering graph representation learning with test-time graph transformation, 2022.

[96] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[97] Mingxuan Ju, Tong Zhao, Qianlong Wen, Wenhao Yu, Neil Shah, Yanfang Ye, and Chuxu Zhang. Multi-task self-supervised graph neural networks enable stronger task generalization. *arXiv preprint arXiv:2210.02016*, 2022.

[98] Mingxuan Ju, Tong Zhao, Qianlong Wen, Wenhao Yu, Neil Shah, Yanfang Ye, and Chuxu Zhang. Multi-task self-supervised graph neural networks enable stronger task generalization, 2023.

[99] Wang-Cheng Kang, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Ting Chen, Lichan Hong, and Ed H Chi. Learning to embed categorical features without embedding tables for recommendation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 840–850, 2021.

[100] Zekarias T. Kefato and Sarunas Girdzijauskas. Self-supervised graph neural networks without explicit negative sampling. *CoRR*, abs/2103.14958:1–8, 2021. URL https://arxiv.org/abs/2103.14958.

[101] Ehsan Khadangi, Alireza Bagheri, and Amin Shahmohammadi. Biased sampling from facebook multilayer activity network using learning automata. *Applied Intelligence*,

45(3):829–849, Oct 2016. ISSN 1573-7497. doi: 10.1007/s10489-016-0784-0. URL https://doi.org/10.1007/s10489-016-0784-0.

[102] Jung Yeol Kim and K.-I. Goh. Coevolution and correlated multiplexity in multiplex networks. *Phys. Rev. Lett.*, 111:058702, Jul 2013. doi: 10.1103/PhysRevLett.111.058702. URL https://link.aps.org/doi/10.1103/PhysRevLett.111.058702.

[103] Minho Kim and RS Ramakrishna. New indices for cluster validity assessment. *Pattern Recognition Letters*, 26(15):2353–2363, 2005.

[104] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2013.

[105] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv:1611.07308*, 2016.

[106] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR 2017*, pages 1–14, Toulon, France, 2017. OpenReview.net. URL https://openreview.net/forum?id=SJU4ayYgl.

[107] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, pages 1–14, Toulon, France, 2017. ICLR.

[108] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434, 2008.

[109] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[110] Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. Tensorly: Tensor learning in python. *arXiv:1610.09555*, 2016.

[111] Xuan Nhat Lam, Thuc Vu, Trong Duc Le, and Anh Duc Duong. Addressing cold-start problem in recommendation systems. In *Proceedings of the 2nd International Conference on Ubiquitous Information Management and Communication*, ICUIMC '08, page 208–211, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781595939937. doi: 10.1145/1352793.1352837. URL `https://doi.org/10.1145/1352793.1352837`.

[112] D.D. Lee and H.S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[113] Namkyeong Lee, Junseok Lee, and Chanyoung Park. Augmentation-free self-supervised learning on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 7372–7380, Virtual, 2022. AAAI Press.

[114] Lars Lenssen and Erich Schubert. Clustering by direct optimization of the medoid silhouette. In *Similarity Search and Applications: 15th International Conference, SISAP 2022, Bologna, Italy, October 5–7, 2022, Proceedings*, pages 190–204, New York, NY, USA, 2022. Springer, Springer.

[115] Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD*, KDD '06, page 631–636, New York, NY, USA, 2006. ACM.

ISBN 1595933395. doi: 10.1145/1150402.1150479. URL https://doi.org/10.1145/1150402.1150479.

[116] Maosen Li, Siheng Chen, Yangheng Zhao, Ya Zhang, Yanfeng Wang, and Qi Tian. Dynamic multiscale graph neural networks for 3d skeleton based human motion prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 214–223, 2020.

[117] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1754–1763, 2018.

[118] Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. Facing the cold start problem in recommender systems. *Expert systems with applications*, 41(4):2065–2073, 2014.

[119] Shuai Lin, Chen Liu, Pan Zhou, Zi-Yuan Hu, Shuojia Wang, Ruihui Zhao, Yefeng Zheng, Liang Lin, Eric Xing, and Xiaodan Liang. Prototypical graph contrastive learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[120] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, page 2181–2187. AAAI Press, 2015. ISBN 0262511290.

[121] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

[122] Chen Ling, Carl Yang, and Liang Zhao. Deep generation of heterogeneous networks. In *2021 IEEE ICDM*, pages 379–388, Auckland, New Zealand, 2021. IEEE. doi: 10.1109/ICDM51629.2021.00049.

[123] Chengyi Liu, Wenqi Fan, Yunqing Liu, Jiatong Li, Hang Li, Hui Liu, Jiliang Tang, and Qing Li. Generative diffusion models on graphs: Methods and applications. *arXiv preprint arXiv:2302.02591*, 2023.

[124] Gang Liu, Tong Zhao, Jiaxin Xu, Tengfei Luo, and Meng Jiang. Graph rationalization with environment-based augmentations. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1069–1078, 2022.

[125] Zhuoran Liu, Leqi Zou, Xuan Zou, Caihua Wang, Biao Zhang, Da Tang, Bolin Zhu, Yijie Zhu, Peng Wu, Ke Wang, et al. Monolith: real time recommendation system with collisionless embedding table. *arXiv preprint arXiv:2209.07663*, 2022.

[126] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[127] Johannes V Lochter, Renato M Silva, and Tiago A Almeida. Deep learning models for representing out-of-vocabulary words. In *Brazilian Conference on Intelligent Systems*, pages 418–434. Springer, 2020.

[128] Johannes V Lochter, Renato M Silva, and Tiago A Almeida. Multi-level out-of-vocabulary words handling approach. *Knowledge-Based Systems*, 251:108911, 2022.

[129] J MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. 5th Berkeley Symposium on Math., Stat., and Prob*, page 281, CA, USA, 1965. University of California Press.

[130] Aditya Krishna Menon and Charles Elkan. Link prediction via matrix factorization. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 437–452. Springer, 2011.

[131] Sabrina J Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y Lee, Benoît Sagot, et al. Between words and characters: a brief history of open-vocabulary modeling and tokenization in nlp. *arXiv preprint arXiv:2112.10508*, 2021.

[132] Glenn W Milligan and Martha C Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50:159–179, 1985.

[133] John Moody. Fast learning in multi-resolution hierarchies. *Advances in neural information processing systems*, 1, 1988.

[134] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv:1707.05005*, 2017.

[135] M. E. J. Newman, D. J. Watts, and S. H. Strogatz. Random graph models of social networks. *Proceedings of the National Academy of Sciences*, 99(suppl_1):2566–2572, 2002. doi: 10.1073/pnas.012582999. URL `https://www.pnas.org/doi/abs/10.1073/pnas.012582999`.

[136] V. Nicosia, G. Bianconi, V. Latora, and M. Barthelemy. Growing multiplex networks. *Phys. Rev. Lett.*, 111:058701, Jul 2013. doi: 10.1103/PhysRevLett.111.058701. URL https://link.aps.org/doi/10.1103/PhysRevLett.111.058701.

[137] Ali Noroozi and Mansoor Rezghi. A tensor-based framework for rs-fmri classification and functional connectivity construction. *Frontiers in Neuroinformatics*, 14, 2020. ISSN 1662-5196. doi: 10.3389/fninf.2020.581897. URL https://www.frontiersin.org/article/10.3389/fninf.2020.581897.

[138] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[139] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

[140] Evangelos Papalexakis and Konstantinos Pelechrinis. Thoops: A multi-aspect analytical framework for spatio-temporal basketball data. In *Proceedings of the 27th ACM CIKM*, CIKM '18, page 2223–2232, New York, NY, USA, 2018. ACM. ISBN 9781450360142. doi: 10.1145/3269206.3272002. URL https://doi.org/10.1145/3269206.3272002.

[141] Hae-Sang Park and Chi-Hyuck Jun. A simple and fast algorithm for k-medoids clustering. *Expert systems with applications*, 36(2):3336–3341, 2009.

[142] Jiwoong Park, Minsik Lee, Hyung Jin Chang, Kyuewang Lee, and Jin Young Choi. Symmetric graph convolutional autoencoder for unsupervised graph representation learning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6519–6528, New York, NY, USA, 2019. IEEE.

[143] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:1–12, 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[144] Georgios A. Pavlopoulos, Maria Secrier, Charalampos N. Moschopoulos, Theodoros G. Soldatos, Sophia Kossida, Jan Aerts, Reinhard Schneider, and Pantelis G. Bagos. Using graph theory to analyze biological networks. *BioData Mining*, 4(1):10, Apr 2011. ISSN 1756-0381. doi: 10.1186/1756-0381-4-10. URL `https://doi.org/10.1186/1756-0381-4-10`.

[145] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[146] Dan Pelleg and Andrew Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 277–281, New York, NY, USA, 1999. ACM.

[147] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. Graph representation learning via graphical mutual information

maximization. In *Proceedings of The Web Conference 2020*, pages 259–270, New York, NY, USA, 2020. ACM.

[148] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[149] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, New York, NY, USA, aug 2014. ACM. doi: 10.1145/2623330.2623732. URL `https://doi.org/10.1145%2F2623330.2623732`.

[150] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of NAACL-HLT*, pages 2227–2237, 2018.

[151] S Yu Philip, Jiawei Han, and Christos Faloutsos. *Link mining: Models, algorithms, and applications*. Springer, 2010.

[152] Carey E. Priebe, John M. Conroy, David J. Marchette, and Youngser Park. Scan statistics on enron graphs. *Computational & Mathematical Organization Theory*, 11(3):229–247, Oct 2005. ISSN 1572-9346. doi: 10.1007/s10588-005-5378-z. URL `https://doi.org/10.1007/s10588-005-5378-z`.

[153] William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

[154] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

[155] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. Neural collaborative filtering vs. matrix factorization revisited. In *Fourteenth ACM conference on recommender systems*, pages 240–248, Online, 2020. ACM.

[156] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

[157] Giulio Rossetti. ANGEL: efficient, and effective, node-centric community discovery in static and dynamic networks. *Applied Network Science*, 5(1):26, June 2020.

[158] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. ISSN 0377-0427. doi: https://doi.org/10.1016/0377-0427(87)90125-7. URL `https://www.sciencedirect.com/science/article/pii/0377042787901257`.

[159] Aravind Sankar, Yozen Liu, Jun Yu, and Neil Shah. Graph neural networks for friend ranking in large-scale social platforms. In *Proceedings of the Web Conference 2021*, pages 2535–2546, New York, NY, USA, 2021. ACM.

[160] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

[161] Markus Schedl. The lfm-1b dataset for music retrieval and recommendation. In *Proceedings of the 2016 ACM on international conference on multimedia retrieval*, pages 103–110, 2016.

[162] Tobias Schnabel, Mengting Wan, and Longqi Yang. Situating recommender systems in practice: Towards inductive learning and incremental updates. *arXiv preprint arXiv:2211.06365*, 2022.

[163] Erich Schubert. Stop using the elbow criterion for k-means and how to choose the number of clusters instead, 2022.

[164] Erich Schubert and Peter J Rousseeuw. Faster k-medoids clustering: improving the pam, clara, and clarans algorithms. In *Similarity Search and Applications: 12th International Conference, SISAP 2019, Newark, NJ, USA, October 2–4, 2019, Proceedings 12*, pages 171–187, New York, NY, USA, 2019. Springer, Springer.

[165] David Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178, New York, NY, USA, 2010. ACM.

[166] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[167] Shalin Shah. A survey of latent factor models for recommender systems and personalization. *Authorea Preprints*, 2023.

[168] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Gunnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[169] Fatemeh Sheikholeslami and Georgios B. Giannakis. Identification of overlapping communities via constrained egonet tensor decomposition. *IEEE Transactions on Signal Processing*, 66(21):5730–5745, 2018. doi: 10.1109/TSP.2018.2871383.

[170] Jiahui Shi, Vivek Chaurasiya, Yozen Liu, Shubham Vij, Yan Wu, Satya Kanduri, Neil Shah, Peicheng Yu, Nik Srivastava, Lei Shi, et al. Embedding based retrieval in friend recommendation. 2023.

[171] William Shiao and Evangelos E. Papalexakis. Adversarially generating rank-constrained graphs. In *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–8, Porto, Portugal, 2021. IEEE. doi: 10.1109/DSAA53316.2021.9564202.

[172] William Shiao and Evangelos E Papalexakis. Adversarially generating rank-constrained graphs. In *2021 IEEE 8th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–8, New York, NY, USA, 2021. IEEE, IEEE.

[173] William Shiao, Zhichun Guo, Tong Zhao, Evangelos E Papalexakis, Yozen Liu, and Neil Shah. Link prediction with non-contrastive learning. *arXiv preprint arXiv:2211.14394*, abs/2211.14394:1–19, 2022.

[174] William Shiao, Zhichun Guo, Tong Zhao, Evangelos E Papalexakis, Yozen Liu, and Neil Shah. Link prediction with non-contrastive learning, 2022.

[175] Blaž Škrlj, Jan Kralj, and Nada Lavrač. Embedding-based silhouette community detection. *Machine Learning*, 109:2161–2193, 2020.

[176] Shaden Smith, Jee W. Choi, Jiajia Li, Richard Vuduc, Jongsoo Park, Xing Liu, and George Karypis. FROSTT: The formidable repository of open sparse tensors and tools, 2017. URL `http://frostt.io/`.

[177] Christian L Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. Networkit: A tool suite for large-scale complex network analysis. *Network Science*, 4(4):508–530, 2016.

[178] Aixin Sun. On challenges of evaluating recommender systems in an offline setting. In *Proceedings of the 17th ACM Conference on Recommender Systems*, pages 1284–1285, 2023.

[179] Aixin Sun. Take a fresh look at recommender systems from an evaluation standpoint. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2629–2638, 2023.

[180] Qiaoyu Tan, Ninghao Liu, Xing Zhao, Hongxia Yang, Jingren Zhou, and Xia Hu. Learning to hash with graph neural networks for recommender systems. In *Proceedings of The Web Conference 2020*, pages 1988–1998, 2020.

[181] Xianfeng Tang, Yozen Liu, Neil Shah, Xiaolin Shi, Prasenjit Mitra, and Suhang Wang. Knowing your fate: Friendship, action and temporal explanations for user engagement prediction on social apps. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2269–2279, San Diego, CA, 2020. ACM.

[182] Xianfeng Tang, Yozen Liu, Xinran He, Suhang Wang, and Neil Shah. Friend story ranking with edge-contextual local graph convolutions. In *Proceedings of the Fifteenth*

ACM International Conference on Web Search and Data Mining, pages 1007–1015, Singapore, 2022. ACM.

[183] Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Mehdi Azabou, Eva L. Dyer, Rémi Munos, Petar Velickovic, and Michal Valko. Large-scale representation learning on graphs via bootstrapping. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, pages 1–18, Virtual, 2022. OpenReview.net. URL `https://openreview.net/forum?id=0UXT6PpRpW`.

[184] Yuandong Tian, Xinlei Chen, and Surya Ganguli. Understanding self-supervised learning dynamics without contrastive pairs, 2021. URL `https://arxiv.org/abs/2102.06810`.

[185] Manasi Vartak, Arvind Thiagarajan, Conrado Miranda, Jeshua Bratman, and Hugo Larochelle. A meta-learning perspective on cold-start recommendations for items. *Advances in neural information processing systems*, 30, 2017.

[186] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax, 2018. URL `https://arxiv.org/abs/1809.10341`.

[187] Maksims Volkovs, Guangwei Yu, and Tomi Poutanen. Dropoutnet: Addressing cold start in recommender systems. *Advances in neural information processing systems*, 30, 2017.

[188] Chenyang Wang, Yuanqing Yu, Weizhi Ma, Min Zhang, Chong Chen, Yiqun Liu, and Shaoping Ma. Towards representation alignment and uniformity in collaborative

filtering. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1816–1825, 2022.

[189] Fei Wang, Hector-Hugo Franco-Penya, John D Kelleher, John Pugh, and Robert Ross. An analysis of the application of simplified silhouette to the evaluation of k-means clustering validity. In *Machine Learning and Data Mining in Pattern Recognition: 13th International Conference, MLDM 2017, New York, NY, USA, July 15-20, 2017, Proceedings 13*, pages 291–305, New York, NY, USA, 2017. Springer, Springer.

[190] Li Wang, Binbin Jin, Zhenya Huang, Hongke Zhao, Defu Lian, Qi Liu, and Enhong Chen. Preference-adaptive meta-learning for cold-start recommendation. In *IJCAI*, pages 1607–1614, 2021.

[191] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*, pages 1–7. 2017.

[192] Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the web conference 2021*, pages 1785–1797, 2021.

[193] Shijie Wang, Guiling Sun, and Yangyang Li. Svd++ recommendation algorithm based on backtracking. *Information*, 11(7):369, 2020.

[194] Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *International Conference on Machine Learning*, pages 9929–9939. PMLR, 2020.

[195] Yilun Wang, Yu Zheng, and Yexiang Xue. Travel time estimation of a path using sparse trajectories. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 25–34, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2956-9. doi: 10.1145/2623330.2623656. URL http://doi.acm.org/10.1145/2623330.2623656.

[196] Yiwei Wang, Bryan Hooi, Yozen Liu, Tong Zhao, Zhichun Guo, and Neil Shah. Flashlight: Scalable link prediction with effective decoders. *arXiv preprint arXiv:2209.10100*, 2022.

[197] Yue Wang and Xintao Wu. Preserving differential privacy in degree-correlation based graph generation. *Transactions on data privacy*, 6:127–145, 08 2013.

[198] Zhitao Wang, Yong Zhou, Litao Hong, Yuanhang Zou, Hanjing Su, and Shouzhi Chen. Pairwise learning for neural link prediction. *CoRR*, abs/2112.02936:1–10, 2021. URL https://arxiv.org/abs/2112.02936.

[199] Xiang Wei, Boqing Gong, Zixia Liu, Wei Lu, and Liqiang Wang. Improving the improved training of wasserstein gans: A consistency term and its dual effect. *arXiv:1803.01541*, 2018.

[200] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 1113–1120, 2009.

[201] Zixin Wen and Yuanzhi Li. The mechanism of prediction head in non-contrastive self-supervised learning. *arXiv preprint arXiv:2205.06226*, 2022.

[202] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pages 478–487, Cambridge, MA, USA, 2016. PMLR, PMLR.

[203] Dongkuan Xu and Yingjie Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2:165–193, 2015.

[204] Bo Yang, Xiao Fu, Nicholas D Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *international conference on machine learning*, pages 3861–3870, Cambridge, MA, USA, 2017. PMLR, PMLR.

[205] Liangwei Yang, Zhiwei Liu, Chen Wang, Mingdai Yang, Xiaolong Liu, Jing Ma, and Philip S Yu. Graph-based alignment and uniformity for recommendation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 4395–4399, 2023.

[206] Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications. *ACM Computing Surveys*, 56(4):1–39, 2023.

[207] Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. Understanding negative sampling in graph representation learning, 2020. URL `https://arxiv.org/abs/2005.09863`.

[208] Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. Understanding negative sampling in graph representation learning. In *Proceedings*

of the 26th ACM SIGKDD international conference on knowledge discovery & data mining, pages 1666–1676, New York, NY, USA, 2020. ACM.

[209] Haoteng Yin, Muhan Zhang, Yanbang Wang, Jianguo Wang, and Pan Li. Algorithm and system co-design for efficient subgraph-based graph representation learning. *Proceedings of the VLDB Endowment*, 15(11):2788–2796, 2022.

[210] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983, London, UK, jul 2018. ACM. doi: 10.1145/3219819.3219890. URL `https://doi.org/10.1145%2F3219819.3219890`.

[211] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983, 2018.

[212] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical Graph Representation Learning with Differentiable Pooling. *Advances in Neural Information Processing Systems*, 2018-December:4800–4810, 6 2018. URL `http://arxiv.org/abs/1806.08804`.

[213] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International conference on machine learning*, pages 5708–5717, Cambridge, MA, 2018. PMLR, JMLR.org.

[214] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In Jennifer G. Dy and Andreas Krause, editors, *ICML 2018*, volume 80, pages 5694–5703, Stockholm, Sweden, 2018. PMLR. URL `http://proceedings.mlr.press/v80/you18a.html`.

[215] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33:5812–5823, 2020.

[216] Chuanming Yu, Xiaoli Zhao, Lu An, and Xia Lin. Similarity-based link prediction in social networks: A path and node combined approach. *Journal of Information Science*, 43(5):683–695, 2017.

[217] Junliang Yu, Xin Xia, Tong Chen, Lizhen Cui, Nguyen Quoc Viet Hung, and Hongzhi Yin. Xsimgcl: Towards extremely simple graph contrastive learning for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[218] Ahmad Zareie and Rizos Sakellariou. Similarity-based link prediction in social networks using latent relationships between the users. *Scientific Reports*, 10(1):1–11, 2020.

[219] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction, 2021. URL `https://arxiv.org/abs/2103.03230`.

[220] Caojin Zhang, Yicun Liu, Yuanpu Xie, Sofia Ira Ktena, Alykhan Tejani, Akshay Gupta, Pranay Kumar Myana, Deepak Dilipkumar, Suvadip Paul, Ikuhiro Ihara, et al. Model size reduction using frequency based double hashing for recommender systems. In

Proceedings of the 14th ACM Conference on Recommender Systems, pages 521–526, 2020.

[221] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. Discrete collaborative filtering. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 325–334, 2016.

[222] Hengrui Zhang, Qitian Wu, Junchi Yan, David Wipf, and Philip S Yu. From canonical correlation analysis to self-supervised graph neural networks. *Advances in Neural Information Processing Systems*, 34:76–89, 2021.

[223] Kunpeng Zhang, Shaokun Fan, and Harry Jiannan Wang. An efficient recommender system using locality sensitive hashing. 2018.

[224] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5165–5175, New York, NY, USA, 2018. Curran Associates.

[225] Muhan Zhang and Yixin Chen. Inductive matrix completion based on graph neural networks. *arXiv preprint arXiv:1904.12058*, 2019.

[226] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *CoRR*, abs/1812.04202:1–24, 2018. URL `http://arxiv.org/abs/1812.04202`.

[227] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. Data augmentation for graph neural networks. In *Proceedings of the AAAI Conference*

on Artificial Intelligence, volume 35, pages 11015–11023, Cambridge, MA, 2021. AAAI Press.

[228] Tong Zhao, Wei Jin, Yozen Liu, Yingheng Wang, Gang Liu, Stephan Günneman, Neil Shah, and Meng Jiang. Graph data augmentation for graph machine learning: A survey. *arXiv preprint arXiv:2202.08871*, 2022.

[229] Tong Zhao, Gang Liu, Stephan Günnemann, and Meng Jiang. Graph data augmentation for graph machine learning: A survey, 2022. URL `https://arxiv.org/abs/2202.08871`.

[230] Tong Zhao, Gang Liu, Daheng Wang, Wenhao Yu, and Meng Jiang. Learning from counterfactual links for link prediction. In *International Conference on Machine Learning*, pages 26911–26926, Cambridge, MA, 2022. PMLR, PMLR.

[231] Tong Zhao, Xianfeng Tang, Danqing Zhang, Haoming Jiang, Nikhil Rao, Yiwei Song, Pallav Agrawal, Karthik Subbian, Bing Yin, and Meng Jiang. Autogda: Automated graph data augmentation for node classification. In *The First Learning on Graphs Conference*, pages 1–17, Cambridge, MA, 2022. PMLR.

[232] Wayne Xin Zhao, Shanlei Mu, Yupeng Hou, Zihan Lin, Yushuo Chen, Xingyu Pan, Kaiyuan Li, Yujie Lu, Hui Wang, Changxin Tian, et al. Recbole: Towards a unified, comprehensive and efficient framework for recommendation algorithms. In *proceedings of the 30th acm international conference on information & knowledge management*, pages 4653–4664, 2021.

[233] Wayne Xin Zhao, Yupeng Hou, Xingyu Pan, Chen Yang, Zeyu Zhang, Zihan Lin, Jingsen Zhang, Shuqing Bian, Jiakai Tang, Wenqi Sun, et al. Recbole 2.0: towards a more up-to-date recommendation library. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 4722–4726, 2022.

[234] Wenqing Zheng, Edward W Huang, Nikhil Rao, Sumeet Katariya, Zhangyang Wang, and Karthik Subbian. Cold brew: Distilling graph node representations with incomplete or missing neighborhoods. *arXiv preprint arXiv:2111.04840*, 2021.

[235] Dawei Zhou, Lecheng Zheng, Jiejun Xu, and Jingrui He. Misc-gan: A multi-scale generative model for graphs. *Frontiers in Big Data*, 2:3, 2019. ISSN 2624-909X. doi: 10.3389/fdata.2019.00003. URL `https://www.frontiersin.org/article/10.3389/fdata.2019.00003`.

[236] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep graph contrastive representation learning. *CoRR*, abs/2006.04131:1–17, 2020. URL `https://arxiv.org/abs/2006.04131`.

[237] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*, pages 2069–2080, New York, NY, USA, 2021. ACM.

[238] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, 2017.