NETWORK BIOLOGY POWERED BY GRAPH DEEP LEARNING: UNDERSTANDING THE MOLECULAR BASIS OF HUMAN GENETICS

By

Renming Liu

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computational Mathematics, Science & Engineering—Doctor of Philosophy

2024

ABSTRACT

Graph learning is revolutionizing the 25-year-old field of network biology by advancing our ability to gain novel molecular insights into biological functions and complex diseases using genome-scale molecular interaction networks. Earlier developed computational network biology methods are limited by their accuracy, scalability, and biological context-specificity. This thesis develops several effective and efficient graph learning methods to further our ability to uncover the multifaceted, complex, and context-specific roles of human genes.

In living cells, biochemical and biomolecular entities, like protein, RNA, and DNA, intricately interact to exert and regulate biological functions and express molecular traits. This interconnected nature renders the network biology perspective, which models the intermolecular relationships as a graph, quintessential in understanding genes' biological roles. As a newly developed field rapidly evolving due to its versatility and broad applicability, graph learning aims to build powerful models that unravel complex interaction patterns to accurately classify node labels, opening exciting opportunities to understand human genetics.

This thesis contributes toward a deeper understanding of human genetics using gene interaction networks by developing algorithms, methods, applications, benchmarks, and software. Specifically, through a comprehensive study using various biological networks and gene classification tasks, I first demonstrate that building machine learning models using the network adjacency matrix systematically achieves more accurate gene classifications compared to traditional network propagation-based approaches. Despite the superior performance, the proposed approach does not scale well to large and dense networks, hindering its applicability. To resolve this limitation, I next develop several effective and efficient graph representation learning methods and software by carefully considering the data characteristics of biological networks and the context-specific nature of biological systems. Finally, I consolidate a comprehensive resource for graph deep learning-based gene classification, allowing future developments to be easily built on top of the work done in this thesis.

To my parents for their unwavering love, support, and belief in me,
without which my achievements would linger as mere shadows of dreams.

ACKNOWLEDGEMENTS

I am deeply grateful to my advisor, Dr. Arjun Krishnan, for his immense support and mentorship throughout my Ph.D. journey, training me to think critically and articulate my scientific research clearly, unconditionally supporting all of my curiosity and passion, and always believing in me. Arjun taught me to think like a scientist, an invaluable skill that has profoundly shaped my research approach. He introduced and guided me to the fascinating world of machine learning, igniting a passion that became the cornerstone of my dissertation. Arjun has offered extraordinary guidance during all of my critical milestones, such as preparing for conference talks, the comprehensive exam, the dissertation defense, and the job search process. I am genuinely grateful for the numerous insightful discussions we have had, during which he generously shared his experience and advice. These conversations have been instrumental in shaping my professional development across various stages. I am also extremely thankful to Dr. Matthew J. Hirn (Matt), who co-advised me with Arjun for the first three years of my Ph.D. and taught me the solid mathematical foundations of modern deep learning. Matt always explains the intimidatingly complicated mathematical terms and theories behind deep learning in the most straightforward and intuitively understandable manner. Learning these theories from Matt was a joyful experience, as weird as it may sound. His way of mathematical thinking has forever influenced mine. The gratitude I feel towards both Arjun and Matt is beyond words. Their dedication and steadfast support have been pivotal in my Ph.D. journey. I feel extraordinarily fortunate to have had them as my mentors, and I know I would not be where I am today without their guidance and encouragement.

I want to express my heartfelt appreciation to my Ph.D. committee members, Dr. Christina Chan, Dr. Jiliang Tang (JT), and Dr. Yuying Xie, for their insightful and constructive suggestions that significantly shaped my dissertation. I am especially thankful to JT for his extraordinary support during the crucial final stages of my Ph.D. by providing thoughtful feedback to my preparations for my job search and dissertation defense. I am grateful to be additionally trained by JT from a computer scientist's perspective. JT's guidance has broadened my scientific perspective and skills, leaving a lasting impact on my professional development. I also want to thank Yuying for guiding

me and introducing me to the exciting fields of multi-omics and spatial single-cell analysis. I feel deeply blessed to have been served by such dedicated and brilliant minds.

I am incredibly fortunate to have been a part of several supportive and inclusive labs, including Arjun's Krishnan Lab, Matt's CEDAR Team, and JT's DSE Lab, all of which have provided healthy working environments to develop my research and skills. I am greatly thankful to every colleague in each of these labs. I am especially grateful for the help from and discussions with Dr. Christopher A. Mancuso (Chris) and Dr. Kayla A. Johnson from the Krishnan Lab. Chris has been a fantastic mentor and has provided me tremendous hands-on guidance to kick start my research at the very beginning of my Ph.D., and has been continually doing so ever since. Kayla has been unwaveringly supportive of everyone in the lab, going out of her way to assist anyone in need and offering help wherever she can. I am incredibly thankful for her responsive feedback on my writing countless times. I am also grateful for a lot of enriching and inspiring discussions about cell biology, biomedical technologies, and human complex diseases with Kayla, Dr. Stephanie L. Hickey, Alex McKim, and Hao Yuan from the Krishnan Lab. In addition, I would also like to thank Chris, Kayla, Hao, Anna Yannakopoulos, and Keenan Manpearl in the Krishnan Lab for contributing to the exciting projects we worked on. I am deeply thankful to all my colleagues for their help and support, which have greatly enhanced my personal and professional growth.

I am immensely grateful to have the privilege to work with many brilliant collaborators. It has been a great pleasure working on various projects related to graph deep learning with folks from Mila, particularly Semih Cantürk, Dr. Ladislav Rampášek, Dr. Dominique Beaini, and Dr. Guy Wolf. I was extremely blessed to be able to work closely with Ladislav and Semih on cutting-edge research problems in graph learning. I also want to thank my collaborators in the DANCE team led by JT, including Wenzhuo Tang, Jiayuan Ding, Hongzhi Wen, and Xinnan Dai, for working hard together on pioneering works of deep learning methods for single-cell analysis. Working with such talented people has been an amazing experience, and I look forward to continually collaborating in the coming years.

I want to thank each and every one of my friends for being there for me, supporting me, and

sharing countless unforgettable moments. Thank you, Xin Gu, Yi Duan, Qiuyuan Gao, Yucong Qin, and Liyi Chen, for sharing an incredible part of my life when I was young and dedicated to music in a Choir and a Symphony orchestra. I cannot be more grateful to have met and become close friends with Zhengyi Lian, Ziyang Li, Qing He, Haoming Zheng, Youqing Chen, Zhaoming Li, Zhenbo Fang, Shiqi Chen, and Yanchao Pan. Each of you has enriched my life immeasurably. Lastly, I would like to thank all my closest friends during my Ph.D. that have been invaluable to me, including Jiaxin Yang, Tianyu Yang, Wenzhuo Tang, Kaiqi Yang, Hanbing Wang, Yaxin Li, Haitao Mao, and Kai Guo. I am truly blessed to have known all my friends in my life.

Finally, I must extend my deepest gratitude to my Mom, Dad, and the rest of my family for their unconditional love, unwavering support, and steadfast belief in me. I am especially grateful to my Mom, who has wholeheartedly supported every decision I have made and consistently went above and beyond to help me whenever needed. The depth of appreciation and indebtedness I feel towards my parents is immeasurable and light-years beyond what words can adequately express. Thank you, Mom, Dad, and all my family members, for everything you have done for me. You have truly shaped the person I am today.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Background

Grasping the precise functional roles of human genes and the context in which they operate is pivotal for uncovering the molecular basis of human genetics. This understanding not only facilitates the connection between genotype and phenotype [109] but also is instrumental in developing treatment strategies for complex diseases [19, 261]. Despite the recent landmark achievement of the complete, telomere-to-telomere assembly of the human genome in 2022 [186], our knowledge about every human gene remains greatly incomplete, as experimentally annotating and validating gene functions is exceedingly time-consuming and resource-intensive [130, 11]. As of January 2024, only about half of the 20K human protein-coding genes are functionally annotated with high confidence[1]. This knowledge gap is further exacerbated by the fact that the functional roles of genes and proteins inherently pertain to their biological contexts, such as specific tissues and cell types [93, 80]. Developing accurate predictive models to annotate genes' biological roles and making these resources and tools easily accessible to biomedical researchers are vital to close this gap.

Learning about genes through the lens of network biology Biological systems are astonishingly complex. Different biological entities, such as protein, RNA, and DNA intricately interact to carry out specific functions or traits. At the cellular level, collections of proteins and RNA bind together to form functional units, such as the pre-initiation complex and ribosomes, that carry out vital processes. At the genomic level, regulatory elements, like promoters and enhancers, interact with transcription factors and other regulatory proteins to precisely control the expression of one or several genes. Cascading these gene regulatory circuits leads to highly complex mechanisms underlying cellular differentiation [214], cell fate determination [233, 134], and diseases [168]. *Network biology* is a 25-year-old field that aims to understand genetics using these diverse and intricate interactions,

[1] Based on gene ontology annotations (http://release.geneontology.org/2023-11-15/annotations/goa_human.gaf.gz) for GOBPs with EXP or HTP evidence codes

which are formulated into molecular interaction networks [298, 19, 21, 241, 261]. In these networks, or more commonly referred to as *graphs* in mathematical terms, nodes represent genes or proteins and edges represent the functional association between the two nodes.

Since the introduction of network biology, much progress has been made toward deepening our understanding of genes through analyzing the *structural* and *proximal* characteristics of molecular interaction networks. On the one hand, these networks provide rich structural prior that correlates with genes' functional roles. For example, essential genes tend to have a larger number of interacting partners compared to non-essential genes [77, 110]. One reasonable hypothesis is that essential genes are more evolutionarily conservative and that newly emerged genes are more likely to interact with high degree genes [22]. Furthermore, cancerous genes usually have higher connectivity than non-cancerous genes [114] and the "bottleneckness" is a good indicator of the gene essentiality [289]. On the other hand, the notion of *proximity* in these networks characterizes the relationship between groups of genes. Genes or proteins participating in the same biological process or pathway tend to co-express or physically interact [19, 148]. Similarly, genes associated with a particular disease are more likely to reside in a distinct neighborhood of the molecular interaction network than random. These neighborhoods are commonly referred to as a "disease module" and have shed light on the intricate interplay between cellular processes and diseases [19]. Moreover, the proximity between two disease modules in the network indicates their phenotypical and symptom similarity [172]. These proximal characteristics have thus given rise to numerous early studies implementing the *guilt-by-association* (GBA) principle [188] via label propagation [126, 51]. Applications of these approaches have shown remarkable success in identifying cancerous pathways [138], annotating disease genes [253, 126], reprioritizing GWAS hits [136], and predicting drug-binding [285].

Graph learning offers immense opportunities for learning on biological networks Graph learning aims to build numerical representations and predictive models using graphs as the computational backbone [88, 34, 33]. These methods have demonstrated effectiveness in solving various computational problems where the data can be naturally formulated as graphs, such as predicting the chemical properties of a molecule, categorizing papers in a citation network, and

identifying friendships in a social network [295, 279]. The fundamental principle behind these graph learning approaches is that the graph structure and connections are closely linked to the characteristics of the graphs or their nodes. Graph learning methods fall into two main categories: *embeddings*, which focus on translating graph structure into vector space representations, and *graph neural networks* (GNNs), which extend neural network models to operate on graphs directly. Both approaches play pivotal roles in extracting meaningful insights from complex graph-structured data.

The early development of network embedding was led by *DeepWalk* [202] in 2014, which extends the seminal work in natural language processing, *word2vec* [174, 173], to networks. *DeepWalk* operates by generating random walks on the network and treating the walk sequences as a "corpus" in text. The subsequent *word2vec* algorithm then optimizes the resulting low-dimensional space based on this corpus. Following *DeepWalk*, many other random walk based network embedding methods were introduced [83, 61, 217, 2, 182]. *node2vec* [83] is a notable extension of *DeepWalk*. It performs second-order random walks that can search over the graph more flexibly, resembling the Breadth-First Search (BFS) or the Depth-First Search (DFS) strategies. Such searching flexibility has shown practical advantages in downstream prediction tasks. Since its introduction in 2016, *node2vec* has been particularly popular in computational biology [183, 291] and has been used for essential protein prediction [292, 266] and various disease gene prediction tasks [15, 16, 161, 286, 200].

In 2013, Bruna and coworkers [36] pioneered the work on GNN by drawing parallels between convolution in the Euclidean space, as grids, and the graph Laplacian. This spectral formulation is effective but also computationally expensive due to the full eigen-decomposition of the graph Laplacian. Defferrard and colleagues [57] later proposed an approximation technique via Chebyshev polynomials of the graph Laplacian to greatly reduce the filter parameters. In 2016, Kipf and Welling [124] further simplified this via a first-order approximation, leading to a simple and yet effective graph convolution module. Additionally, this formulation has a spatial operational interpretation of *aggregating* neighboring nodes' representations followed by a *transformation*. This two steps recipe was later generalized by Gilmer and coworkers [74], formalizing the Message Passing Neural Network (MPNN). To date, many popular GNN architectures such as GraphSAGE [89],

GAT [257], GIN [282], and many more [74, 295, 279], fall into the framework of MPNN. GNNs have been applied to solve various real-world problems due to their exceptional performance [295, 279].

This advent of graph learning techniques presents a transformative potential to network biology, opening doors to novel discoveries and completing our knowledge about human genes by building accurate predictive models on the underlying molecular interaction networks. Despite these opportunities, tremendous gaps remain in effectively and efficiently applying these graph learning paradigms to elucidate genes' biological roles: It is unclear (1) how to *formulate* the problem of gene function and disease annotation into graph learning tasks, (2) whether applying these advanced methods will achieve *superior results* when compared to traditional label propagation methods in network biology given the scarce labels and unique characteristics of biological networks, and (3) if graph learning approaches are *scalable* enough to be applied to genome-scale molecular interaction networks. These considerations highlight the need for further research and refinement in applying graph learning techniques to advance our understanding of human genes.

1.2 Dissertation contributions

This dissertation bridges the gap between advanced graph learning approaches and understanding human genetics through network biology, ultimately leading to a deeper understanding of key genetic factors influencing human health and diseases. We present comprehensive studies and in-depth investigations seeking optimal ways to predict genes' biological roles using genome-scale gene interaction networks. Across all projects, we make the code and data publicly available to ensure easy reproducibility and adaptation of our work. When applicable, we package the developed methods or resources into Python packages and register them on PyPI, further enabling our works to be applied to broader areas in biomedical research. We summarize the primary contributions of this dissertation as follows.

- Traditional network biology leverages label propagation to expand gene annotations by diffusing the relatively limited known annotations for a particular function, disease, or trait over the gene interaction network. More recently, some studies have presented ideas for

supervised learning using the network adjacency matrix as the feature [128]. However, it was unclear whether one approach is systematically better than the other and whether supervised learning can leverage the underlying network structure as label propagation naturally does. In GenePlexus [153] (Chapter 2), we demonstrate conclusive evidence, using extensive gene classification tasks and gene interaction networks, that supervised learning consistently outperforms label propagation. Furthermore, it does so by accounting for the underlying network connectivity. Finally, we highlight the opportunities and the need for further developing effective and efficient network embedding methods to enhance supervised learning on gene interaction networks subsequently.

- Despite the promising opportunities of using network embedding, particularly *node2vec* [83], to build efficient supervised learning models for gene classification, significant computational burdens remain. Specifically, the original *node2vec* implementations failed to embed existing genome-wide interaction networks with dense connections. In Chapter 3, we develop a highly optimized *node2vec* implementation, called PecanPy [151], that enables *node2vec* embedding large scale (> 100K nodes) and dense (> 100M edges) gene interaction networks. Remarkably, through benchmarking on networks with diverse sizes and densities, we show that PecanPy runs up to an order of magnitude faster with an order of magnitude less memory usage than the original *node2vec* implementations. These improvements are achieved by meticulously optimizing the graph data structure, parallelism, and random walk strategies. We package PecanPy into a user-friendly Python software, which can be easily installed using the standard Python Package Index (PyPI) repository.

- Upon comprehensive evaluation of *node2vec* embeddings using our PecanPy implementations on diverse genome-wide gene interaction networks, we identify a critical shortcoming of *node2vec* in handling weighted graphs, especially when they are dense. Resolving this limitation is critical for network biology as many existing gene interaction networks are dense and weighted, either by construction or by integrating gene interaction evidence from multiple

sources. In Chapter 4, we propose a natural extension of *node2vec*, called *node2vec+* [150], which adequately accounts for edge weights in the weighted graph. We demonstrate through systematic evaluations using synthetic and real-world network data, including several densely weighted gene interaction networks, that *node2vec+* significantly outperforms *node2vec*, surpassing the more powerful graph neural network methods.

- Our PecanPy and *node2vec+* enable fast, efficient, and effective embedding of a dense and weighted genome-scale gene interaction network to predict gene-disease associations. However, the exact cellular roles of a gene or protein depend highly on its precise biological and cellular conditions. This biological context-specificity motivates the development of a gene interaction network embedding method that can induce biological contexts, such as tissue, cell types, and diseases. In Chapter 5, we bridge this gap by presenting our contextualized graph attention model CONE [155] for learning biological context-aware gene embeddings. CONE is a versatile approach that can contextualize the embeddings of any gene interaction network given biological contexts in the form of gene sets, such as tissue-specific genes. Through various disease gene classification and therapeutic target prediction tasks, we show that CONE (1) successfully leverages biological contextual information to identify diseases-related genes and therapeutic targets and (2) sheds light on the relevant biological contexts for particular diseases.

- Previous chapters have established the foundation for unveiling genes' precise biological roles using genome-scale gene interaction networks. The core technical components enabling this are the powerful graph deep learning (GDL) techniques for representing and learning on graphs. Similar to network biology, the field of GDL is evolving rapidly, with more powerful and expressive models being developed every year. This fast-paced development of GDL necessitates a comprehensive resource for the streamlined adaptation of the work developed in previous chapters to continually improve network-based gene annotation applications by using more *up-to-date* architectures and designing *specialized* architectures that take advantage of

the unique data characteristics of biological networks, such as density (Chapter 4) and context-specificity (Chapter 5). In Chapter 6, we present such an extensive resource as a benchmarking dataset Python package, `obnb` [152], along with a systematic benchmarking study using current state-of-the-art GDL methods, spanning graph diffusion, graph embedding, and graph neural networks. `obnb` is designed to be compatible with the two most popular graph deep learning frameworks in Python, namely, PyTorch Geometric (https://www.pyg.org/) and Deep Graph Library (https://www.dgl.ai/), making it straightforward for the GDL community to further develop on top of it. Our analyses (1) reveal that the efficacy of GDL methods is tied to the *corrected homophily ratio* given the gene interaction network, and (2) point to promising future directions, such as integrating structural or sequential information from pre-trained genomics and protein foundation models.

1.3 Dissertation structure

We organize the rest of this dissertation as follows. Chapter 2 formally sets up the network-based gene classification problem and rigorous evaluation framework, showing that supervised learning systematically outperforms traditional label propagation-based approaches and opening up opportunities for supervised learning with learned network embeddings. Chapter 3 and Chapter 4 then dive into technical contributions that significantly improve (1) the computational efficiency and (2) the quality of network embeddings for gene classification. Chapter 5 brings the capability of gene interaction network embeddings one step further by developing a versatile approach to induce biological context-specificity. Chapter 6 organizes a comprehensive resource for network-based gene classification benchmarking, paving the way for future development of novel and biologically inspired graph deep learning architectures for gene classifications. Finally, we conclude this dissertation and discuss broader impact with promising future directions in Chapter 7.

CHAPTER 2

SUPERVISED LEARNING IS AN ACCURATE METHOD FOR NETWORK-BASED GENE CLASSIFICATION

Assigning every human gene to specific functions, diseases, and traits is a grand challenge in modern genetics. Computational methods leveraging molecular interaction networks, such as supervised learning and label propagation, are vital to addressing this challenge. Despite being a popular machine learning technique across fields, supervised learning has been applied only in a few network-based studies for predicting pathway-, phenotype-, or disease-associated genes. It is unknown how supervised learning broadly performs across different networks and diverse gene classification tasks and how it compares to label propagation, the widely benchmarked canonical approach for this problem. Here, we present a comprehensive benchmarking study of supervised learning for network-based gene classification. We use stringent evaluation schemes to evaluate this approach and the classic label propagation techniques on hundreds of diverse prediction tasks and multiple networks. Our results demonstrate that supervised learning on a gene's full network connectivity outperforms label propagation and achieves high prediction accuracy by efficiently capturing local network properties, rivaling label propagation's appeal for naturally using network topology. We further show that supervised learning on the full network is superior to learning on node embeddings, an increasingly popular approach for concisely representing network connectivity. These results show that supervised learning is an accurate approach for prioritizing genes associated with diverse functions, diseases, and traits and should be considered a staple of network-based gene classification workflows. The code to reproduce the experiments and analyses in this study is available on GitHub: https://github.com/krishnanlab/GenePlexus.

2.1 Introduction

In the post-genomic era, a grand challenge is to characterize every gene across the genome in terms of the cellular pathways they participate in, and which multifactorial traits and diseases they are associated with. Computationally predicting the association of genes to pathways, traits, or diseases – the task termed here as *gene classification* – has been critical to this quest, helping

prioritize candidates for experimental verification and for shedding light on poorly characterized genes [231, 198, 213, 224, 207, 112] to the success of these methods has been the steady accumulation of large amounts of publicly available data collections such as curated databases of genes and their various attributes [242, 144, 119, 206, 37, 276, 281, 38], controlled vocabularies of biological terms organized into ontologies [49, 14, 229, 237], high-throughput functional genomic assays [67, 137, 17], and molecular interaction networks [245, 142, 240, 80, 100].

While protein sequence and 3D structure are remarkably informative about the corresponding gene's molecular function [10, 273, 213, 112], the pathways or phenotypes that a gene might participate in significantly depends on the other genes that it works with in a context-dependent manner. Molecular interaction networks – graphs with genes or proteins as nodes and their physical or functional relationships as edges – are powerful models for capturing the functional neighborhood of genes on a whole-genome scale [256, 120]. These networks are often constructed by aggregating multiple sources of information about gene interactions in a context-specific manner [108, 80]. Therefore, unsurprisingly, several studies have taken advantage of these graphs to perform network-based gene classification [272, 126, 253].

The canonical principle of network-based gene classification is *guilt-by-association*, the notion that proteins and genes that are strongly connected in the network are likely to perform the same functions and, hence, participate in similar higher-level attributes such as phenotypes and diseases [268]. Instead of solely aggregating *local* information from direct neighbors [230], this principle is better realized by propagating pathway or disease labels across the network to capture *global* patterns, achieving state-of-the-art results [272, 180, 253, 126, 51, 138, 293, 190, 256]. These global approaches belong to a class of methods referred to here as *label propagation*. Distinct from label propagation is another class of methods for gene classification that relies on the idea that network patterns characteristic of genes associated with a specific phenotype or pathway can be captured using supervised machine learning [24, 133, 80, 129, 84, 192]. While this class of methods – referred to here as *supervised learning* – has yielded promising results in several applications, how it broadly performs across different types of networks and diverse gene classification tasks is unknown.

Consequently, supervised learning is used far less than label propagation for network-based gene classification.

This study aims to perform a comprehensive, systematic benchmarking of supervised learning (SL) for network-based gene classification across several genome-wide molecular networks and hundreds of diverse prediction tasks using meaningful evaluation schemes. Within this rigorous framework, we compare supervised learning to a widely-used, classic label propagation (LP) technique, testing both the original (**A**djacency matrix) and a diffusion-based representation of the network (**I**nfluence matrix). This combination results in four methods (listed with their earliest known references): label propagation on the adjacency matrix (LP-A) [230], label propagation on the influence matrix (LP-I) [190], supervised learning on the adjacency matrix (SL-A) [24], and supervised learning on the influence matrix (SL-I) [133]. Additionally, we evaluate the performance of supervised learning using node embeddings as features, as the use of node embeddings is burgeoning in network biology.

Our results demonstrate that SL outperforms LP for gene-function, -disease, and -trait prediction. We also observe that SL captures local network properties as efficiently as LP, where both methods achieve more accurate predictions for gene sets that are more tightly clustered in the network. Lastly, we show that SL using the full network connectivity is superior to using low-dimensional node embeddings as features, which, in turn, is competitive to LP.

2.2 Methods

We chose a diverse set of undirected, human gene interaction networks based on criteria laid out in [100] (Figure 2.1): (1) networks constructed using high- or low-throughput data, (2) the type of interactions the network was constructed from, and (3) if annotations were directly incorporated in constructing the network. We used versions of the networks released before 2017 so as not to bias the temporal holdout evaluations. Unless otherwise noted, we used all edge weights, and all networks' nodes were mapped into Entrez genes using the `MyGene.info` database [276, 281]. If the original node ID is mapped to multiple Entrez IDs, we add edges between all possible mappings. The networks used in this study are BioGRID [240], the full STRING network [245], as well as the

subset with just experimental support (referred to as STRING-EXP in this study), InBioMap [142], and the tissue-naïve network from GIANT [80], referred to as GIANT-TN in this study. These networks cover a wide size range, with the number of nodes ranging from 14K to 25K and the number of edges ranging from 141K to 38M. More information on the networks can be found in Appendix A.1.1.

2.2.1 Network representations

We consider three distinct representations of molecular networks: the adjacency matrix, an influence matrix, and node embeddings. We describe each representation in the rest of this section.

Adjacency matrix Let $G = (V, E, w)$ denote an undirected molecular network, where V is the set of vertices (genes), E is the set of edges (associations between genes), and $w : E \to \mathbb{R}$ is the edge weight function that indicates the strengths of the associations. G can be represented as a weighted *adjacency matrix*, $\mathbf{M} \in \mathbb{R}^{|V|\times|V|}$, where

$$\mathbf{M}_{i,j} = \begin{cases} w(v_i, v_j) & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Influence matrix G can also be represented as an *influence matrix*, $\mathbf{F} \in \mathbb{R}^{|V|\times|V|}$, which captures both local and global structure of the network. F is obtained using a random walk with restart transformation kernel [138],

$$\mathbf{F} = \alpha\big(\mathbf{I} - (1-\alpha)\mathbf{W_D}\big)^{-1} \tag{2.2}$$

where, $\alpha \in (0, 1)$ is the restart parameter, $\mathbf{D}$ is the diagonal degree matrix whose diagonal elements are given by $\mathbf{D}_{i,i} = \sum_j \mathbf{M}_i$, and $\mathbf{W_D} = \mathbf{MD}^{-1}$ column normalized adjacency matrix. We use a restart parameter of 0.85 in this study as suggested by a previous evaluation [100], which also shows optimal performance in our experiments (Figure A.1).

Node embeddings G can be additionally transformed into a low-dimensional representation through the process of *node embedding*. In this study we used the *node2vec* algorithm [83], which extends ideas from the word2vec algorithm [174, 173] from natural language processing to graphs. The objective of *node2vec* is to find a low-dimensional representation of the adjacency matrix,

$E \in \mathbb{R}^{|V| \times d}$, where the hidden dimension $d \ll |V|$. This is done by optimizing the following log-probability objective function:

$$\mathcal{L} = \sum_{v \in V} \log \Big(Pr\big(\mathcal{N}_S(v) | \mathbf{E}(v)\big) \Big) \tag{2.3}$$

where $\mathcal{N}_S(v)$ is the network neighborhood of node v generated through a sampling strategy S, and $\mathbf{E}(v) \in \mathbb{R}^d$ is the feature vector of node v. In *node2vec*, the sampling strategy is based on random walks that are controlled using two parameters p and q, in which a high value of q keeps the walk local (a breadth-first search), and a high value of p encourages outward exploration (a depth-first search). The values of p and q were both set to 0.1 for every network in this study. Detailed *node2vec* hyperparameter tuning information can be found in Appendix A.1.2.

2.2.2 Prediction methods

We compared the prediction performance across four specific methods across two classes, label propagation (LP) and supervised learning (SL).

Label Propagation LP methods are the most widely used methods in network-based gene classification and achieve state-of-the-art results [126, 51]. In this study, we considered two LP methods, label propagation on the adjacency matrix (LP-A) and label propagation on the influence matrix (LP-I). First, we constructed a binary vector of ground-truth labels, $x \in \mathbb{R}^{|V| \times 1}$, where $x_i = 1$ if gene i is a positively labeled gene in the training set, and 0 otherwise. In LP-A, we constructed a score vector, $s \in \mathbb{R}^{|V| \times 1}$, denoting the predictions,

$$s = \mathbf{M}x \tag{2.4}$$

Thus, the predicted score for a gene using LP-A is equal to the sum of the weights of the edges between the gene and its direct, positively labeled network neighbors. In LP-I, the score vector, s, is generated using Equation 2.2, except $\mathbf{M}$ is by replaced by $\mathbf{F}$, the influence matrix (Equation 2.2). The prediction performance is evaluated by taking into account of positive and negative examples. In both LP-A and LP-I, only positive examples in the training set are used to calculate the score vector s to reflect how label propagation is typically used in practice [126, 51, 205]. We found

that accounting for negative examples in the propagation step does no change the LP prediction performance and thus use the simpler approach of only propagating positive examples in the main text (Appendix A.2.1).

Supervised Learning Supervised learning (SL) can be used for network-based gene classification by using each gene's network neighborhoods as feature vectors, along with gene labels, in a classification algorithm. Here, we use logistic regression with ℓ_2 regularization as the SL classification algorithm, which is a linear model that aims to minimize the following cost function:

$$\mathcal{L} = \frac{1}{2} w^\top w + C \sum_{i=1}^{n} \log\Big(\exp\big(-y_i(\mathbf{X}_i w + b)\big) + 1\Big) \tag{2.5}$$

where $w \in \mathbb{R}^m$ is the vector of weights for a model with m features, C determines the regularization strength, n the number of examples, y is the ground-truth label, $\mathbf{X} \in \mathbb{R}^{n \times m}$ is the data matrix, and b is the intercept. After training a model using the labeled genes in the training set, the learned model weights are used to classify the genes in the testing set, returning a prediction probability for these genes that is bounded between 0 and 1. The regularization parameter, C, was set to 1.0 for all models in this study. We use `scikit-learn`'s [196] logistic-regression implementation to perform model training and prediction.

In this study, three different network-based gene-level feature vectors are used to train three different SL classifiers: the rows of the adjacency matrix (SL-A), the rows of the influence matrix (SL-I), and the rows of the node embedding matrix (SL-E). Model selection and hyperparameter tuning are described in detail in Appendix A.1.2.

2.2.3 Gene set collections

We curate a number of gene set collections to test predictions on a diverse set of tasks: function, disease, and trait (Figure 2.1). Function prediction is defined as predicting genes associated with biological processes that are part of the Gene Ontology (referred to here as GOBP) [49, 14] obtained from `MyGene.info` [276, 281], and pathways from the Kyoto Encyclopedia of Genes and Genomes [119], referred to as KEGGBP since disease-related pathways were removed from the original KEGG annotations in the Molecular Signatures Database [144, 242]. Disease prediction

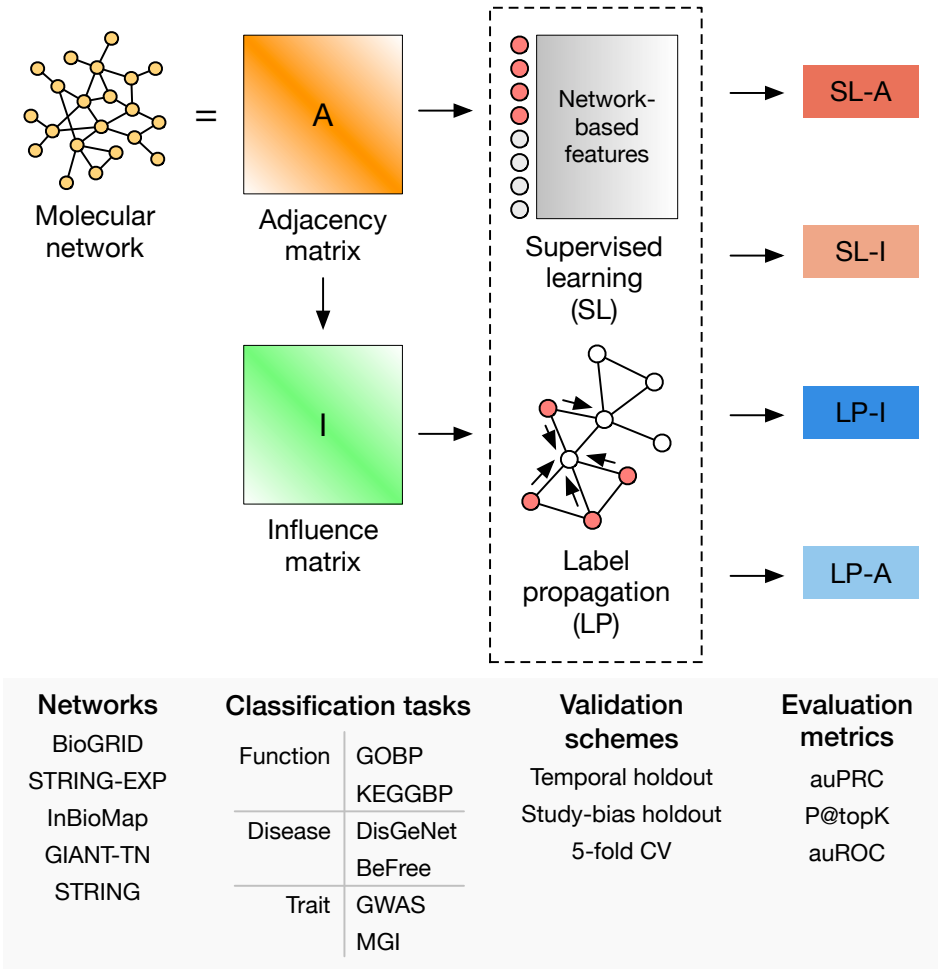


Figure 2.1: **Workflow for gene classification pipeline.** Four methods are compared: supervised learning on the adjacency matrix (SL-A), supervised learning on the influence matrix (SL-I), label propagation on the adjacency matrix (LP-A), and label propagation on the influence matrix (LP-I). Model performance on a variety of gene classification tasks is evaluated over a number of different molecular networks, validation schemes, and evaluation metrics. Additionally, the performance of supervised learning using node embeddings as features (SL-E) is evaluated (not shown in this figure).

is defined based on predicting genes associated with diseases in the DisGeNET database [206]. Annotations from this database were divided into two separate gene set collections: those that were manually curated (referred to as DisGeNet in this study) and those derived using the BeFree text-mining tool (referred to as BeFree in this study). Trait prediction is defined as predicting genes linked to human traits from Genome-wide Association Studies (GWAS), curated from a community challenge [47], and mammalian phenotypes (annotated to human genes) from the Mouse Gene

Informatics (MGI) database [37].

Gene set preprocessing Each of these six gene set collections contains about a hundred to tens of thousands of gene sets that vary widely in specificity and redundancy. Therefore, each collection is preprocessed to ensure that each source's final set of prediction tasks are specific, largely non-overlapping, and not driven by multi-attribute genes. First, if gene sets in a collection corresponded to terms in an ontology (e.g., biological processes in the GOBP collection), annotations were propagated along the ontology structure to obtain a complete set of annotations for all gene sets. Second, we remove gene sets if the number of genes annotated to the gene set is above a certain threshold and then compare these gene sets to each other in order to remove gene sets that are highly overlapping with other gene sets in the collection, resulting in a set of specific, non-redundant gene sets. Finally, individual genes that appear in more than ten of the remaining gene sets in a collection are removed from all the gene sets in that collection to remove multi-attribute (e.g., multi-functional) genes that are potentially easy to predict [73]. More details on the gene set preprocessing and gene set attributes are provided in Appendix A.1.3.

Selecting positive and negative examples In each gene set collection, genes annotated to that set are designated as the set of positive examples for a given gene set. The SL methods additionally require a set of negative genes for each given gene set for training; both SL and LP methods require a set of negative genes for each gene set for testing. A set of negative genes is generated by (a) finding the union of all genes annotated to all gene sets in the collection, (b) removing genes annotated to the given gene set, and c) removing genes annotated to any gene set in the collection which significantly overlapped with the given gene set (p-value < 0.05 based on the one-sided Fisher's exact test).

2.2.4 Validation schemes

We perform extensive and rigorous evaluations based on three validation schemes: temporal holdout, study-bias holdout, and five-fold cross-validation (5FCV). We briefly describe each validation scheme below and defer details to Appendix A.1.4.

Temporal holdout Within a gene set collection, the genes that only had an annotation to any gene set in the collection after January 1st, 2017 are considered test genes, and all other genes

are considered training genes. Temporal holdout is the most stringent evaluation scheme for gene classification as it mimics the practical scenario of using current knowledge to predict the future and is the preferred evaluation method used in the CAFA challenges [213, 112]. Since Gene Ontology is the only source with clear date stamps for all its annotations, temporal holdout only applies to the GOBP gene set collection.

Study-bias holdout For study-bias holdout, genes are first ranked by the number of PubMed articles they were mentioned in, obtained from [35]. The top two-thirds of the most-mentioned genes are considered training genes, and the rest of the least-mentioned genes are used for testing. Study-bias holdout mimics the real-world situation of learning from well-characterized genes to predict novel un(der)-characterized genes.

Five-fold cross validation The last validation scheme is the traditional five-fold cross-validation, where the genes are split into five equal folds in a stratified manner, where the proportion of genes in the positive and negative classes is preserved in each split. In all these schemes, only gene sets with at least ten positive genes in both the training and test sets are considered.

2.2.5 Evaluation Metrics

In this study, we considered three evaluation metrics: the area under the precision-recall curve (auPRC), the precision of the top K-ranked predictions (P@TopK), and, for completeness, the area under the receiver-operator curve (auROC). For P@TopK, we set K equal to the number of positives in the testing set. Since the standard auPRC and P@TopK scores are influenced by the *prior* of finding a positive example, which is computed as the proportion of positives to the total of positives and negatives, we express both metrics as the log_2 fold change of the original metric to the prior. More details on the evaluation metrics are described in Appendix A.1.5.

2.3 Results and discussions

We systematically compare the performance of four gene classification methods (Figure 2.1): supervised learning on the adjacency matrix (SL-A), supervised learning on the influence matrix (SL-I), label propagation on the adjacency matrix (LP-A), and label propagation on the influence

matrix (LP-I). We choose six gene set collections that represent three prominent gene-classification tasks: gene-function (GOBP, KEGGBP), gene-disease (DisGeNet, BeFree), and gene-trait (GWAS, MGI) prediction. We use three different validation schemes: temporal holdout (train on genes annotated before 2017 and test on genes annotated in 2017 or later; only done for GOBP as it has clear timestamps), holdout based on study-bias (train on well-studied genes and predict on less-studied genes), and the traditional five-fold cross-validation (5FCV). Temporal holdout and study-bias holdout validation schemes are presented in the main text as they are more stringent and reflective of real-world tasks compared to 5FCV [116].

To ascertain the robustness of the relative performance of the methods to the underlying network, we choose five different genome-scale molecular networks that differ in their content and construction (Appendix A.1.1). To be in concert with temporal holdout evaluation and curtail data leakage, all the networks used throughout this study are the latest versions released before 2017. We present evaluation results based on the area under the precision-recall curve (auPRC) in the main text and results based on the precision at top-k (P@topK) and the area under the ROC curve (auROC) in Appendix A.2, all of which lead to a consistent conclusion in our study.

2.3.1 SL consistently outperforms LP across diverse gene classification tasks and gene interaction networks

SL methods rank higher than LP methods on average Our first analysis directly compares all four prediction methods against each other for each gene set in a given collection. For each gene set collection–network combination, we rank the four methods per gene set (based on auPRC) using the standard competition ranking and calculate each method's average rank across all the gene sets in the collection (Figure 2.2). For function prediction, SL-A is the top-performing method by a wide margin (particularly clear based on GOBP temporal holdout), with SL-I being the second-best method. For disease and trait prediction, SL-A and SL-I still outperform LP-I, but to a lesser extent. In all cases, LP-A is the worst-performing method. The large performance difference between the SL and LP methods in the GOBP temporal holdout validation is noteworthy since temporal holdout is the most stringent validation scheme and the one employed in community challenges such as

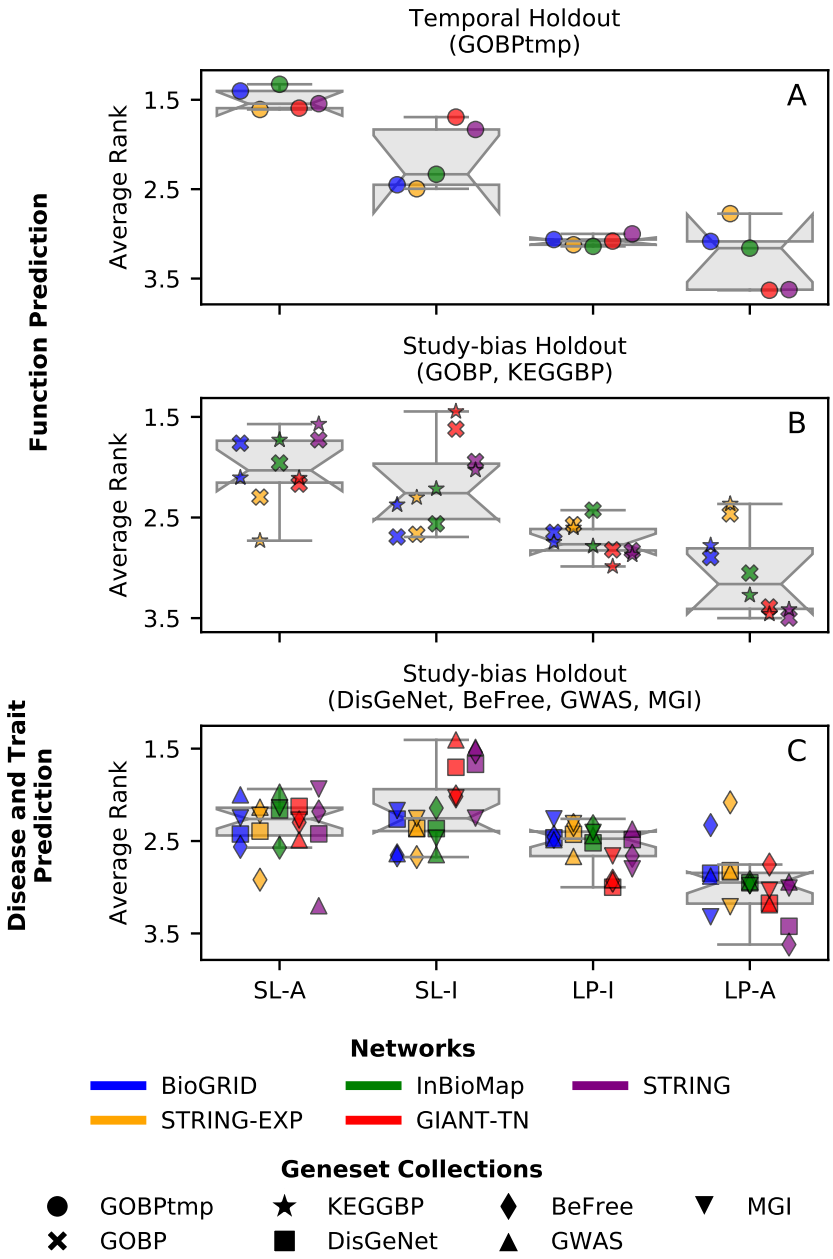


Figure 2.2: **Average rank across the four methods.** Each point in each boxplot represents the average rank for a gene set collection–network combination, obtained based on ranking the four methods in terms of performance for each gene set in a gene set collection using the standard competition ranking. (A) Functional prediction tasks using GOBP temporal holdout, (B) Functional prediction tasks using study-bias holdout for GOBP and KEGGBP, and (C) Disease and trait prediction tasks using study-bias holdout for DisGeNet, BeFree, GWAS, and MGI. The results are shown for auPRC where different colors represent different networks and different marker styles represent the different gene set collections. SL methods outperform LP methods for all prediction tasks.

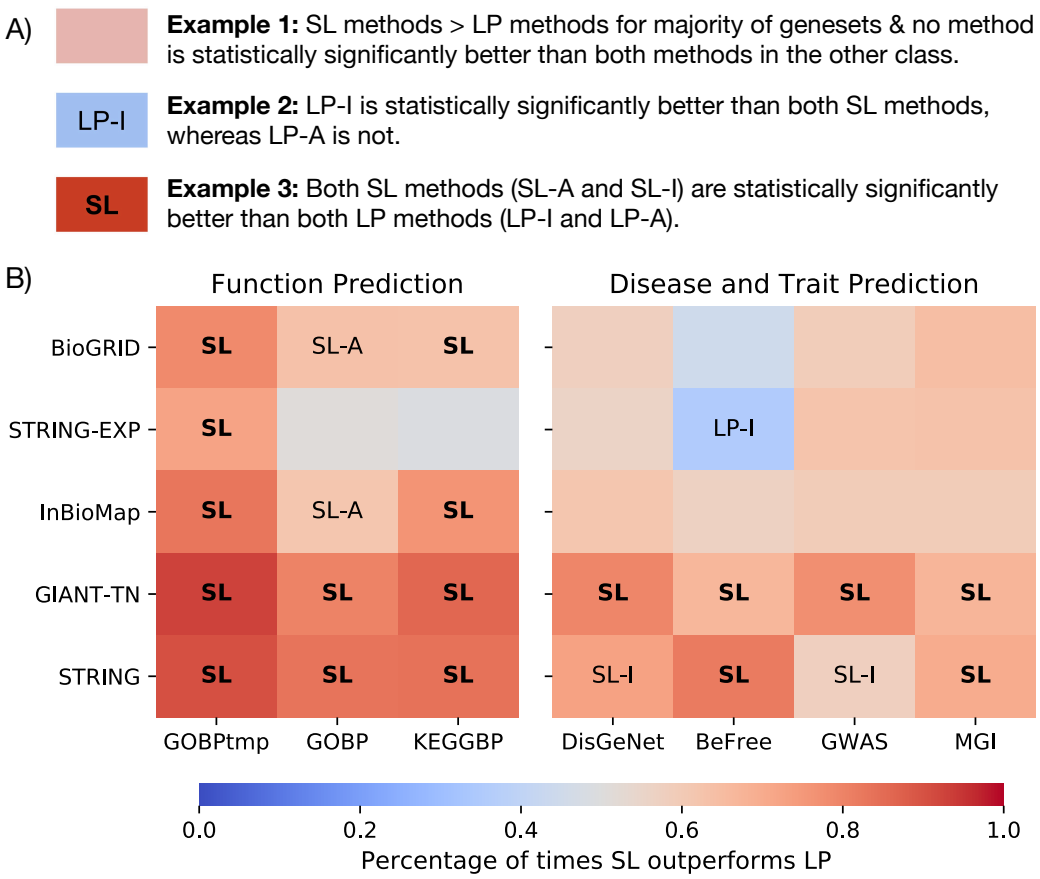


Figure 2.3: **Testing for a statistically significant difference between SL and LP methods.** (A) A key on interpreting the analysis. For each network-gene set combination, each method is compared to the two methods from the other class (i.e. SL-A vs LP-I, SL-A vs LP-A, SL-I vs LP-I, SL-I vs LP-A). If a method was found to be significantly better than both methods from the other class (Wilcoxon ranked-sum test with an FDR threshold of 0.05), the cell is annotated with that method. If both models in that class were found to be significantly better than the two methods in the other class, the cell is annotated in bold with just the class. The color scale represents the fraction of gene sets that were higher for the SL methods across all four comparisons. The first column uses GOBP temporal holdout, whereas the remaining 6 columns use study-bias holdout. (B) SL methods show a statistically significant improvement over LP methods, especially for function prediction.

CAFA [213, 112].

SL outperforms LP on a significant portion of tasks Following the observation that SL methods outperform LP methods based on relative ranking, we use a non-parametric paired test (Wilcoxon signed-rank test) to statistically assess the difference between specific pairs of methods (Figure 2.3A). For each gene set collection–network combination, we compare the two methods in one class to the two methods in the other class (i.e., we compare SL-A to LP-A, SL-A to LP-I, SL-I to LP-A,

and SL-I to LP-I). Each comparison yields a p-value along with the number of gene sets in the collection where one method outperforms the other. After correcting the four p-values for multiple hypothesis testing [26], if a method from one class outperforms both methods from the other class independently (in terms of the number of winning gene sets), and if both adjusted p-values are < 0.05, we consider a method to have significantly better performance compared to the entire other class.

In addition, we track the percentage of times the SL methods outperform the LP methods across all four comparisons within a gene set collection–network combination. The results show that, for function prediction, SL is almost always significantly better than LP when considering auPRC (Figure 2.3B). Based on temporal holdout on GOBP, both SL-A and SL-I always significantly outperform both LP methods. Based on the study-bias holdout evaluation, in the ten function prediction gene set collections–network combinations using GOBP and KEGGBP, SL-A is a significantly better method eight times (80%), and SL-I is a significantly better method six times (60%). Neither LP-I nor LP-A ever significantly outperform the SL models. The performance of SL and LP are more comparable for disease and trait prediction, but SL methods still perform better in a larger fraction of gene sets. For the 20 disease and trait gene set collection–network combinations, SL-I is a significantly better method eight times (40%), SL-A is a significantly better method six times (30%), LP-I is a significantly better method once (5%), and LP-A is never a significantly better method.

SL outperforms LP by notable effective sizes To visually inspect not only the relative performance of all four methods but also to see how well the models are performing in an absolute sense, we examine the boxplots of the auPRC values for every gene set collection–network combination (Figure 2.4). The first notable observation is that, regardless of the method, function prediction tasks have much better performance results than disease and trait prediction tasks (Figure 2.4B). Based on temporal holdout for function prediction (GOBPtmp), SL-A is the top-performing model based on the highest median performance for every network. Additionally, for all networks except STRING-EXP, SL-I is the second best-performing model. For the ten combinations of five networks

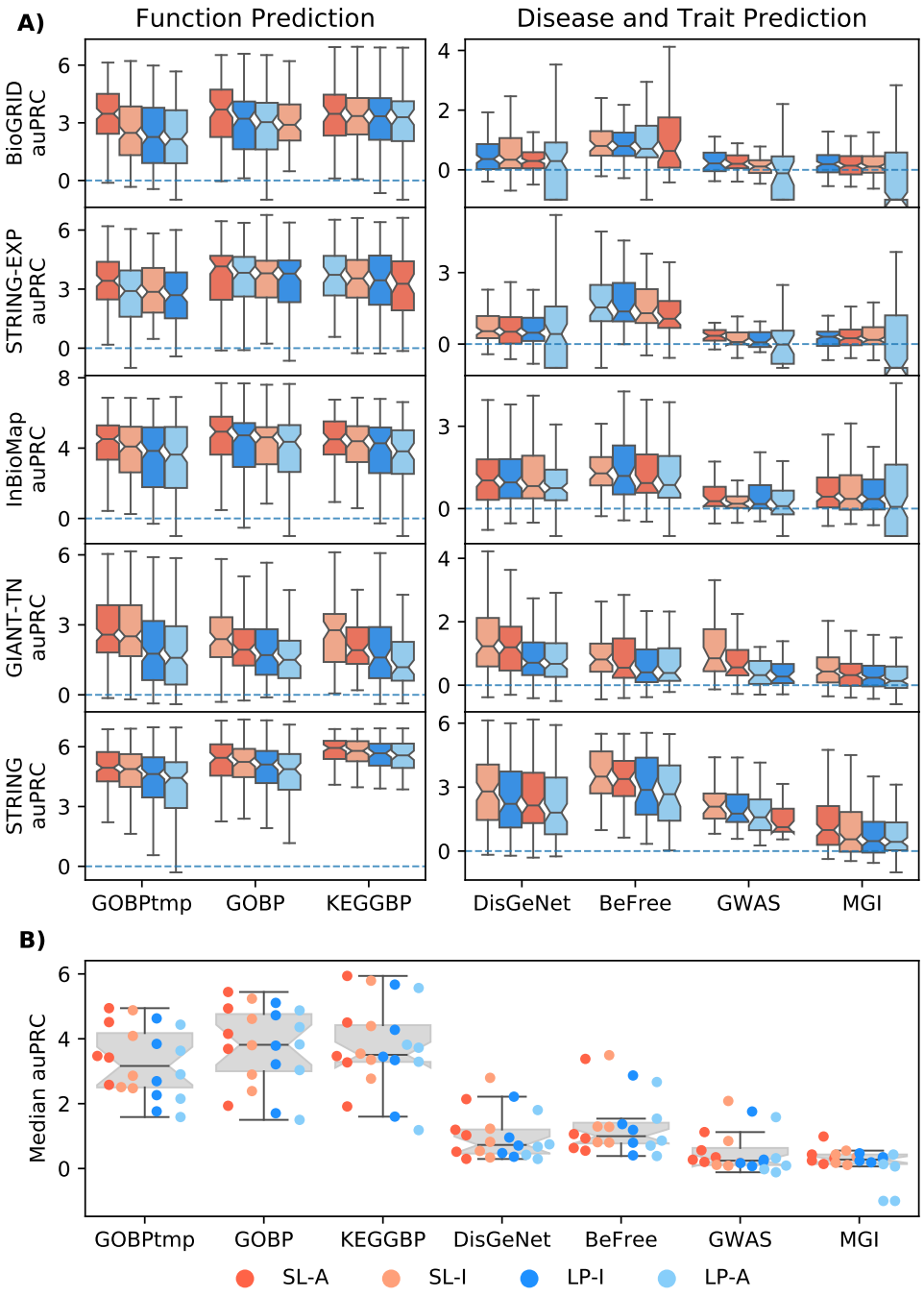


Figure 2.4: **Boxplots for performance across all gene set collection–network combinations.** (A) The performance for each individual gene set collection–network combination is compared across the four methods; SL-A (red), SL-I (light red), LP-I (blue), and LP-A (light blue). The methods are ranked by median value with the highest scoring method on the left. Results show SL methods outperform LP methods, especially for function prediction. (B) Each point in the plot is the median value from one of the boxplots in A. This shows that both SL and LP methods perform better for function prediction compared to disease/trait prediction.

with GOBP and KEGGBP, the top method based on the highest median performance is an SL method all but once, with SL-A being the top model seven times (70%), SL-I being the top model two times (20%, GOBP and KEGGBP on GIANT-TN), and LP-A being the top model once (10%, KEGGBP on STRING-EXP). As noted earlier, for disease and trait prediction, SL and LP methods have more comparable performance. Of the 20 gene set collection–network combinations, each of SL-A, SL-I, LP-I, and LP-A is the top method based on median performance five (25%), ten (50%), four (20%), and one (5%) times, respectively.

Although the boxplots in Figure 2.4 can give an idea of effect sizes, to further quantify this, we compute the ratios of auPRC values across all gene sets (Figure A.8). For each gene set, we calculate the ratio of auRPC values, find the percent increase or decrease, and then take the median value for every gene set collection–network combination. The results show that SL-A has a significant effect size when compared to LP-I for function prediction (53% for temporal holdout and 19% for study-bias holdout). In addition, for all prediction tasks, the effect size seen between the SL methods and LP-I is equal to or greater than that between LP-I and LP-A, where LP-I is widely considered a much better model than LP-A and thus, the comparison between LP-I and LP-A can be viewed as a baseline effect size.

2.3.2 Performance of SL correlates with network properties, similar to LP

Among the two classes of network-based models – SL and LP – it is intuitively clear how LP directly uses network connections to propagate information from the positively labeled nodes to other nodes close in the network. On the other hand, while SL is an accurate method for gene classification, it has not been studied if SL's performance is tied to any traditional notion of network connectivity. To shed light on this, we investigate the performance of SL-A and LP-I as a function of three different properties of individual gene sets in a collection: the number of annotated genes, edge density (a measure of how tightly connected the gene set is within itself), and segregation (a measure of how isolated the gene set is from the rest of the network). Detailed information on how the gene set and network properties are calculated are elaborated in Appendix A.1.3.

While the performance of neither SL-A nor LP-I has a strong association with the size of the

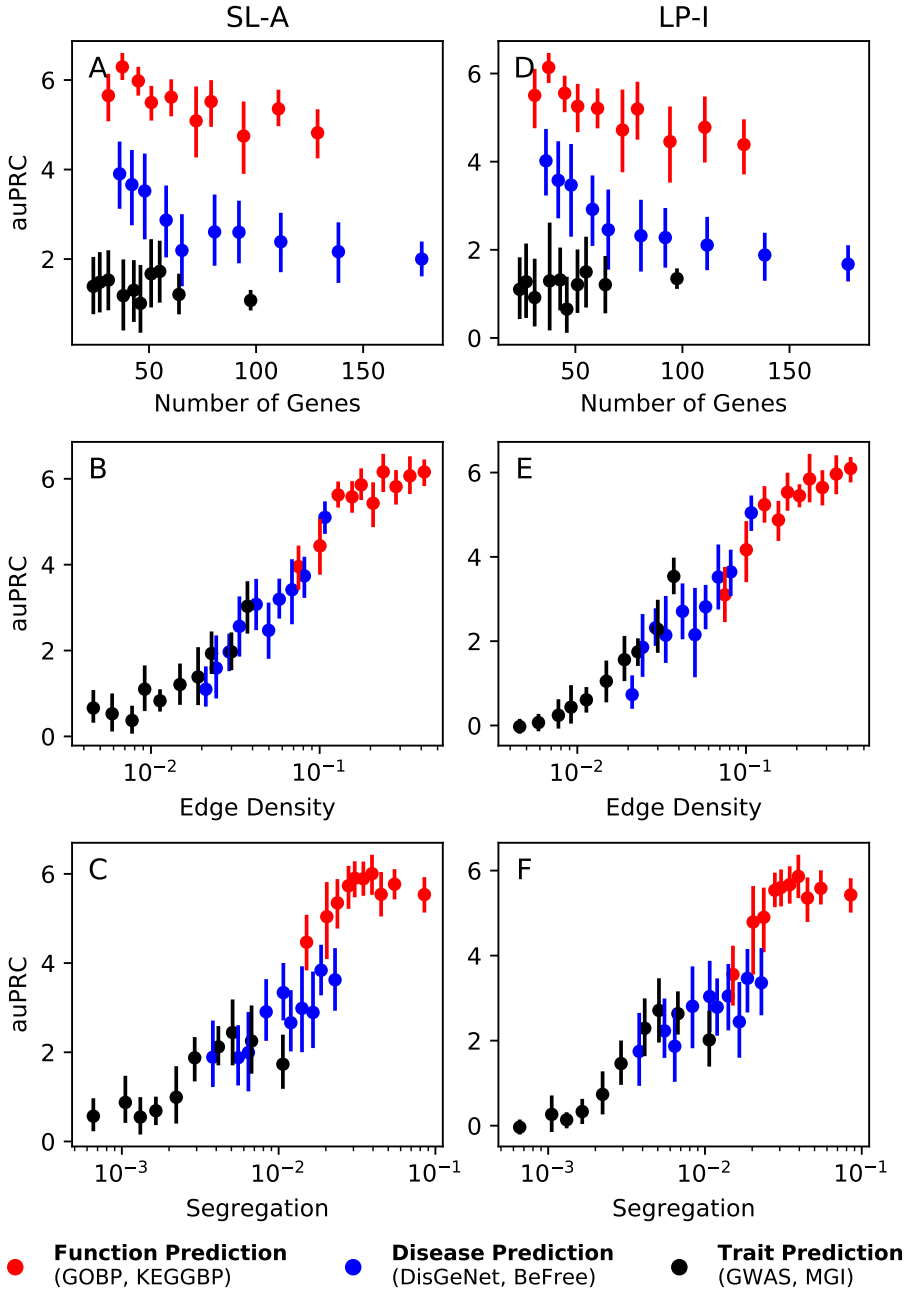


Figure 2.5: **Performance vs Network/gene set properties.** SL-A (A-C) is able to capture network information as efficiently as LP-I (D-F), for the STRING network. There is no correlation between the number of genes in the gene set versus performance (A,D), but there is a strong correlation between the performance and the edge density (B,E) as well as segregation (C,F). The different colored dots represent function gene sets (red, GOBP and KEGGBP), disease gene sets (blue, DIGenet and BeFree), and trait gene sets (black, GWAS and MGI). The vertical line is the 95% confidence interval. Similar trends can be seen for the other networks (Figure A.9).

gene set, the performance of SL-A has a strong positive correlation with both edge density and segregation of the gene set (Figure 2.5 left column). The observed trends are highly consistent on LP-I (Figure 2.5 right column). For visual clarity, Figure 2.5 presents results for just the STRING network, but similar results are seen in the other networks as well (Figure A.9). Thus, the performances of SL and LP across networks and types of prediction tasks are highly correlated with the local network clustering of the genes of interest. This result substantiates SL as an approach that can accurately predict gene attributes by taking advantage of local network connectivity.

The above observation also explains why SL and LP methods are, in general, more accurate for function prediction than disease and trait prediction (Figure 2.4B). This is because molecular interaction networks are primarily intended, either through curation or reconstruction, to reflect biological relationships between genes/proteins as they pertain to "normal" cellular function. Consequently, gene sets related to functions are more tightly clustered than those related to diseases and traits in the genome-wide molecular networks used in this study (Figure A.3), leading to suboptimal prediction performance of disease and trait gene predictions.

2.3.3 Network embedding methods demonstrate the potential for efficient and accurate network-based gene classification

As machine learning on node embeddings is gaining popularity for network-based node classification, we compare the top SL and LP methods tested here to this approach. Specifically, we compare LP-I and SL-A to an SL method using embeddings (SL-E) obtained from the *node2vec* algorithm [83] (Figure 2.6). For function prediction, we observe that SL-E substantially outperforms LP-I. For GOBP temporal holdout, SL-E is always significantly better than LP-I. For the GOBP and KEGGBP study-bias holdout, out of the ten gene set collection–network combinations, SL-E is significantly better than LP-I 5 times (50%), whereas the converse is true only once (10%). These patterns nearly reverse for the 20 disease/trait prediction tasks, with LP-I performing significantly better than SL-E 6 times (30%), and SL-E significantly outperforming LP-I 3 times (15%). Overall, SL-E performs comparably to the LP baselines, with clear advantages in function predictions.

On the other hand, the comparison between SL-E and SL-A shows that SL-A demonstrably

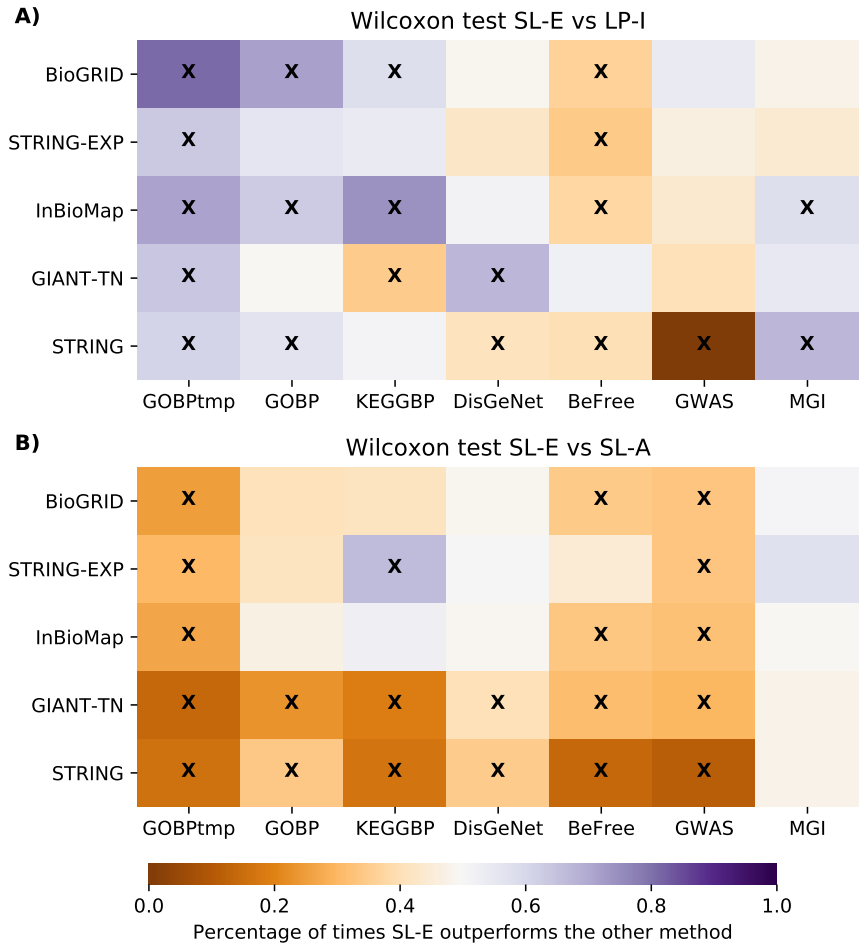


Figure 2.6: **Performance of SL-E vs LP-I and SL-A.** We compare the performance of supervised learning on the embedding matrix (SL-E) vs LP-I and SL-A using a Wilcoxon ranked-sum test. The performance metric is auPRC, the color scale represents the fraction of terms that were higher for the SL-E model (with purple being SL-E had a higher fraction of better performing gene sets compared to either LP-I or SL-A) and an $\times$ signifies that the p-value from the Wilcoxon test was below 0.05. A) Shows that SL-E is quite competitive with the classic method of LP-I and B) shows that SL-A outperforms SL-E in a majority of cases.

outperforms SL-E for both function and disease/trait prediction tasks. Among the 30 gene set collection–network combinations, SL-A is a significantly better model for 20 times (67%), whereas SL-E comes out on top just once (3%). This shows that although methods that use node embeddings are a promising avenue of research, they should be compared to the strong baseline of SL-A when possible. Meanwhile, the inherently low-dimensional nature of network embeddings offers substantial computational advantages over utilizing full network features. Therefore, even though there is a notable performance disparity between SL-E and SL-A currently, there lies a valuable

potential in further advancing graph embedding techniques, enabling accurate and efficient supervised learning on hundreds of gene classification tasks.

2.4 Conclusion

In this work, we have established that supervised learning systematically outperforms label propagation for network-based gene classification across networks and prediction tasks, including functions, diseases, and traits. Particularly, supervised learning, where every gene is its own feature, can capture network information just as well as label propagation. We further demonstrated that supervised learning on the adjacency matrix demonstrably outperforms supervised learning using node embeddings, and thus we strongly recommend that future work on using node embeddings for gene classification draws a comparison to using supervised learning on the adjacency matrix. Despite the observed performance discrepancies, the development of node embeddings remains a worthwhile pursuit due to their notable computational efficiency in training gene classification models.

2.5 Extensions

PyGenePlexus [167] packages the functionalities of this work into a user-friendly software written in Python (https://github.com/krishnanlab/PyGenePlexus). Users can easily build and generate predictions using SL or LP methods on the preprocessed molecular interaction networks given any queried gene sets. These predictions indicate how related every gene in the network is to the input gene set. Additionally, PyGenePlexus offers interpretability by highlighting relevant biological processes to the query gene set and returns the network connectivity of the top predicted genes to help biomedical researchers gain deeper insights into their biomedical problems of interest.

GenePlexusWeb [164] brings the usability of GenePlexus one step further by offering the same functionalities in PyGenePlexus on a website (https://www.geneplexus.net/). Once a user uploads their own set of human genes and chooses between a number of different human network representations, GenePlexusWeb returns a table showing the predictions of how relevant every gene in the network is to the input set. Interpretability is enhanced through visualizing the subnetwork induced

by top predicted genes on the website.

GenePlexusZoo [165] extends GenePlexus to five other species beyond human by considering many-to-many orthology information. It casts molecular networks across multiple species into a common embedding space using PecanPy (Chapter 3) and builds SL models on them. This cross-species extension of GenePlexus significantly enables research in human genes by leveraging current knowledge and experimental data from model organisms. We demonstrate that this multi-species network representation improves both gene classification within a single species and knowledge transfer across species, even in cases where the inter-species correspondence is undetectable based on shared orthologous genes. Thus, GenePlexusZoo effectively leverages the high evolutionary molecular, functional, and phenotypic conservation across species to discover novel genes associated with diverse biological contexts.

CHAPTER 3

PECANPY: A FAST, EFFICIENT AND PARALLELIZED PYTHON IMPLEMENTATION OF *NODE2VEC*

Learning low-dimensional representations (embeddings) of nodes in large graphs is critical to applying machine learning on massive biological networks. Node2vec is the most widely used method for node embedding. However, its original Python and C++ implementations scale poorly with network density, failing for dense biological networks with hundreds of millions of edges. We have developed PecanPy, a new Python implementation of node2vec that uses cache-optimized compact graph data structures and precomputing/parallelization to result in fast, high-quality node embeddings for biological networks of all sizes and densities. The PecanPy software and documentation are freely available at https://github.com/krishnanlab/pecanpy, along with the code to reproduce all the benchmarking experiments at https://github.com/krishnanlab/pecanpy_benchmarks.

3.1 Introduction

Large-scale molecular networks are powerful models that capture interactions between biomolecules (genes, proteins, metabolites) on a genome scale [171] and provide a basis for predicting novel associations between individual genes/proteins and various cellular functions, phenotypic traits, and complex diseases [153, 231]. An area of research that has gained rapid adoption in network science across disciplines is learning low-dimensional numerical representations, or *embeddings*, of nodes in a network for easily leveraging machine learning (ML) algorithms to analyze large networks [78, 39, 88]. Since each node's embedding vector concisely captures its network connectivity, node embeddings can be conveniently used as feature vectors in any ML algorithm to learn/predict node properties or links [88]. One of the earlier node embedding methods that continues to show good performance in various node classification tasks, especially on biological networks [153, 291, 183], is a random-walk based approach called *node2vec* [83]. Recent studies on the task of network-based gene classification have shown that *node2vec* achieves the best performance among the state-of-the-art embedding methods for gene classification [291] and that using embedding generated from *node2vec* achieves prediction performance comparable to the

state-of-the-art label propagation methods [153].

Despite its popularity, the original *node2vec* software implementations, written in Python or C++, have significant bottlenecks in seamlessly applying to all current biological networks. First, due to inefficient memory usage and data structure, they do not scale to large and dense networks produced by integrating several data sources on a genome-scale (17–26k nodes and 3–300mi edges) [80, 246]. On the other hand, the embarrassingly parallel precomputations of calculating transition probabilities and generating random walks are not parallelized in the original software. Resolving these issues will enable embedding large and dense biological networks, even large biological knowledge graphs. Finally, the original implementations only support integer-type node identifiers (IDs), making it inconvenient to work with molecular networks typically available in databases where nodes may have non-integer IDs.

Recent work presented in preprints [294] and unpublished code repositories have proposed other implementations of *node2vec* (Appendx B.1). However, they either do not provide publicly available software or do not present a full analysis of their implementation, including a benchmark that ensures the quality of the resulting embeddings. Here, we present PecanPy, an efficient Python implementation of *node2vec* that is parallelized, memory efficient, and accelerated using Numba [132] with a cache-optimized data structure (Figure 3.1). We have extensively benchmarked our software using networks from the original study and multiple additional large biological networks to demonstrate both the computational performance and the quality of the node embeddings. In the rest of this paper, we first summarize the optimization and the performance of PecanPy and then go into the details of our implementation.

3.2 PecanPy implementation and optimization

The *node2vec* procedure consists of four stages: loading, preprocessing, walking, and training. Comprehensive evaluations of the four stages (Figure B.1, B.2, B.3, B.4, B.5) show that the training stage only takes 1.2% (median) of the total runtime for the original Python implementation, in contrast to 95.1% for the original C++ implementation. The low fraction of training runtime indicates the efficiency of skip-gram training using the `gensim` Python package [216]. Therefore,

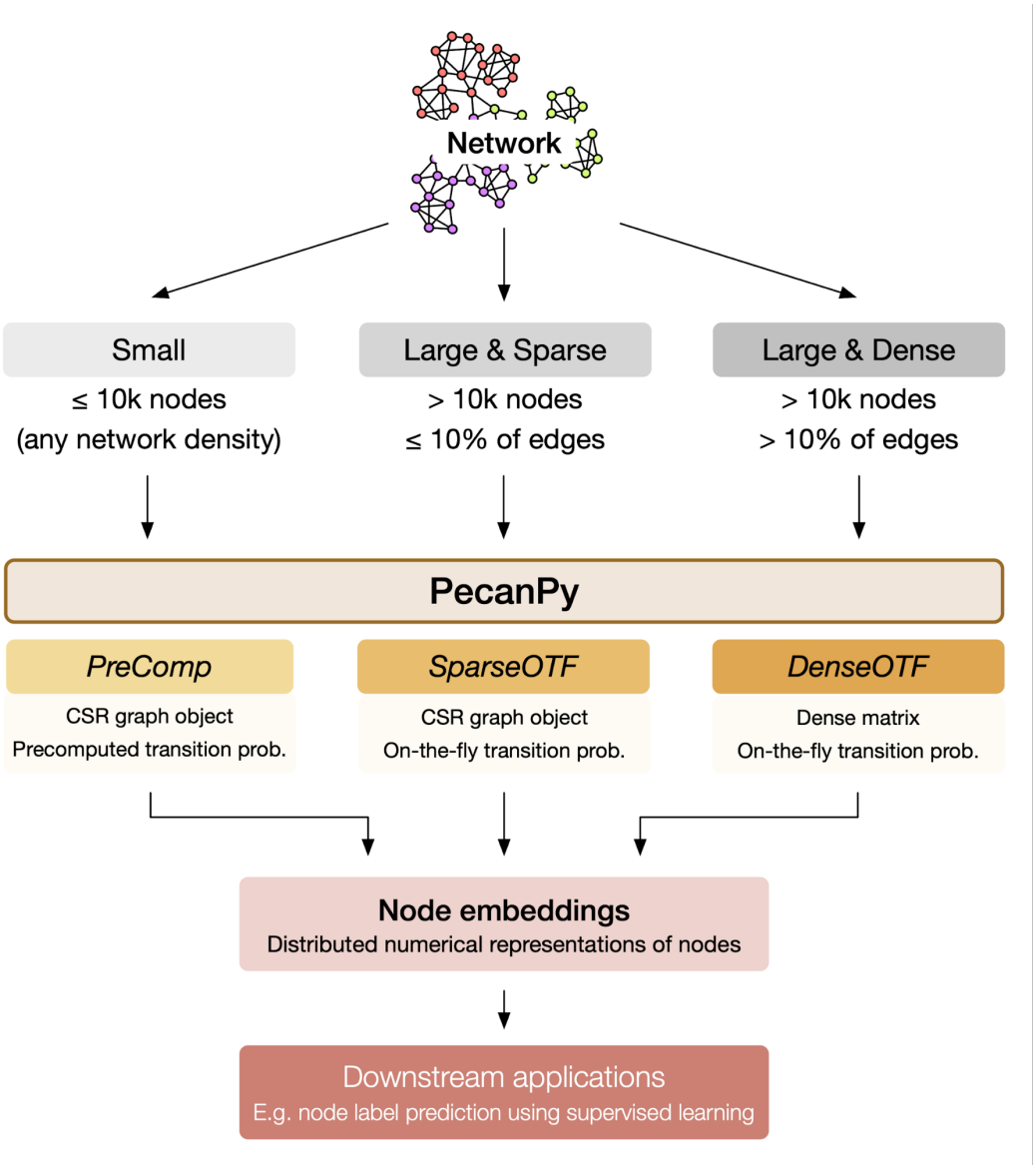


Figure 3.1: **Overview PecanPy for fast and efficient *node2vec*.** PecanPy is a Python implementation of the *node2vec* algorithm. PecanPy can operate in three different modes – PreComp, SparseOTF, and DenseOTF – that are optimized for networks of different sizes and densities; PreComp for networks that are small ($\leq 10,000$ nodes; any density), SparseOTF for networks that are large and sparse (>10,000 nodes and $\leq$ 10% of possible edges, i.e., density ≤ 0.1), and DenseOTF for dense networks ($> 10,000$ nodes and $>$ 10% of possible edges). These modes appropriately take advantage of compact sparse row (CSR) or dense matrix graph data structures, precomputing transition probabilities (prob.), and computing 2^{nd}-order transition probabilities during walk generation to achieve significant improvements in performance.

we only optimize the first three stages of the *node2vec* program.

We focus on the implementation in Python as it is currently the most widely-used high-level language in machine learning, making it convenient to use and develop further as part of the community. Further, the Numba compiler can be used to achieve C++ level performance. Our optimization spans three aspects: (1) we implement computationally- and memory-optimized *graph data structures* with efficient loading strategies, (2) we provide an option to compute transition probabilities *on-the-fly* without saving them, leading to significant reductions in memory usage and (3) we *parallelize* the processes of transition probability computation and walk generation. Finally, as a straightforward but critical improvement for usability, we have made string-type node IDs acceptable in addition to the integer type, which was the only ID type accepted by the original implementations.

We present these improvements as PecanPy, a new software for parallelized, efficient, and accelerated node2vec in Python (Figure 3.1). PecanPy operates in three different modes – PreComp, SparseOTF, and DenseOTF – each optimized for networks of different sizes and densities (Table 3.1). PreComp precomputes and stores all second-order transition probabilities as in the original implementations and, hence, is more suitable for small and sparse networks. On the other hand, SparseOTF and DenseOTF both compute second-order transition probabilities for the On-The-Fly during walk generation without saving them. SparseOTF (like PreComp), with its use of a compact sparse row (CSR) matrix representation, is better suited for networks that are large and sparse. DenseOTF, which uses the full adjacency matrix as the underlying graph data structure, is well-suited for dense networks. In the rest of this section, we provide a detailed description of the optimization done in PecanPy.

Table 3.1: **Capabilities of different node2vec implementations.**

Implementation	**Graph data structure**	**Precomp. trans. prob.**	**Parallelized walk**	**Non-integer IDs**
Original Python impl.	NetworkX	✓	✗	✗
Original C++ impl.	NetworkX	✓	✓	✗
`node2vectors`	NetworkX	✗	✓	✗
PecanPy-PreComp	CSR	✓	✓	✓
PecanPy-SparseOTF	CSR	✗	✓	✓
PecanPy-DenseOTF	Dense matrix	✗	✓	✓

3.2.1 Eficient graph data structure improves network loading and memory usage

First, we improve the underlying graph data structure, making it more efficient to load and operate on the graph data. In the original Python implementation, NetworkX [87], which implicitly assumes that the input is a multigraph, was used to handle all operations on networks using a graph object in the form of nested dictionaries (dict-of-dict-of-dict). The levels of these dictionaries correspond to node, neighbors of node, edge type, and edge weight. Thus, explicit declaration of edge types for weighted edges is required. However, *node2vec* only deals with homogeneous networks where only one edge type is present in the network. In the original Python implementation, all edge types are set to"weight" by default. This extra piece of information requires 295 additional bytes of memory for every single edge stored in a NetworkX graph (empty dictionary = 240 bytes, empty string = 49 bytes, single character = 1 byte). This requirement causes not only memory overheads but also computation overheads by reading the dictionary for "edge type,"which is irrelevant for homogeneous networks. To address these issues, in this work, we implemented a lite graph object as a network loader in the form of a list-of-dict, which assumes the network has only one type of edge. As shown in Figure B.5, the loading time and memory usage (first and second bars in each group) for list-of-dict are significantly lower than those for NetworkX. Thus, our special lite graph object efficiently loads networks with reduced memory usage and shorter load time than NetworkX.

3.2.2 Cache optimization further enhances operations on graphs

Next, we optimize the graph data structure further for better *cache utilization* during computation. Despite faster loading and lower memory usage using list-of-dict, operating on Python dictionaries is still suboptimal for cache utilization. Specifically, the neighboring-edge data, which are often used together to compute transition probabilities, are not physically close to each other in the memory. Meanwhile, the design principle behind cache is that units of memory that are physically nearby are likely to be used together. Consequently, every time a specific piece of data in memory is accessed, a chunk of neighboring physical memory – called the *cache line* – is also copied from RAM to cache. Reading from the cache could be up to 100 times faster than reading from RAM. Thus, to fully leverage the spatial locality of cache lines, inspired by a recent blog post (https://www.

singlelunch.com/2019/08/01/700x-faster-node2vec-models-fastest-random-walks-on-a-graph/), we further converted the list-of-dict graph data structure to the compact sparse row (CSR) format, implemented using NumPy arrays [252]. In this way, neighboring-edge data is placed physically close together, thus improving cache utilization. These optimizations result in speedups in the preprocessing (Figure B.4C) and walk generation (Figure B.4B) steps of PreComp compared to that of the original Python implementation in the single-core setup. More specifically, on the BlogCatalog and Wikipedia networks, PreComp achieves up to an order-of-magnitude speedup in the preprocessing phase over the original Python implementation.

Due to the compactness of the CSR graph data structure, the memory usage is further reduced compared to the list-of-dict graph data structure, as shown in Figure B.5B. However, CSR's compactness also hinders dynamically constructing sparse matrices, as adding new entries requires reconstructing the whole representation. Therefore, in practice, we first use list-of-dict to load the network into memory and then convert the full network to CSR, where all subsequent computations are performed.

3.2.3 Optimization for dense networks

The CSR representation described above is memory-efficient only if the network is relatively sparse because it stores non-zero entries of the adjacency matrix using both the indices and the weights of edges. For dense networks, explicit indexing of edges leads to memory overhead for the same reasons that redundant edge-type information strains memory, as discussed above. To address this issue, for dense networks like GIANT-TN (25,825 nodes; fully connected and weighted, with 333,452,400 edges), we use the dense NumPy matrix instead of the CSR to store the graph data.

CSR and dense matrix show similar walk generation times, as illustrated in a comparison between SparseOTF, which uses CSR, and DenseOTF (Figure B.1B). In the realm of dense networks, using dense matrix over CSR results in faster network loading and less memory usage. For dense networks, loading time takes up most of the runtime. For example, in the multi-core setup, loading the GIANT-TN network as CSR contributes to nearly 80% of the runtime (Figure B.1H). This burden is mostly due to the inefficiency in reading edge list files as text line-by-line. To mitigate this issue

of long loading time for dense networks, we implemented an option in our software that offers users the ability to first convert the edge list file to dense matrix format and save it as a binary Numpy npz file, which could then be loaded as a network in the future. As shown in Figure S6A, for sparse networks like PPI, loading networks as CSR is faster than loading as npz files, but as the density of the network increases, loading networks as npz files becomes a better option. Similar arguments could be made for memory usage (Figure B.5B). For GIANT-TN (Figure B.3D, SparseOTF vs DenseOTF), loading as npz only took 9 seconds with 6GB of peak memory usage, resulting in a 70-times speedup over CSR, which took more than 10 mins to load with 39GB of peak memory usage.

This trade-off due to the networks' densities raises a question of when exactly is using a dense matrix more optimal than using CSR. A dense matrix requires $8 \times |V|^2$ bytes of memory, while a CSR requires roughly $12 \times |E|$ bytes of memory (using 32-bit unsigned integer as index and 64-bit floating point number as data), where $|V|$ is the number of nodes and $|E|$ is the number of edges. Hence, in theory, for any network with a density of less than two-thirds, CSR should be used over the corresponding dense matrix. However, since CSR cannot be directly loaded but requires an intermediate conversion using a list-of-dict, the peak memory usage is also affected by the list-of-dict graph data structure. Based on the observations from Figure B.5B, where the fold difference in peak memory usage between CSR and Numpy matrix is much smaller for other sparse networks like BioGRID, we empirically set the balancing point for network density to be around 0.1.

3.2.4 Reducing memory usage via on-the-fly transition probability computation

Next, we detail our approach to reducing memory usage by computing 2^{nd} order transition probabilities on the fly. The original *node2vec* implementations precompute and store all the 2^{nd}-order transition probabilities in advance, which takes up at least $|E|^2/|V|$ space in memory. One solution to remedy this issue is to calculate the 2^{nd} order transition probabilities On-The-Fly (OTF) during walk generation without saving them. Another reason for not precomputing 2^{nd} order transition probabilities is that, as the network becomes larger and denser, it is very likely that most of the pre-calculated 2^{nd} order transition probabilities are never used in the generation of walks,

causing not only the space but also the invested computation time to be wasted. We combine the CSR and the dense matrix representation, each with the OTF strategy, into separate modes in our software, called SparseOTF and DenseOTF, respectively. As an example of the improvement made by the OTF strategy, the DenseOTF implementation can embed a dense network like the $\sim$ 26K fully-connected weighted GIANT-TN network with > 333M edges in just an hour with only 6GB of peak memory usage using a single core. Remarkably, this means that even for an extremely dense and large network like GIANT-TN, the software could be run on a personal computer configured with a reasonable amount of memory (e.g., 16GB). Conversely, both original implementations fail to run even the sparsified version GIANT-TN-c01 (similar number of nodes but with only $\sim$ 11% of the edges in GIANT-TN) on a supercomputer configured with 200GB memory. Moreover, the runtime of 1 hour for embedding the GIANT-TN network using DenseOTF with single-core configuration is even shorter than that for embedding a significantly smaller and sparser network STRING (67% of the nodes and 1% of edges as in GIANT-TN) using the original Python implementation with 28 cores, which took 5 hours to finish. For relatively small and sparse graphs where the amount of memory on a computer is sufficient to fit all 2^{nd}-order transition probabilities, it could indeed save time generating walks by avoiding redundant computations for transition probabilities (Figure B.1B). Hence, we leave the decision of the trade-off between speed and memory to the user by providing precomputation (PreComp) and the on-the-fly (SparseOTF or DenseOTF) modes in the PecanPy software.

3.2.5 Parallel random walk generation

All three modes of our new *node2vec* implementations are fully parallelized. As mentioned earlier, each node's transition probabilities computation and random walk generation are embarrassingly parallel. Specifically, in the process of walk generation, each node is used as a starting point for an independent random walk with a fixed length on the graph unless a dead end is reached, which causes early stopping. This process is repeated multiple times depending on the input parameter specified by the user. As each random walk is independent, multiple walks can be performed in parallel. Similarly, the precomputation of 2^{nd}-order transition probabilities only depends on the

1^{st}-order transition probabilities. Hence, in this work, the processes of walk generation and 2^{nd}-order transition probabilities precomputation are parallelized using Numba.

3.3 Benchmarking computational efficiency and quality of embeddings

3.3.1 Baseline *node2vec* implementations

We compile a list of existing *node2vec* implementations (Appendix B.1) and note that beyond the original implementations (in Python and C++), none of the other implementations have attributes that are likely to improve their speed and memory usage over and above the original Python or C++ implementations with the exception of one implementation called `nodevectors`. Benchmarking results for the overall speed and memory usage indicate that `nodevectors` handled at least one dense network better than the original implementations. However, the `nodevectors` node embedding vectors achieve very poor performance in node classification tasks (Figure 3.3). Therefore, we conduct our detailed, stage-by-stage benchmark of PecanPy only against the original Python and C++ implementations.

3.3.2 Benchmarking network data

We use a collection of eight networks for benchmarking different *node2vec* implementations. Table 3.2 shows the summary statistics of these networks, including the number of nodes, edges, and network density. PPI, BlogCatalog, and Wikipedia are from the original *node2vec* paper [83] (download link from node2vec webpage https://snap.stanford.edu/node2vec/). BioGRID [240], STRING [245], and GIANT-TN [80] are molecular interaction networks (download from https://doi.org/10.5281/zenodo.3352323), where nodes are proteins/genes and edges are interactions between them. GIANT-TN-c01 is a sub-network of GIANT-TN where edges with edge weight below 0.01 are discarded. SSN200 [135] is a cross-species network of proteins from 200 species (download from https://bioinformatics.cs.vt.edu/~jeffl/supplements/2019-fastsinksource/), with the edges representing protein sequence similarities.

Table 3.2: **Properties of the diverse networks used in this study.**

Network	Weighted	Num. nodes	Num. edges	Density	File size
PPI	✗	3,852	38,273	5.16E-3	707 KB
Wikipedia	✓	4,777	92,406	8.10E-3	2.0 MB
BlogCatalog	✗	10,312	333,983	6.28E-3	3.2 MB
BioGRID	✗	20,558	238,474	1.13E-3	2.5 MB
STRING	✓	17,352	3,640,737	2.42E-2	60 MB
SSN200	✓	814,731	72,618,574	2.19E-4	2.0 GB
GIANT-TN-c01	✓	25,689	38,904,929	1.18E-1	1.1 GB
GIANT-TN	✓	25,825	333,452,400	1.00E+0	7.2 GB

3.3.3 Runtime and memory usage

We profile the runtime and memory usage for each implementation on each network. All benchmarking experiments are done on compute units with 28-core Intel Xeon CPU E5-2680 v4 @2.4GHz. Each one of the four stages in the *node2vec* program is timed individually. To profile the original implementations, we add timers to each stage using the built-in time function in Python and C++. Memory usage is profiled using the system's built-in GNU timer, which could measure the maximum resident size (physical memory usage) throughout the runtime of a program. We profile the runtime and the memory usage of each implementation using two different resource configurations. The primary results presented are based on the *multi-core configuration*, with 28 cores, 200GB allocated memory, and a 24-hour wall-time limit. We also profile the implementations with a *single-core configuration*, with a single core, 32 GB memory allocated, and an 8-hour wall-time limit. The multi-core setup emulates performance on a high-performance computing facility, while the single-core setup emulates the scenario of the software being run on a personal computer.

3.3.4 Node embedding quality evaluation

Each of the three networks used in the *node2vec* paper (PPI, BlogCatalog, Wikipedia) contains node labels. We use these node classification tasks to probe the node embedding quality of the different implementations. Some of the labelsets from the data repository appeared to have very few positive examples. For a more rigorous evaluation, we only use label sets containing at least ten

positive examples. In total, there are 38 node classes in BlogCatalog, 50 node classes in PPI, and 21 node classes in Wikipedia. For each node class, a one vs. rest L2 regularized logistic regression model is trained and evaluated through 5-fold cross-validation. Each test fold is evaluated by the area under the receiver operator curve (auROC) separately, and the mean auROC score across the five folds is reported. This process is repeated ten times for each label set, and the mean value of the reported scores is taken as the final evaluation score. Using this evaluation procedure, each network and each implementation has a list of auROC scores depending on the number of node classes in the network. For each prediction task, we perform a Wilcoxon paired test to compare this list of scores to that of the original Python implementation. The resulting statistics are used to determine whether the quality of the resulting node embedding from a particular implementation is statistically significantly different from that of the original implementation

3.4 Results and discussions

We comprehensively benchmark PecanPy and the original Python and C++ implementations of *node2vec* on a collection of eight networks, including three networks from the original *node2vec* paper [83] and five large biological networks that together span a wide range of sizes (approximately 4K to 800K nodes, and approximately 38K to 333M edges) and densities (0.02% to 100%; Table 3.2; Figure 3.2, B.6, B.7, and B.8).

3.4.1 PecanPy significantly improves runtime and memory usage for *node2vec* over the original implementations

Across the board, PecanPy (in one of its three modes) is substantially faster than the original implementations (Figure 3.2). In fact, there are three large networks (SSN200, GIANT-TN-c01, GIANT-TN) that run successfully only using PecanPy 's OTF implementations. Other implementations failed to run the GIANT-TN network due to memory limitations that arise from storing 2^{nd}-order transition probabilities. The original software failed to run SSN200 because they do not support non-integer-type node IDs; this is supported in PecanPy. DenseOTF failed for SSN200 since its dense-network design requires more than 5TB of memory to create a double precision dense matrix of size 800k. However, it considerably improves memory usage and speed

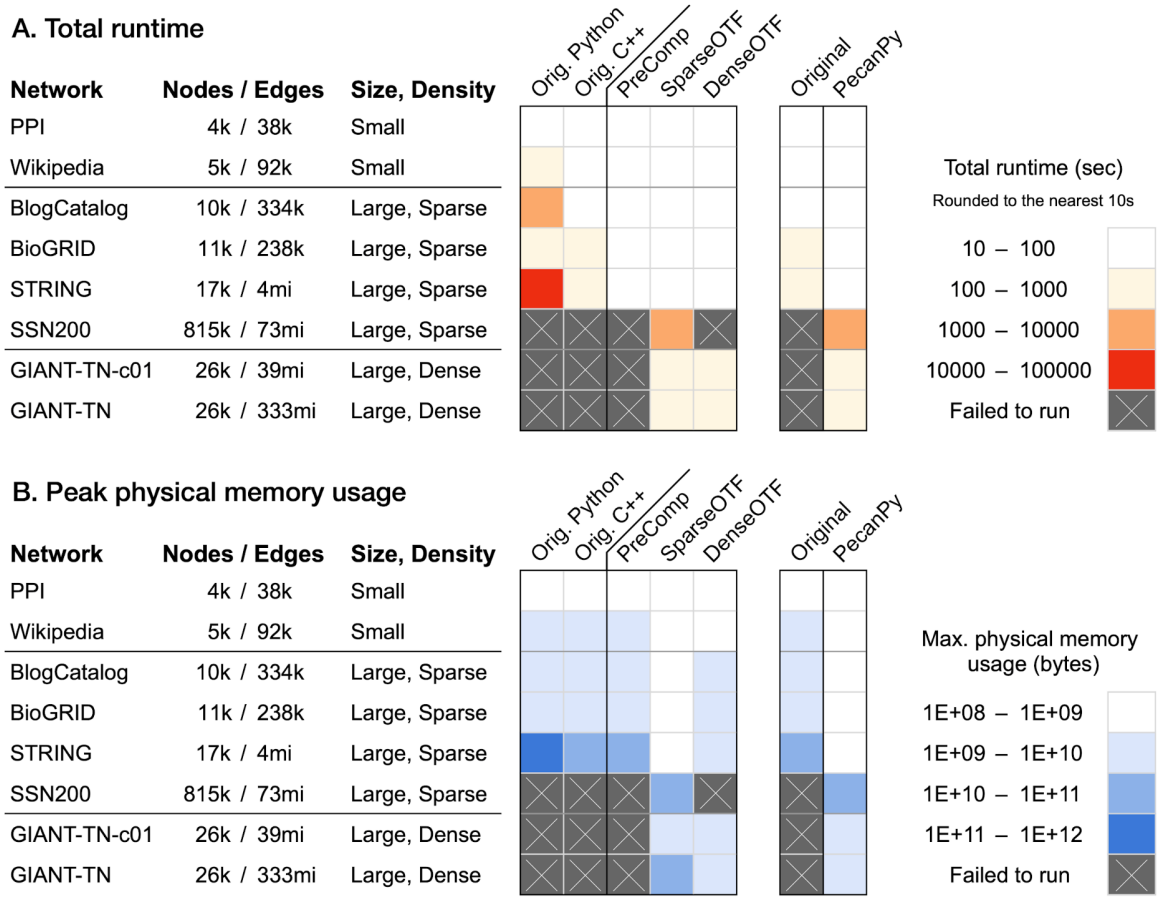


Figure 3.2: **Summary of runtime and memory of PecanPy and the original implementations of node2vec using multiple cores.** The eight networks of varying sizes and densities are along the rows. The software implementations are along the columns. The first heatmap (on the left) shows the performance of the original Python and C++ software along with the three modes of PecanPy (PreComp, SparseOTF, and DenseOTF). The adjacent 2-column heatmap (on the right) summarizes the performance of the original (best of Python and C++ versions) and PecanPy (best of PreComp, SparseOTF, and DenseOTF) implementations. Lighter colors correspond to lower runtime in panel A and lower memory usage in panel B. Crossed grey indicates that the particular implementation (column) failed to run for a particular network (row).

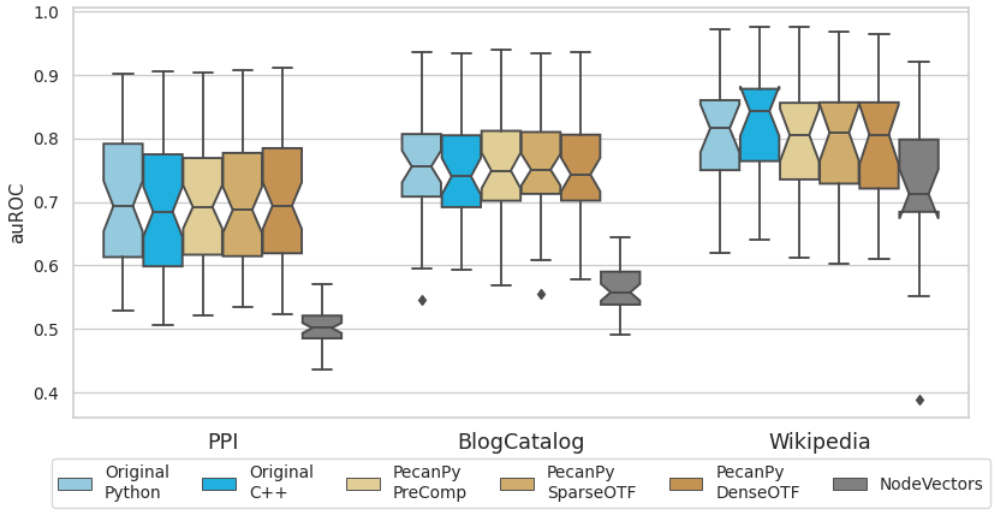


Figure 3.3: **Evaluation of embedding in node classification tasks.** Each group of boxplots corresponds to one of three networks. Individual boxplots in a group correspond to the distribution of auROC scores using node embeddings generated using a specific implementation (different colors).

for large dense networks like GIANT-TN due to the more efficient network loading scheme, i.e., reading a numpy array file instead of a text (edge list) file (Figure B.3). For relatively small and sparse networks (e.g., BioGRID, BlogCatalog), using PreComp invariably results in faster walk generation, thus achieving an overall shorter runtime (Figure B.3). These results underscore the importance of the three modes of PecanPy. Similarly, in terms of memory usage, one of the modes of the PecanPy reduces the maximum resident size by up to two orders of magnitude compared to the original implementations. All these trends are magnified on a single core (Figure B.2, B.6, B.8). Another implementation `nodevectors` that handled at least one dense network better than the original implementations still performed worse overall than PecanPy (Figure B.9). Together, these results demonstrate that PecanPy is much more computationally and memory efficient than the original *node2vec* implementations.

3.4.2 PecanPy produces *node2vec* embedding of the same quality as the original implementations

To ensure the quality of node embeddings generated by our new implementations, we evaluated their use as feature vectors in node classification tasks using datasets from the original paper,

including BlogCatalog, PPI, and Wikipedia. As shown in Figure 3.3, our implementations achieve the same performance as the original Python implementation. On the other hand, the original C++ implementation is significantly worse than the original Python implementation for PPI (Wilcoxon p-value = 1.98e-3) while being significantly better for Wikipedia (Wilcoxon p-value = 2.41e-4). These differences are likely due to the different skip-gram implementations in the C++ and the Python implementations. Finally, the performance of `nodevectors` feature vectors is worse than that of all other implementations, potentially due to implementation errors by the author. In summary, PecanPy is an accurate reimplementation of the original *node2vec*, resulting in statistically indifferent performance on downstream node classification tasks.

3.5 Conclusion

We have developed an efficient *node2vec* Python software, PecanPy, with significant improvement in both speed and memory usage. Extensive benchmarks have demonstrated that PecanPy can efficiently generate quality node embeddings for networks at multiple scales, including large (>800k nodes) and dense (fully connected network of 26k nodes) networks that the original implementations failed to execute. PecanPy is freely available at https://github.com/krishnanlab/pecanpy, can be easily installed via the pip package-management system (https://pypi.org/project/pecanpy/), and has been confirmed to work on a variety of networks, weighted or unweighted, with a wide range of sizes and densities. Therefore, it can find broad utility beyond biology.

CHAPTER 4

ACCURATELY MODELING BIASED RANDOM WALKS ON WEIGHTED NETWORKS USING *NODE2VEC+*

Accurately representing biological networks in a low-dimensional space, also known as network embedding, is a critical step in network-based machine learning and is carried out widely using *node2vec*, an unsupervised method based on biased random walks. However, while many networks, including functional gene interaction networks, are dense, weighted graphs, *node2vec* is fundamentally limited in its ability to use edge weights during the biased random walk generation process, thus under-using all the information in the network. Here, we present *node2vec+*, a natural extension of *node2vec* that accounts for edge weights when calculating walk biases and reduces to *node2vec* in the cases of unweighted graphs or unbiased walks. Using two synthetic datasets, we empirically show that *node2vec+* is more robust to additive noise than *node2vec* in weighted graphs. Then, using genome-scale functional gene networks to solve a wide range of gene function and disease prediction tasks, we demonstrate the superior performance of *node2vec+* over *node2vec* in the case of weighted graphs. Notably, due to the limited amount of training data in the gene classification tasks, graph neural networks such as GCN and GraphSAGE are outperformed by both *node2vec* and *node2vec+*. Code to reproduce the benchmarking results is available on GitHub: https://github.com/krishnanlab/node2vecplus_benchmarks.

4.1 Introduction

Graphs and networks naturally appear in many real-world datasets, including social networks and biological networks. The graph structure provides insightful information about the role of each node in the graph, such as protein function in a protein-protein interaction network [153, 128]. To more efficiently and effectively mine information from large-scale graphs with thousands or millions of nodes, several node embedding methods have been developed [53, 89]. Among them, *node2vec* has been the top choice in bioinformatics due to its superior performance compared to many other methods [16, 291]. However, many biological networks, such as [80, 113], are dense and weighted by construction, which we demonstrate to be undesirable conditions for *node2vec* that can lead to

sub-optimal performance.

Node2vec [83] is a second-order random walk based embedding method. It is widely used for unsupervised node embedding for various tasks, particularly in computational biology [183], such as for gene function prediction [153], disease gene prediction [199, 15], and essential protein prediction [266, 292]. Some recent works built on top of *node2vec* aim to adapt *node2vec* to more specific types of networks [270, 251], generalize *node2vec* to higher dimensions [86], augment *node2vec* with additional downstream processing [97], or to study *node2vec* theoretically [82, 56, 212]. Nevertheless, none of these follow-up works account for the fact that *node2vec* is less effective for weighted graphs, where the edge weights reflect the (potentially noisy) similarities between pairs of nodes. This failing is due to the inability of *node2vec* to differentiate between small and large edges connecting the previous vertex with a potential next vertex in the random walk, which subsequently causes less accurate modeling of the intended walk bias.

Meanwhile, another line of recent works on graph neural networks (GNNs) has shown remarkable performance in prediction tasks that involve graph structure, including node classification [33, 279]. Although GNNs and embedding methods like *node2vec* are related in that they both aim at projecting nodes in the graph to a feature space, two main differences set them apart. First, GNNs typically require labeled data, while embedding methods do not. This label dependency makes the embeddings generated by a GNN tied to the quality of the labels, which in some cases, like in biological networks, are noisy and scarce. Second, GNNs typically require node features as input to train, which are not always available. In the absence of given node features, one needs to generate them, and often GNN algorithms use trivial node features such as the constant features or node degree features. These two differences give node embedding methods a unique place in node classification, apart from the GNN methods.

Here, we propose an improved version of *node2vec* that is more effective for weighted graphs by taking into account the edge weight connecting the previous vertex and the potential next vertex. The proposed method *node2vec+* is a natural extension of *node2vec*; when the input graph is unweighted, the resulting embeddings of *node2vec+* and *node2vec* are equivalent in expectation.

Moreover, when the bias parameters are set to neutral, *node2vec+* recovers a first-order random walk, just as *node2vec* does. Finally, we demonstrate the superior performance of *node2vec+* through extensive benchmarking on both synthetic datasets and network-based gene classification datasets using various functional gene interaction networks. *Node2vec+* is implemented as part of `PecanPy` [151] and is available on GitHub: https://github.com/krishnanlab/PecanPy.

4.2 Methods

We start by briefly reviewing the *node2vec* method. Then we illustrate that *node2vec* is less effective for weighted graphs due to its inability to identify *out* edges. Finally, we present a natural extension of *node2vec* that resolves this issue.

4.2.1 *Node2vec* overview

In the setting of node embeddings, we are interested in finding a mapping $f : V \to \mathbb{R}^d$ that maps each node $v \in V$ to a d-dimensional vector so that the mutual proximity between pairs of nodes in the graph is preserved. In particular, a random walk based approach aims to maximize the probability of reconstructing the neighborhoods for any node in the graph based on some sampling strategy S. Formally, given a graph $G = (V, E)$ (the analysis generalizes to directed and/or weighted graphs), we want to maximize the log probability of reconstructing the sampled neighborhood $\mathcal{N}_S(v)$ for each $v \in V$:

$$\max_f \sum_{v \in V} \log \mathbb{P}(\mathcal{N}_S(v) | f(v)) \tag{4.1}$$

Under the conditional independence assumption, and the parameterization of the probabilities as the softmax normalized inner products [83, 173], the objective function above simplifies to:

$$\max_f \sum_{v \in V} \left(\sum_{v' \in \mathcal{N}_S(v)} \langle f(v'), f(v) \rangle - \log Z_v \right) \tag{4.2}$$

In practice, the partition function $Z_v = \sum_{v' \in V} \langle f(v), f(v') \rangle$ is approximated by negative sampling [174] to save computational time. Given any sampling strategy S, equation (4.2) can find the corresponding embedding f, which is achieved in practice by feeding the random walks generated to the skipgram with negative sampling [173].

Node2vec devises a second-order random walk as the sampling strategy. Unlike a first-order random walk [202], where the transition probability of moving to the next vertex v_n, denoted as $\mathbb{P}(v_n|v_c)$, depends only on the current vertex v_c, a second-order random walk also depends on the previous vertex v_p, with transition probability $\mathbb{P}(v_n|v_c, v_p)$. It does so by applying a bias factor$\alpha_{pq}(v_n, v_p)$ to the edge $(v_c, v_n) \in E$ that connects the current vertex and a potential next vertex. This bias factor is a function that depends on the relation between the previous vertex and the potential next vertex, and is parameterized by the *return* parameter p, and the *in-out* parameter q. In this way, the random walk can be generated based on the following transition probabilities:

$$\mathbb{P}(v_n|v_c, v_p) = \begin{cases} \frac{\alpha_{pq}(v_n,v_p)w(v_c,v_n)}{\sum_{v\in\mathcal{N}(v_c)}\alpha_{pq}(v,v_p)w(v_c,v)} & \text{if } (v_c, v_n) \in E \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

where the bias factor is defined as:

$$\alpha_{pq}(v_n, v_p) = \begin{cases} \frac{1}{p} & \text{if } v_p = v_n \\ 1 & \text{if } v_p \neq v_n \text{ and } (v_n, v_p) \in E \\ \frac{1}{q} & \text{if } v_p \neq v_n \text{ and } (v_n, v_p) \notin E \end{cases} \tag{4.4}$$

According to this bias factor, *node2vec* differentiates three types of edges: 1) the *return* edge, where the potential next vertex is the previous vertex (Figure 4.1a); 2) the *out* edge, where the potential next vertex is *not* connected to the previous vertex (Figure 4.1b); and 3) the *in* edge, where the potential next vertex is connected to the previous vertex (Figure 4.1c). Note that the first-order (or unbiased) random walk can be seen as a special case of the second-order random walk where both the *return* parameter and the *in-out* parameter are set to neutral ($p = 1,\ q = 1$).

We now turn our attention to weighted networks, where the edge weights are not necessarily zeros or ones. Consider the case where v_n is connected to v_p, but with a small weight (Figure 4.1d), i.e. $(v_n, v_p) \in E$ and $0 < w(v_n, v_p) \ll 1$. According to the definition of the bias factor, no matter how small $w(v_n, v_p)$ is, (v_c, v_n) would always be considered as an *in* edge. Since in this case v_n and v_p are barely connected, (v_c, v_n) should in fact be considered as an *out* edge. In the extreme case of

a fully connected weighted graph, where $(v, v') \in E$ for all $v, v' \in V$, *node2vec* completely loses its ability to identify *out* edges.

Thus, *node2vec* is less effective for weighted networks due to its inability to identify potential *out* edges where the terminal vertex v_n is loosely connected to a previous vertex v_p. Next, we propose an extension of *node2vec* that resolves this issue, by taking into account of the edge weight $w(v_n, v_p)$ in the bias factor.

4.2.2 *Node2vec+*

The main idea of extending *node2vec* is to identify potential *out* edges $(v_c, v_n) \in E$ coming from v_p, where v_n is loosely connected to v_p. Intuitively, we can determine the "looseness" of (v_c, v_n) based on some threshold edge value. However, given that the distribution of edge weights of any given node in the graph is not known *a priori*, it is hard to come up with a reasonable threshold value for all networks. Instead, we define the looseness of (v_c, v_n) based on the edge weight statistics for each node v.

$$\begin{aligned} \mu(v) &= \frac{\sum_{v' \in \mathcal{N}(v)} w(v, v')}{|\mathcal{N}(v)|} \\ \sigma(v) &= \sqrt{\frac{\sum_{v' \in \mathcal{N}(v)} \big(w(v, v') - \mu(v)\big)^2}{|\mathcal{N}(v)|}} \\ \tilde{w}_\gamma(v, u) &= \frac{w(v, u)}{\max\{\mu(v) + \gamma\sigma(v), \epsilon\}} \end{aligned} \tag{4.5}$$

Formally, we first define $\tilde{w}_\gamma(v, u)$, a normalized version of the edge weight $w(v, u)$, based on the mean $\mu(v)$ and the standard deviation $\sigma(v)$ of the edge weights connecting v, as in equation (4.5). In practice, we clip the denominator of $\tilde{w}_\gamma(v, u)$ by a small number ϵ (1e-6 by default) to prevent divide by zero in some cases when γ is set to be negative. Then, we say $v \in V$ is γ-*loosely connected* (or simply *loosely connected* if $\gamma = 0$) to $u \in V$ if $\tilde{w}_\gamma(v, u) < 1$. Intuitively, we would like to treat an edge as being "not connected" if it is "small enough". Finally, an edge (v, u) is γ-*loose* if v is γ-loosely connected to u, and otherwise it is γ-*tight*. Without loss of generality, we consider the case of $\gamma = 0$ in the subsequent sections to simplify the notion of *looseness*.

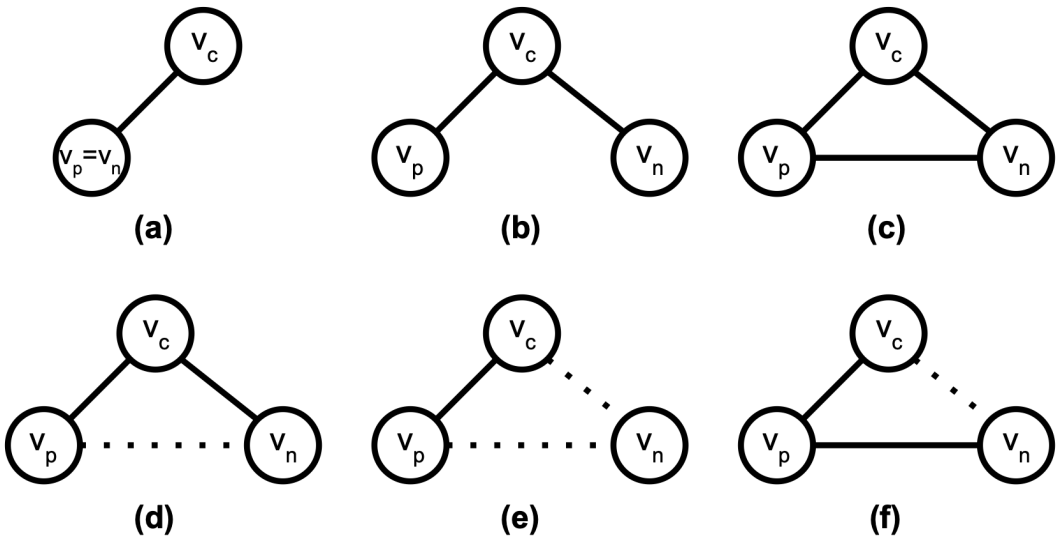


Figure 4.1: **Illustration of different settings of *return* and *in-out* edges.** v_p, v_c, and v_n indicate the previous, current, and next vertices. The solid and dotted lines represent edges with large and small edge weights, respectively. (a-c) are *return*, *out*, and *in* edges considered by *node2vec*. (d-f) are variations of (c) when taking into account of edge weights, where node2vec fail to distinguish from (c).

Based on the definition of looseness of edges, and assuming $v_p \neq v_n$, there are four types of (v_c, v_n) edges (see Figure 4.1, (c-f)). Following *node2vec*, we categorize these edge types into *in* and *out* edges. Furthermore, to prevent amplification of noisy connections, we added one more edge type called the *noisy* edge, which is always suppressed.

***Out* edge (4.1b, 4.1d)** As a direct generalization to *node2vec*, we consider (v_c, v_n) to be an *out* edge if (v_c, v_n) is tight and (v_n, v_p) is loose. The *in-out* parameter q then modifies the out edge to differentiate "inward" and "outward" nodes, and subsequently leads to Breadth First Search or Depth First Search like searching strategies [83]. Unlike *node2vec*, however, we further parameterize the bias factor α based on $\tilde{w}_\gamma(v_n, v_p)$. Any choice of monotonic function should work, but we choose to use the linear interpolation in this study for simplicity and leave it as future work to explore more sophisticated interpolation functions such as the sigmoidal functions. Specifically, for an *out* edge (v_c, v_n), the bias factor is computed as $\alpha_{\gamma pq}(v_p, v_c, v_n) = \frac{1}{q} + (1 - \frac{1}{q})\tilde{w}_\gamma(v_n, v_p)$. Thus the amount of modification to the *out* edge depends on the level of looseness of (v_n, v_p). When $w(v_n, v_p) = 0$, or equivalently $(v_n, v_p) \notin E$, the bias factor for (v_c, v_p) is $\frac{1}{q}$, same as that defined in *node2vec*.

***Noisy* edge (4.1e)** We consider (v_c, v_n) to be a *noisy* edge if both (v_c, v_n) and (v_n, v_p) are loose. Heuristically, the *noisy* edges are not very informative and thus should be suppressed regardless of the setting of q to prevent amplification of noise. Thus, the bias factor for a *noisy* edge is set to be $\min\{1, \frac{1}{q}\}$.

***In* edge (4.1c, 4.1f)** Finally, we consider (v_c, v_n) to be an *in* edge if (v_n, v_p) is tight, regardless of $w(v_c, v_n)$. The corresponding bias factor is set to neutral as in *node2vec*.

Combining the above, the bias factor for *node2vec+* is defined as follows:

$$\alpha_{\gamma pq}(v_p, v_c, v_n) = \begin{cases} \frac{1}{p} & \text{if } v_p = v_n \\ 1 & \text{if } \tilde{w}_\gamma(v_n, v_p) \geq 1 \\ \min\{1, \frac{1}{q}\} & \text{if } \tilde{w}_\gamma(v_n, v_p) < 1 \\ & \quad \text{and } \tilde{w}_\gamma(v_c, v_n) < 1 \\ \frac{1}{q} + (1 - \frac{1}{q})\tilde{w}_\gamma(v_n, v_p) & \text{if } \tilde{w}_\gamma(v_n, v_p) < 1 \\ & \quad \text{and } \tilde{w}_\gamma(v_c, v_n) \geq 1 \end{cases} \tag{4.6}$$

Note that the last two caess in equation (4.6) include cases of $(v_n, v_p) \notin E$. Based on the biased random walk searching strategy using this bias factor, the embedding can be generated accordingly using (4.2). One can verify, by checking equation (4.6), that this is indeed a natural extension of *node2vec* in the sense that (1) for an unweighted graph, the *node2vec+* is equivalent to *node2vec*, and (2) when p and q are set to 1, *node2vec+* recovers a first-order random walk, same as *node2vec* does. Finally, by design, *node2vec+* is able to identify potential *out* edges that would have been obliviated by *node2vec*.

4.2.3 Synthetic hierarchical cluster graphs construction

In the next section, we first illustrate the effectiveness of using several synthetic graphs. Here, we provide the detailed construction steps for the hierarchical cluster graphs. At a high level, they are created by first sampling in a representation space of the corresponding tree structure and then applying an RBF kernel to obtain pairwise connection scores. In the following, we describe (1) the construction of the tree that represents the hierarchy, (2) the representation each node in the tree,

(3) constructing the hierarchical cluster graph given the node representations, and (4) maximally sparsifying the constructed hierarchical cluster graph.

Tree construction We first construct the cluster centroids using a tree structure. A perfect binary tree is a binary tree where all nodes except the leaf nodes have two children, and all leaf nodes have the same level. This definition can be generalized to perfect K-trees, in which all the interior nodes have K number of nodes, for K greater than or equal to one. We denote $T_{K,L}$ as the perfect K-tree with maximum level L. Figure S1 shows the example of $T_{2,2}$, a perfect binary tree (or perfect 2-tree) with a maximum level of two.

Representing nodes in the tree A straightforward solution to represent the nodes in $T_{K,L}$ is one-hot encoding. For a more compact representation, we leave out the indicator for the root node and represent it as all zeros. Thus, the dimension of the indicator array is equal to the total number of nodes in $T_{K,L}$, excluding the root node, that is,

$$|V(T_{K,L})| - 1 = \sum_{l=1}^{L} K^l \tag{4.7}$$

However, if we use one-hot encoding, then all nodes are equally distanced in the Euclidean space. Instead, we combine the one-hot encoded representations of all the ancestor nodes as the final representation for each node, denoted as $\phi_i,\ i = 1, \ldots, |V(T_{K,L})|$. In this way, all sibling nodes are equally distanced, with $\sqrt{2}$ times the distance from the parent node. Figure S1 shows the example of both the one-hot encoded and the final representations of the grey nodes. Notice the difference between the final representation and the on-hot encoded representation of the grey leaf node.

Hierarchical clusters We draw data points x from a Gaussian distribution around each node in the $T_{K,L}$ tree:

$$x \sim \mathcal{N}(\phi_i, \sigma),\ i = 1, \ldots, |V(T_{K,L})| \tag{4.8}$$

In the case of K3L2, the data points are drawn using $T_{3,2}$. The parameter , which controls the noisiness of the sampled data points, is set to 0.01 by default. Throughout the study, we fix the number of data points per node in the tree to 30. Finally, we turn the sampled data points into a fully connected weighted graph using the RBF kernel.

Maximal sparsification of K3L2 We apply a global edge threshold to K3L2 by removing all edges below a certain value t. Sweeping over $[0.01, 0.9]$, we find that the maximum global edge threshold that preserves the graph's connectivity is about 0.45 (Figure 4.2). This sparsification drastically reduces the density of the graph from 1.0 (fully connected) down to about 0.1.

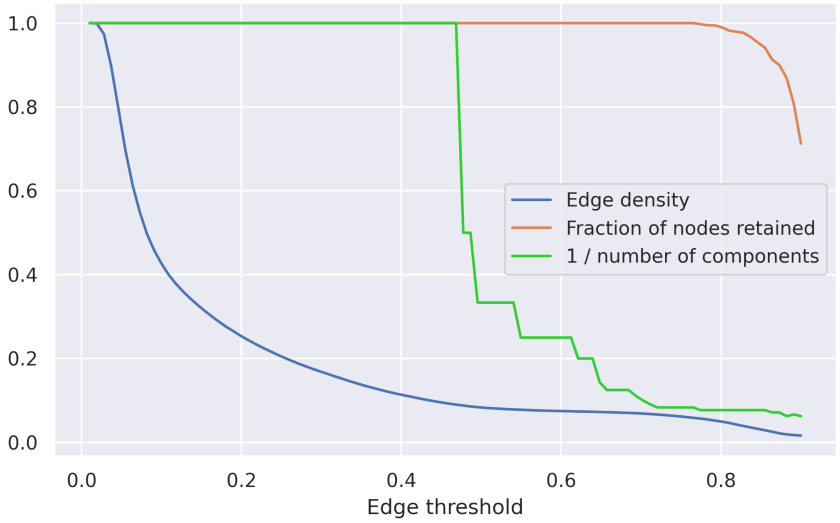


Figure 4.2: **Network statistics of K3L2 as a function of sparsifying edge threshold.** As the edge threshold increase, the edge density of the graph reduces. In the beginning, the number of connected components remains one, indicating that the sparsified graph is connected, but becomes disconnected as the edge threshold increases beyond about 0.45.

4.3 Experiments

4.3.1 Synthetic datasets

We start by demonstrating the ability of *node2vec+* to identify potential *out* edges in weighted graphs using a barbell graph and the hierarchical cluster graphs. For simplicity, we fix $\gamma = 0$ for all experiments in this section.

4.3.1.1 Barbell graph

A barbell graph, denoted as B, is constructed by connecting two complete graphs of size 20 with a common bridge node (Figure 4.3a). All edges in B are weighted 1. There are three types of nodes in B, 1) the bridge node; 2) the peripheral nodes that connect the two modules with the bridge node;

3) the interior nodes of the two modules. By changing the *in-out* parameter q, *node2vec* could put the peripheral nodes closer to the bridge node or interior nodes in the embedding space.

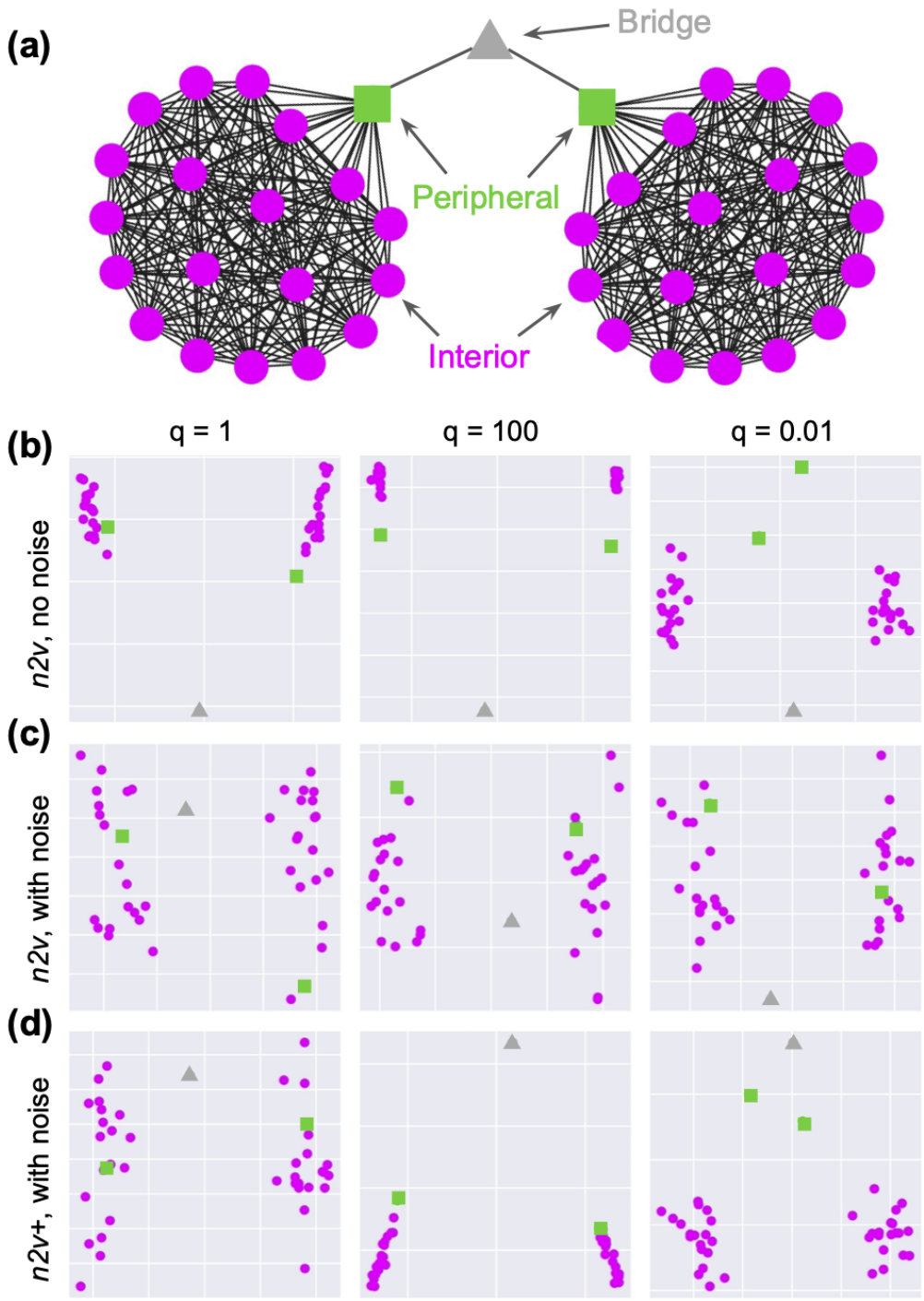


Figure 4.3: **Barbell graph embedding demonstration.** (a) Illustration of the barbell graph, and three different types of nodes indicated by the different marker styles. (b) Embedding of the barbell graph B using *node2vec*. (c-d) Embedding of the noisy barbell graph $\tilde{B}$ using *node2vec* and *node2vec+*, respectively. Each one of (b-d) contains three different settings of q: 1, 100, and 0.01.

When q is large, *node2vec* suppresses the *out* edges, e.g., an edge connecting a peripheral node to the bridge node, coming from an interior node. Consequently, the biased random walks are restricted to the network modules. In this case, the transition from the peripheral nodes to the bridge node becomes less likely compared to a first-order random walk, thus pushing the embeddings between the bridge node and the peripheral nodes away from each other. Conversely, when q is small, the transition between the peripheral nodes and the bridge node is encouraged. In this case, the embeddings of the bridge node and the peripheral nodes are pulled together. To see this, we run *node2vec* with fixed $p = 1$, and three different settings of $q = [1, 100, 0.01]$. Indeed, for $q = 100$, *node2vec* tightly clusters interior nodes and pushes the bridge node away from the peripheral nodes, and for $q = 0.01$, the peripheral nodes are pushed away from the interior nodes (Figure 4.3b). Since *node2vec* and *node2vec+* are equivalent when the graph is unweighted (see Method), we omit the visualization of *node2vec+* embeddings for B.

Next, we perturb the barbell graph by adding loose edges with edge weights of 0.1, making the graph fully connected. This perturbed barbell graph is denoted $\tilde{B}$. As expected, *node2vec* failed to make use of the q parameter (Figure 4.3c), since none of the edges are identified as an *out* edge. On the other hand, *node2vec+* can pick up potential *out* edges and thus qualitatively recovers the desired outcome (Figure 4.3d). Note that both *node2vec* and *node2vec+* have similar results for $\tilde{B}$ when $q = 1$. This confirms that *node2vec+* and *node2vec* are equivalent when p and q are set to neutral, corresponding to embedding with unbiased random walks. Finally, when using non-neutral settings of q, *node2vec+* is able to suppress some noisy edges, resulting in less scattered embeddings of the interior nodes (Figure 4.3d).

4.3.1.2 Hierarchical CLUSTER graph

We use a modified version of the CLUSTER dataset [64] to further demonstrate the advantage of the *node2vec+* due to identifying potential *out* edges. Specifically, the hierarchical cluster graph K3L2 contains $L = 2$ levels (3 including the root level) of clusters, and each parent cluster is associated with $K = 3$ children clusters (Figure 4.4a). There are 30 nodes in each cluster, resulting in a total of 390 nodes. To generate the hierarchical cluster graph, we first generate point clouds via

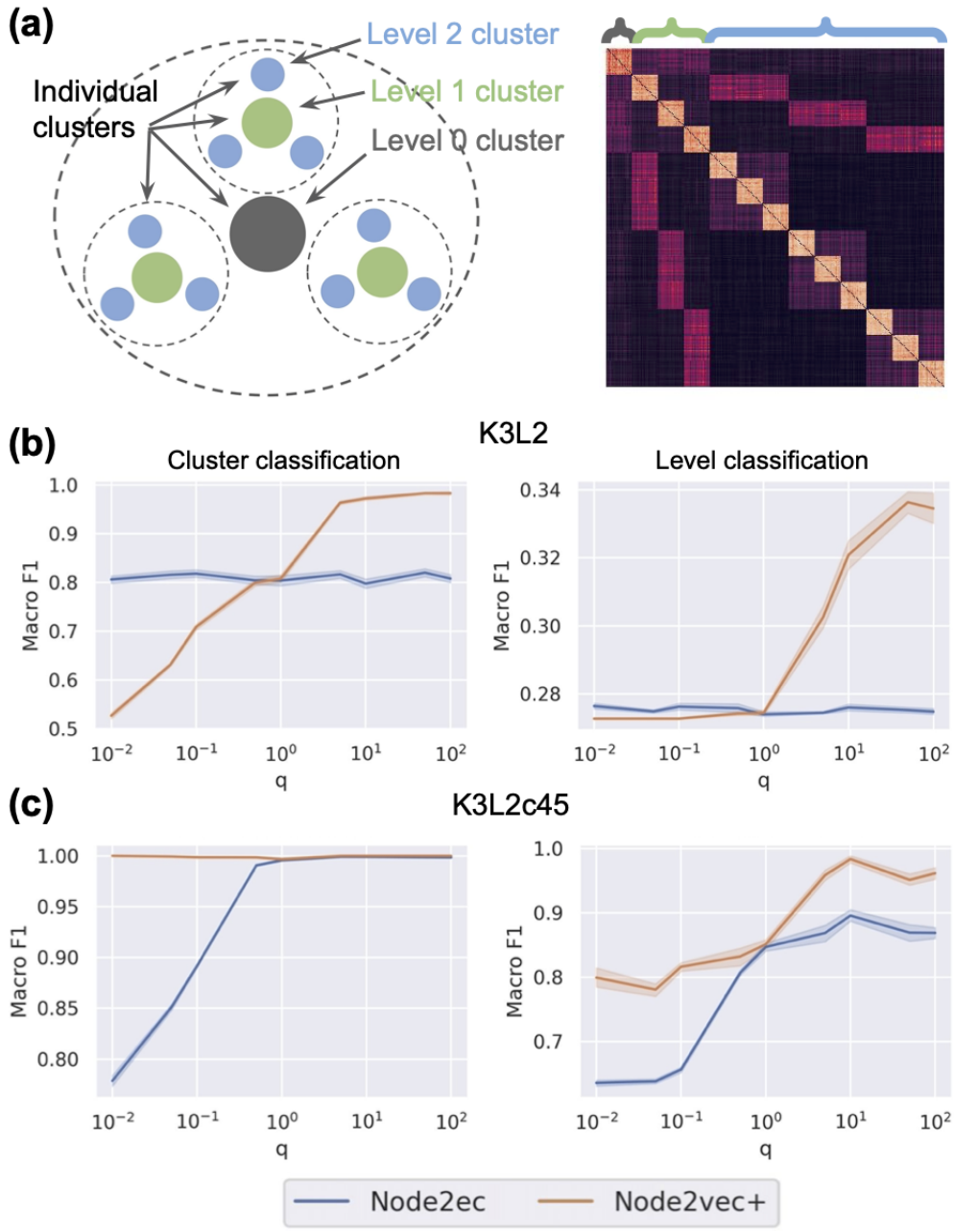


Figure 4.4: **Hierarchical CLUSTER graph classification task.** (a) Illustrations of the K3L2 hierarchical clusters. Left: top-down view of the clusters. Right: adjacency matrix of K3L2; colored brackets indicate the corresponding cluster levels of the nodes. (b) Classification evaluation on K3L2. (c) Classification evaluation on K3L2c45.

a Gaussian process in a latent space so that the Euclidean distance between two points from two sibling clusters is about twice ($\sqrt{2}$ to be precise) the expected Euclidean distance from one of the two points to a point in the parent cluster, which is set to be 1. The noisiness of the clusters is controlled

by the parameter σ, which is set to 0.1 by default. These data points are then turned into a fully connected weighted graph using a RBF kernel. We consider two different tasks (Figure 4.4a), (1) *cluster classification*: identifying individual cluster identity of each node in the graph, and (2) *level classification*: identifying the level to which the clusters correspond to. We split the nodes into 10% training and 90% testing and use the multinomial logistic regression model with l2 regularization for prediction. The evaluation process, including the embedding generation, is repeated ten times, and the final results are reported by Macro F1 scores.

As shown in Figure 4.4b, the performance of *node2vec* is not affected by the q parameter because the graph is fully connected. Meanwhile, *node2vec+* achieves significantly better performance than *node2vec* for large q settings for both tasks, demonstrating the ability of *node2vec+* to identify potential *out* edges and use this information to perform localized biased random walks. Similar results are observed on a couple of different hierarchical cluster graphs K3L3, K5L1, and K5L2 (Figure 4.5).

On the other hand, one might suspect that the issue with the fully connected graph can be alleviated by sparsifying the graph based on certain edge weight thresholds. Such an approach is widely adopted as a post-processing step for constructing functional gene interaction networks. Here, we show that even after sparsifying the graph aggressively, *node2vec+* still outperforms *node2vec*. In particular, we sparsify the K3L2 graph using the edge weight threshold 0.45, which is the largest value that keeps the graph connected. We then perform the same evaluation analysis above on this sparsified graph K3L2c45. In this case, *node2vec* indeed performs significantly better than before the sparsification for both tasks. Nonetheless, *node2vec+* achieves even better performance, still out-competing node2vec (Figure 4.4c).

Finally, we conduct a fine-grained evaluation analysis, showing that node2vec+ consistently outperforms node2vec under a wide range of conditions, including edge threshold, train-test ratio, and noise level (Figure 4.6).

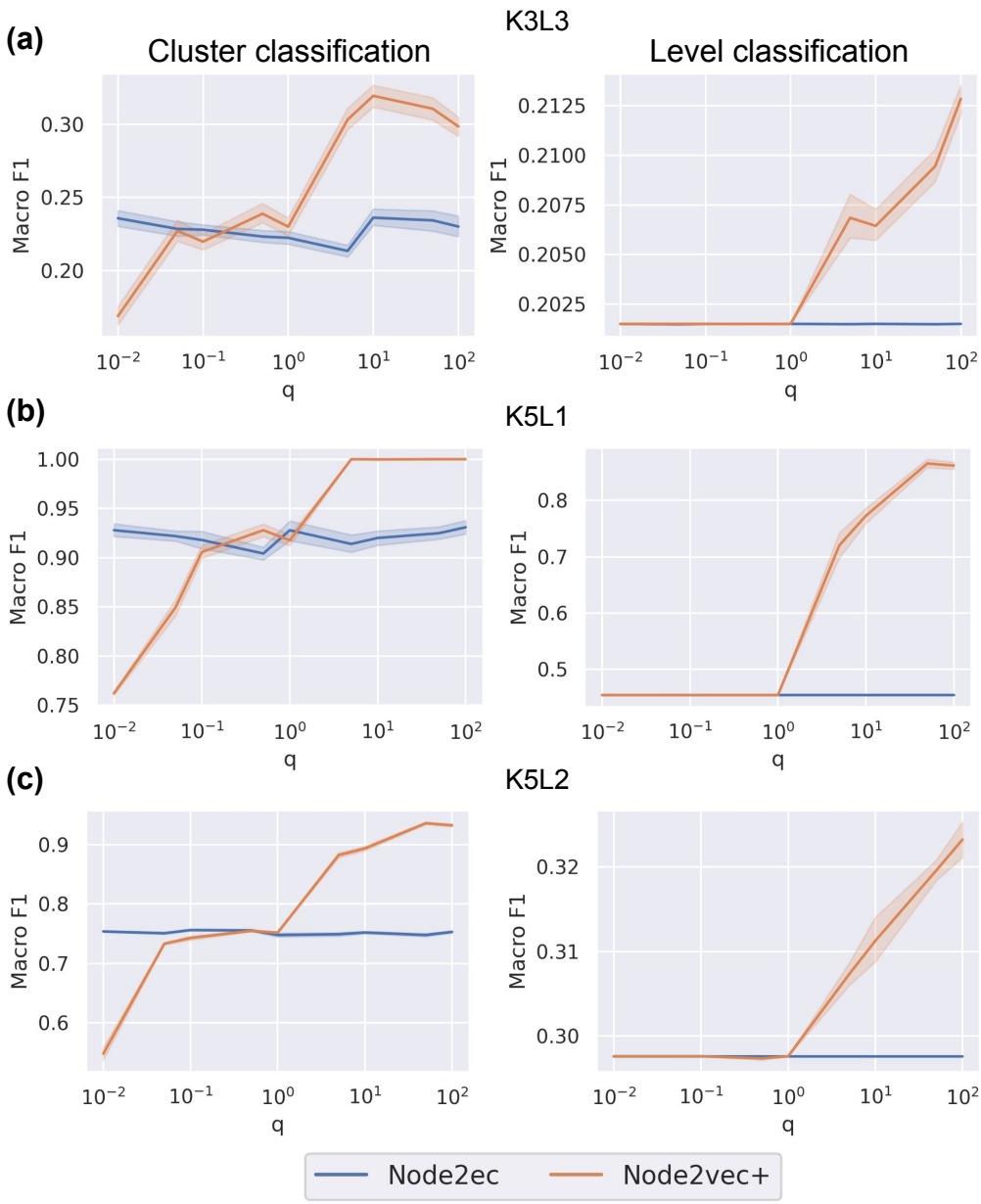


Figure 4.5: **Evaluation of other hierarchical cluster graphs: K3L3, K5L1, and K5L2.**

4.3.2 Real-world datasets

Our primary motivation for developing *node2vec+* stems from the fact that many functional gene interaction networks are dense and weighted. To systematically evaluate the ability of *node2vec+* to embed such biological networks, we consider various challenging gene classification tasks, including gene function and disease gene predictions. Furthermore, we devise experiments with previously benchmarked datasets BlogCatalog and Wikipedia [83] and confirm that *node2vec+* performs equal

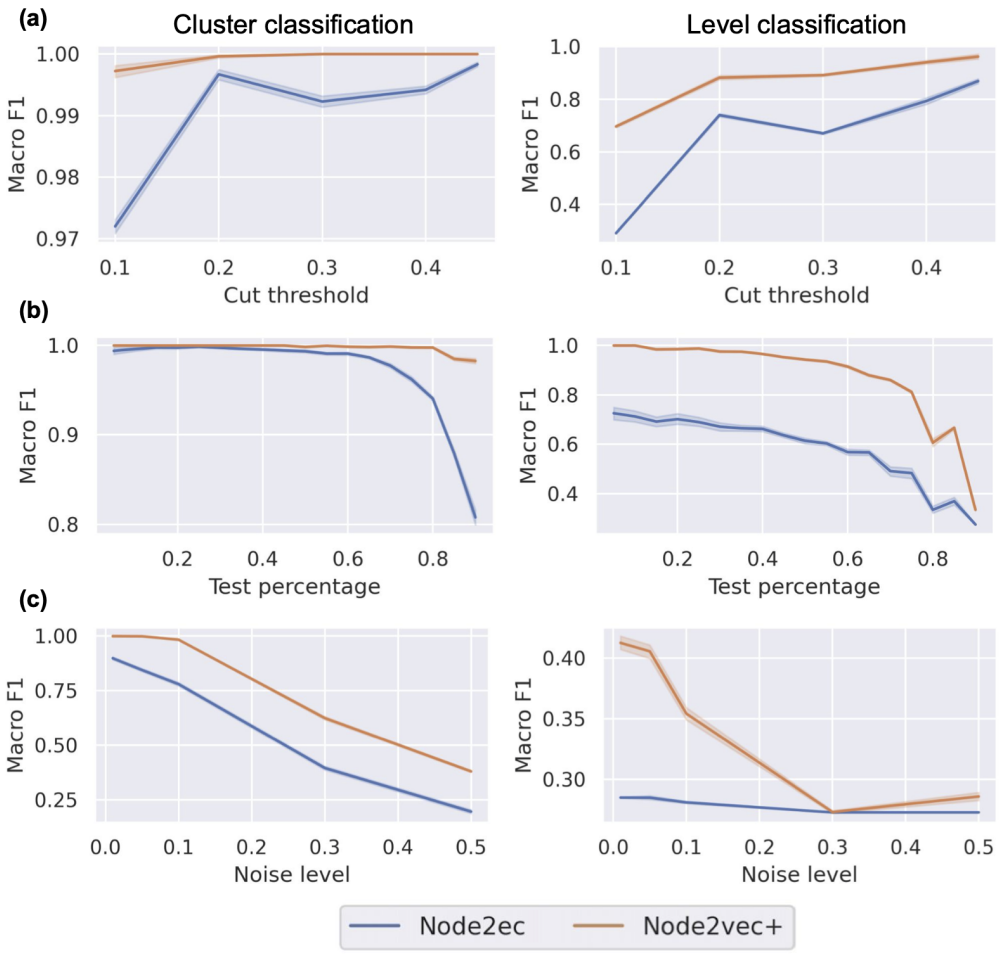


Figure 4.6: **Fine-grained analysis of K3L2.** (a) changing sparsification threshold value. (b) changing train/test raio, larger value means less training data. (c) changing noise level during network construction.

to or better than *node2vec*, depending on whether the network is weighted (Figure 4.7).

4.3.2.1 Datasets

Human functional gene interaction networks We consider functional gene interaction networks, which is a broader class of gene interaction networks that are routinely used to capture gene functional relationships.

- **STRING** [246] is an integrative gene interaction network that combines evidence of protein interactions from various sources, such as text-mining, high-throughput experiments, and etc.
- **HumanBase-global** is a tissue-naive version of the HumanBase [80] tissue-specific networks

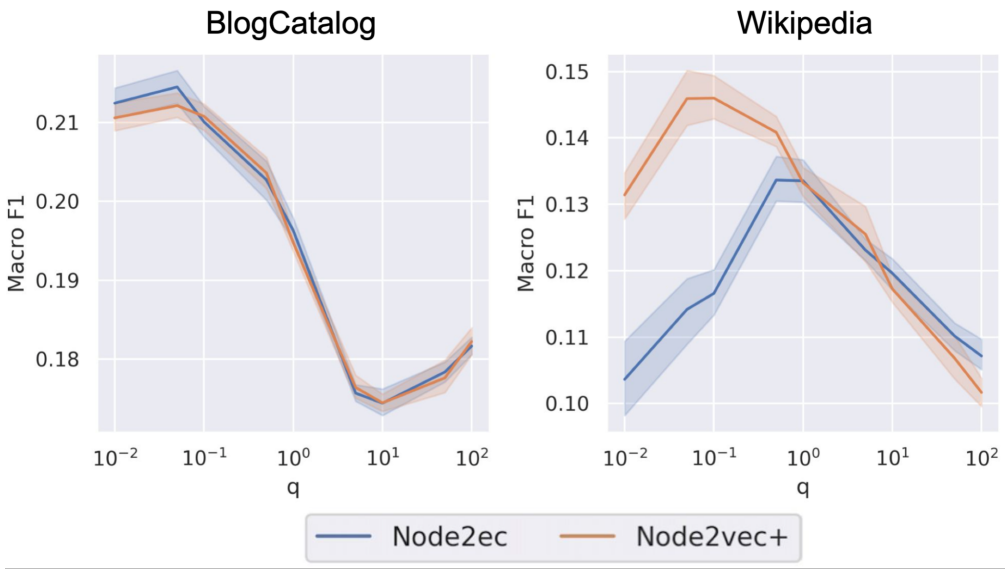


Figure 4.7: **Multi-label classification benchmarks using BlogCatalog and Wikipedia.**

(previously known as GIANT), which are constructed by integrating hundreds of thousands of publicly available gene expression studies, protein-protein interactions, and protein-DNA interactions via a Bayesian approach, calibrated against high-quality known functional gene interactions.

- **HumanBaseTop-global** is a sparsified version of HumanBase-global that eliminates all edges below the prior of 0.1.

Multi-label gene classification tasks We follow the procedure detailed in [153] to prepare the multi-label gene classification datasets. More specifically, we prepare two collections of gene classification tasks (each is called a gene set collection):

- **GOBP**: Gene function prediction tasks derived from the Biological Processes gene sets from the Gene Ontology [49].
- **DisGeNET**: Disease gene prediction tasks derived from the disease gene sets from the DisGeNET database [206].

After filtering and cleaning up the raw gene set collections, we end up with ~45 functional gene prediction tasks and ~100 disease gene prediction tasks (Table 4.1). These gene classification tasks

are challenging primarily due to the scarcity of the labeled examples, with on average 100 and 200 positive examples per task for GOBP and DisGeNET, respectively, relative to the (order of) tens of thousands of nodes in the networks.

We split the genes into 60% training, 20% validation, and 20% testing according to the level at which they have been studied in the literature (based on the number of PubMed publications associated with each gene). In particular, the top 60% most well-studied genes are used for training; the 20% least-studied genes are used for testing, and the rest are used for validation. For GNNs, we report the test scores at the epoch where the best validation score is achieved.

4.3.2.2 Baseline methods

Node embedding We use *node2vec* as our primary baseline for node embedding. We exclude several popular node embedding methods, such as DeepWalk [202], LINE [247], and GraRep [40], from our main analysis, as it has been shown previously in various contexts [83, 16, 291] that *node2vec* performs superior in the node classification settings.

Graph neural network We include two popular GNNs, GCN [124] and GraphSAGE [89] in our comparison. Both methods have shown exceptional performance on many node classification tasks, but their performance on the gene classification tasks here still needs to be better studied. For GraphSAGE, we consider the full-batch training strategy with mean pooling aggregation following the Open Graph Benchmark [99].

The basic GNN architecture consists of three main components: (1) the pre-message-passing (pre-mp) layer that map the initial node features to the hidden dimension, (2) the graph convolution layers, and (3) the post-message-passing (post-mp), or the prediction head, layer that maps the final node embeddings to the prediction values. In addition, the convolution layers have the option to add residual (skipsum) connections. We initialize the linear (pre-/post-mp) layers using Xavier uniform initialization [76]. Finally, to train the GNNs, we use the standard Adam optimizer [122] with a *reduce learning rate on plateau* scheduler, along with dropout and weight decay.

4.3.2.3 Experiment setup

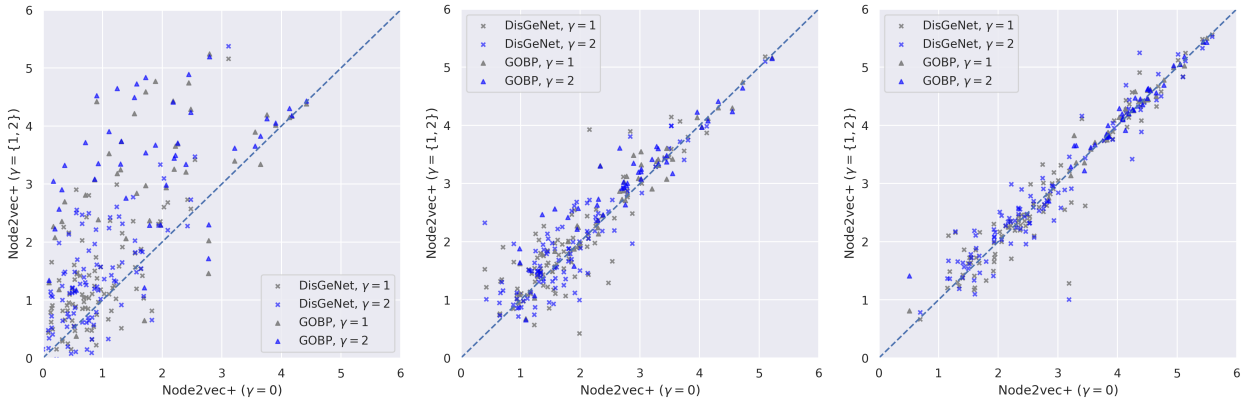
Evaluation metric Following [153], we use the $\log_2 \frac{\text{auPRC}}{\text{prior}}$ as our evaluation metric, which represents the $\log_2$ fold change of the average precision compared to the prior. This metric is more suitable than other commonly used metrics like AUROC as it corrects for the class imbalance issue that is prevalent in the gene classification tasks here, as well as emphasizes the correctness of top predictions.

Tuning embeddings parameters For *node2vec* and node2vec+, we train a one vs rest logistic regression with l2 regularization using the embeddings learned. The parameters for embeddings including dimension, window-size, walk-length, and number of walks per node are set to 128, 10, 80, and 10, respectively, by default. We tune the hyperparameters for *node2vec* (p, q) and for *node2vec+* (p, q, γ) via grid search using the validation sets. To keep the grid search budget comparable, we search p and q over $\{0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100\}^2$ for *node2vec* $(n = 81)$; we search p and q over $\{0.01, 0.1, 1, 10, 100\}^2$, together with $\gamma \in \{0, 1, 2\}$ for *node2vec+* $(n = 75)$.

Tuning GNN parameters For both GNNs, we train one model for each combination of a network and a gene set collection in an end-to-end fashion. The architectures are fixed to five hidden layers with a hidden dimension of 128. Since the gene interaction networks here do not come with node features, we use the constant feature for GCN and the degree feature for GraphSAGE, respectively. We use the Adam optimizer [122] to train the GNNs with $100,000$ max number of epochs. The learning rates are tuned via grid search from 10^{-5} to 10^{-1} based on the validation performance. The optimal learning rates that result in a decent convergence rate without diverging are 0.01 and 0.0005 for GCN and GraphSAGE, respectively.

Table 4.1: **Number of tasks (i.e., gene sets or node classes) for each combinations of network and gene set collection.** The number in the parenthesis is the average number of positive examples.

	GOBP	DisGeNET
HumanBase	46 (98.3)	103 (225.9)
STRING	41 (100.0)	97 (221.5)



(a) **HumanBase-global** (b) **HumanBaseTop-global** (c) **STRING**

Figure 4.8: **Comparison of different γ settings in *node2vec+*.** Each dot represents the testing performance ($\log_2 \frac{\text{auPRC}}{\text{prior}}$) of a specific gene set, with optimally tuned p and q settings.

4.3.2.4 Experimental results

Tuning γ significantly improves performance for dense graph The γ parameter in *node2vec+* (Chapter 4.2.2) controls the threshold of distinguishing *in* edges and *out* edges. A small or negative valued γ considers most non-zero edges as *out* edges. Conversely, a large valued γ identifies less *out* edges. When the input graph is noisy and dense, assigning a larger γ (e.g., 1) can act as a stronger denoiser to suppress spurious *out* edges. Figure 4.8a compares the gene classification test performance between $\gamma = 0$ and $\gamma = \{1, 2\}$ with optimally tuned p, q using the HumanBase-global network. Higher testing scores are achieved by larger γ settings, illustrating that, to properly "denoise" the fully connected weighted graph HumanBase-global, we need to increase the noisy edge thresholds. On the contrary, the difference in performance due to the γ settings is less pronounced for sparse networks like HumanBaseTop (Figure 4.8b) and STRING (Figure 4.8c).

GNN methods performs worse than *node2vec(+)* In all settings, *node2vec+* significantly outperforms both GNN methods (Figure 4.9). Notably, for the STRING network, both *node2vec* and *node2vec+* outperform the two GNNs by a large margin. The sub-optimal GNN performance here illustrates that, despite being powerful neural network architectures that can leverage the graph structures, GNNs alone cannot learn effectively given a limited number of labeled examples. On the contrary, the embedding processes of *node2vec(+)* are task agnostic and can be carried out effectively

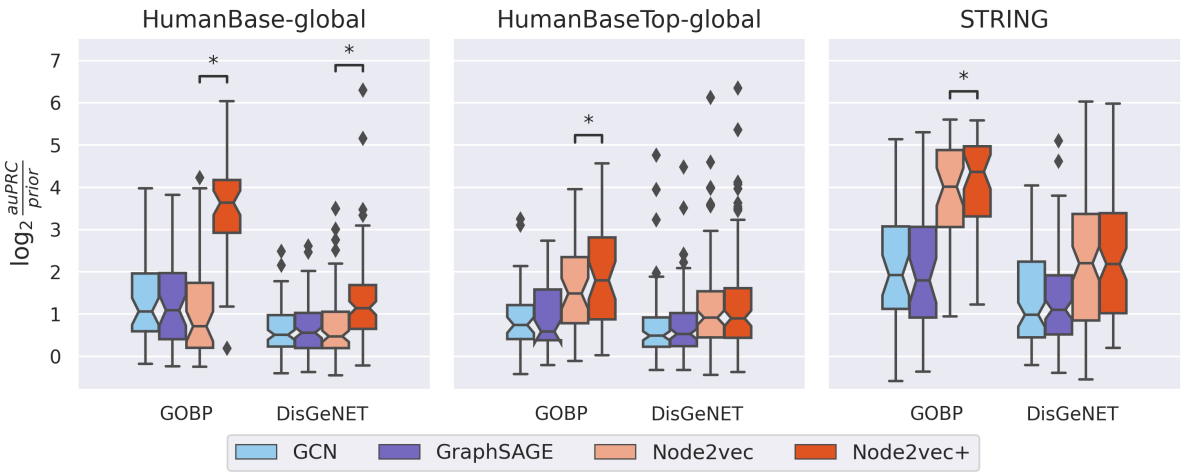


Figure 4.9: **Gene classification tasks using protein-protein interaction networks.** Each panel corresponds to a specific protein-protein interaction network (HumanBase-global, HumanBaseTop-global, and STRING). Each point in a boxplot represents the final test score for a specific task (gene set) in the gene set collection (GOBP or DisGeNET). Starred (*) pairs indicate that the performance between *node2vec* and *node2vec+* are significantly different (Wilcoxon p-value < 0.05).

without labels. These results indicate that gene classification tasks based on gene interaction network are more effectively solved by unsupervised shallow embedding methods than GNNs.

nod2vec+* matches or outperforms *node2vec *node2vec+* significantly outperforms *node2vec* (Wilcoxon paired test [274] p-val < 0.05) except for the DisGeNET tasks using HumanBaseTop-global and STRING networks, in which cases the two methods perform equally (Figure 4.9). The performance differences are especially pronounced when using the fully connected and noisy HumanBase-global network, demonstrating *node2vec+*'s ability to learn robust node representations in the presence of noise. Nevertheless, when the network is less dense (e.g., HumanBaseTop-global), *node2vec+* is still able to perform at least as well as *node2vec*, indicating that *node2vec+* is overall a good replacement of *node2vec*.

4.3.2.5 Tissue-specific functional gene classification

A key feature of functional gene interaction networks constructed using gene expression data is capturing biological context specificity, such as tissue-specificity provided by the HumanBase networks. Thus, we further demonstrate the use case of *node2vec+* using tissue-specific functional gene classification tasks derived from [297]. After processing, there are 25 tissue-specific functional

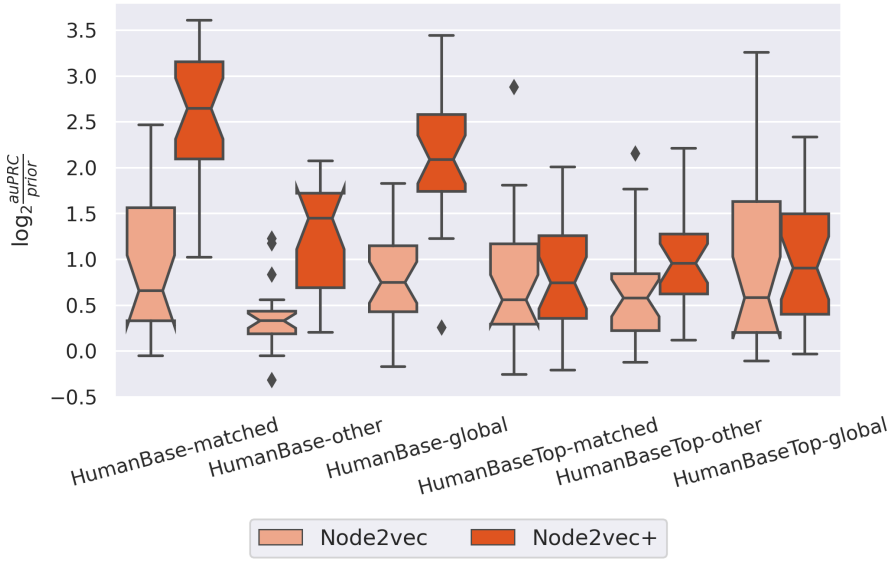
gene classification tasks, with 12 different tissues found in the HumanBase database. We follow a similar experimental setup as above, and for each tissue-specific functional gene classification task, we report the followings: (1) *matched*: the prediction performance using the corresponding tissue-specific network; (2) *other*: the average prediction performance using tissue-specific networks other than the corresponding tissue; (3) *global*: the prediction performance using the tissue-naive network.

Figure 4.10a shows that *node2vec+* outperforms *node2vec* in most scenarios, especially when using the full HumanBase networks. In particular, *node2vec+*, using the *matched* tissue-specific full networks for the given functional gene classification tasks, results in significantly better performance than using *other* (unrelated) tissue-specific networks, as well as the *global* (tissue-naive) network. On the contrary, *node2vec* cannot fully utilize the tissue-specific networks, as indicated by the lack of difference in performance between *matched* and *global* networks.

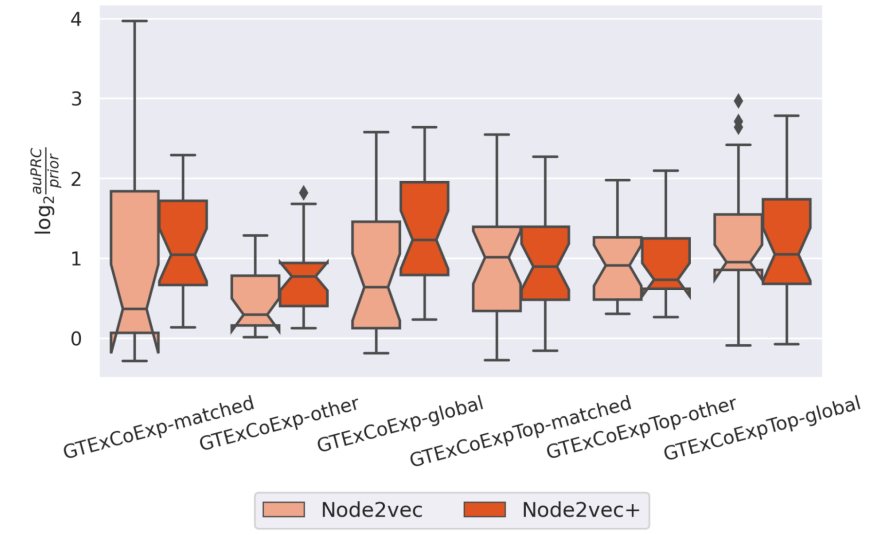
We observe similar results using another collection of tissue-specific co-expression networks, **GTExCoExp**, that are generated using a benchmarked co-expression network generation workflow by [113] (Figure 4.10b). The main difference is that, in this case, using the *global* network achieves slightly better performance than using the *matched* tissue-specific networks.

4.4 Conclusion

We have proposed *node2vec+*, a natural extention of *node2vec*, which improved the second-order random walk on weighted graphs by accounting for edge weights. We demonstrated that the corresponding node embeddings are improved whenever the *in-out* walks positively influence the task (meaning that the optimal q setting is not 1). Particularly, we showed that *node2vec+* better identifies potential out edges on weighted graphs than *node2vec* using the synthetic barbell graph and the hierarchical cluster graph datasets. Evaluations on various challenging gene classification tasks implied the superiority of *node2vec(+)* over *node2vec*, and even over more powerful graph neural networks.



(a) **HumanBase**



(b) **GTExCoExp**

Figure 4.10: **Tissue-specific functional gene classification performance.** Comparison between node2vec and node2vec+ for their tissue specific GOBP gene annotation capabilities using tissue-specific networks.

CHAPTER 5

CONE: CONTEXT-SPECIFIC NETWORK EMBEDDING VIA CONTEXTUALIZED GRAPH ATTENTION

Human gene interaction networks, commonly known as interactomes, encode genes' functional relationships, which are invaluable knowledge for translational medical research and the mechanistic understanding of complex human diseases. Meanwhile, the advancement of network embedding techniques has inspired recent efforts to identify novel human disease-associated genes using canonical interactome embeddings. However, one pivotal challenge that persists stems from the fact that many complex diseases manifest in specific biological contexts, such as tissues or cell types, and many existing interactomes do not encapsulate such information. Here, we propose CONE[1], a versatile approach to generate context-specific embeddings from a context-free interactome. The core component of CONE consists of a graph attention network with contextual conditioning, and it is trained in a noise-contrastive fashion using contextualized interactome random walks localized around contextual genes. We demonstrate the strong performance of CONE embeddings in identifying disease-associated genes when using known biological contexts associated with the diseases. Furthermore, our approach offers insights into understanding the biological contexts associated with human diseases.

5.1 Introduction

The proper operation of cells depends on the precise coordination and interaction of biological entities, such as genes, RNA, and proteins. As a result, complex human diseases are the ramifications of perturbations to groups of genes that give rise to pathological states [20, 262]. Leveraging this interdependence among biological entities, network-based methods have shown great promises in unveiling human genes' function [154] and their associated diseases [129, 287]. Recent approaches achieved this by training machine learning models using network embeddings extracted from the input gene network [291, 154, 264, 263]. However, a crucial limitation remains as many biological network embedding methods do not consider the differences induced by various biological contexts.

[1] https://github.com/krishnanlab/cone

The interacting relationships among genes vary across biological contexts, such as tissues, cell types, or disease states. Many human genes operate in a tissue-dependent manner. For example, *DMD* is preferably expressed in muscle [63]. The heterogeneity of the specific set of genes expressed by a particular biological context contributes significantly to the phenotypic diversity within an organism, aiding the complex, specialized functions required for its survival [27]. Consequently, the dysfunction of genes ultimately leads to diseases manifesting only in specific tissues. For example, Mendelian disorders show clear tissue-specific manifestation and complex diseases have a strong tendency of tissue selectivities, such as neurological disorders and cardiovascular diseases [93]. However, tissue-disease associations are not always straightforward. Apart from the primary affected tissues, diseases may affect seemingly unrelated tissues. One prime example is the high risk of gastrointestinal tract dysfunction observed in patients of Parkinson's disease, a neurological disorder primarily centered in the brain [90]. The cryptic connections between diseases may be partially explained by shared underlying mechanisms among them, which could be well characterized by networks [47].

Numerous functional genomics projects generate data comprising diverse types, qualities, and scopes of genes or molecules [189, 245, 121]. To obtain a high-quality and comprehensive network embedding, several methods are developed to infer a joint network representation by integrating multiple networks [75, 71, 54, 46, 275]. However, a drawback of network integration methods is that the integration process can eliminate context-specific information in each input network, resulting in a context-naive network. In other words, it may assume the same molecular interactions in the kidney and brain, whereas, in reality, interactions are tissue-specific. To predict a range of tissue-specific gene functions or gene-disease relationships, we must integrate context-specific information into the network. Furthermore, state-of-the-art data integration approaches using graph neural networks [71] may not scale well to the number or size of the networks. Therefore, we require a scalable method that can handle networks of varying sizes and numbers.

Contributions Here, we address the critical need for a versatile and scalable method for generating biological context-specific network embeddings. We summarize our main contributions as follows.

1. We propose CONE, a versatile contextual network embedding method that takes context definitions in the form of node sets.

2. The proposed method operates on a shared graph attention network across all contexts, which is contextualized by conditioning on the raw embeddings. This results in a model that scales practically independent of the number of contexts.

3. Through a series of experiments, we demonstrate the value of injecting various biological contexts to improve disease gene prioritization.

Related work A few studies have explored the idea of contextualizing biological network embeddings using contexts such as tissue or cell type specificity. Notably, OhmNet [297] pioneered the tissue-specific gene interaction network embedding by leveraging the hierarchical relationships between different tissue levels and genes. OhmNet learns a multi-layer embedding and operates on the idea that closely related tissues, or layers, should have similar embeddings. However, the original OhmNet method requires a highly specific construction of the hierarchical multi-layer tissue-specific networks, making it hard to extend to broader biological contexts readily. More recently, PINNACLE [140] further expanded the biological contexts into finer-grain definitions based on cell types using cell-type-expressed genes constructed from the Tabula Sapiens single cell atlas [50]. Furthermore, PINNACLE learns context-specific graph attention modules with independent parameters per context, leading to poor scalability, as each new context requires its own model. CONE, on the other hand, provides a versatile approach that is also scalable with respect to the number of contexts.

5.2 Preliminaries

5.2.1 Network Embedding via Sampling

Let $G = (V, E, w)$ be a weighted undirected graph, where the edge weight function $w : V \times V \rightarrow \mathbb{R}$, and denote its corresponding adjacency matrix by $\mathbf{M} \in \mathbb{R}^{|V| \times |V|}$. The goal of a graph embedding method aims to find the mapping $f : V \rightarrow \mathbb{R}^d$ that maps each node $v \in V$ to a d-dimensional

embedding space by minimizing the following objective function.

$$\min_{\hat{f}} \mathcal{L}_{G,S^+,S^-}(\hat{f}) \tag{5.1}$$

where S^+ and S^- are positive and negative edge sampling functions. Particularly, the Singular Value Decomposition (SVD) can be viewed as the optimization of equation 5.1, where $S^{+,-}$ are both uniform sampler on all pairwise entries in the adjacency matrix, f maps to the left (f_L) and right (f_R) embedding representations [1]. The loss function is the squared error between the inner product of the left and right embeddings of the two nodes and the edge weight between them in the graph:

$$\mathcal{L}^{\text{SVD}} = \mathbb{E}_{(u,v)\sim|V|\times|V|}\Big(\langle f_L(u), f_R(v)\rangle - \mathbf{M}_{u,v}\Big)^2 \tag{5.2}$$

Random Walk Sampling Random walk on graphs have been studied extensively, with many applications spanning social network analysis, information retrieval, and so on. In our framework, the random walk procedure can be seen as the node-pair sampling function. For instance, *node2vec* [83] with negative sampling can be reformulated in a noise contrastive fashion [85] as:

$$\mathcal{L}^{\text{RW}} = -\mathbb{E}_{(u,v)\sim S^+(G)} \log\Big(\sigma\big(\langle f(u), f(v)\rangle\big)\Big) - k\mathbb{E}_{(u,v)\sim S^-(G)} \log\Big(1 - \sigma\big(\langle f(u), f(v)\rangle\big)\Big) \tag{5.3}$$

where σ is the sigmoid function, k is the number of negative samples, and the positive sampling is achieved by a sliding window over a second-order biased random walk [83]. We refer to the above as the *random walk loss*.

5.2.2 Graph Attention Neural Network (GAT)

Graph neural network (GNN) is a special type of neural network architecture that operates on the underlying graph structure. It does so by iteratively aggregating information from each node's neighborhood and transforming the aggregated representations [279, 295]. Particularly, GAT [258, 32] uses an attention mechanism to weight each node's neighborhood for aggregation, and the (pre-activation and pre-normalization) layer updating rule is written as follows.

$$h'(u) = \sum_{v\in\mathcal{N}(u)} \alpha_{u,v}\mathbf{W}h_v \tag{5.4}$$

where $\alpha_{u,v}$ is the attention score, $\mathbf{W} \in \mathbb{R}^{d \times d}$ is a learnable linear transformation. In practice, we use the v2 corrected attention proposed in [32]:

$$\alpha_{u,v} = \text{softmax}_v\left(a^\top \text{LeakyReLU}(\mathbf{W}[h(u)||h(v)])\right) \tag{5.5}$$

5.3 Methods

We are interested in learning a *collection* of network embeddings, each specific to a biological context. For example, we can use heart-specific gene embeddings to unravel more tissue-specific genes related to cardiovascular diseases. Contextualizing gene embeddings to biological contexts this way allows us to unveil nuanced relationships between diseases and biological contexts, such as tissues, cell types, and other diseases or traits. The full pipeline of our approach is depicted in Figure 5.1.

From a high level, CONE contains two main components, including (1) a GNN decoder and (2) an MLP context encoder. The GNN decoder converts the raw and learnable node embeddings into the final embeddings. On the other hand, the MLP context encoder projects the context-specific similarity profile that describes the relationships among different contexts (Chapter 5.3.2) into a condition embedding. When added with the raw embeddings, the condition embedding serves as a high-level contextual semantics, similar to the widely-used positional encodings in Transformer models [255]. The embeddings are trained using the losses based on the random walk on the context-specific subgraphs. We employ a straightforward approach to define a context-specific subgraph as the subgraph induced by the genes relevant to that context. Next, we formally describe our approach.

5.3.1 Contextualized network embeddings

Let $\mathcal{C} = \{C_i\}_{i \in 1,\ldots,n_c}$ be a collection of n_c contexts, where each context $C_i \subset V$ is a subset of nodes that defines the local context. We aim to learn a collection of embedding functions $\mathcal{F} = \{f_C\}$ by minimizing the loss function

$$\mathcal{L}_{tot} = \mathcal{L}^{RW}(f_0) + \mathbb{E}_{C \sim \mathcal{C}} \mathcal{L}_C^{RW}(f_C) \tag{5.6}$$

where f_0 is the context-naive embedding that is optimized against the whole network G, and f_C is the context-specific embedding that is optimized against the contextual random walk loss $\mathcal{L}_C^{RW}$ that samples random walks on the subgraph induced on the context set C, that is, $G(C) = (C, \{(u, v) \in E : u, v \in C\}, w)$. Equation 5.6 aims to simultaneously optimize for the global and local contextualized representation of all the nodes v in the network.

A naive attempt for obtaining the contextualized embeddings in equation 5.6 would be to learn independent f_C on the corresponding contextual graph $G(C)$. However, the resulting context-specific embeddings may completely lose the global information of the graph, since each f_C operates independently. We provide empirical evidence for this in Chapter 5.5.6.3.

5.3.2 Contextualized GAT

To address the above-mentioned problem, we propose to learn a *shared embedding encoding model* using GAT, and contextualize different embeddings by conditioning on the raw embedding matrix.

Let $g_\theta : \mathbb{R}^{|V| \times d} \rightarrow \mathbb{R}^{|V| \times d}$ be a GAT network parameterized by θ, and $\mathbf{Z} \in \mathbb{R}^{|V| \times d}$ the raw embedding matrix that is randomly initialized. Drawing parallels from recent work on conditional generation [220], we view context-specific embeddings as generation conditioned on a specific context, and propose to compute the contextualized embedding f_C as

$$f_C^{\text{CONE}} = g_\theta(\mathbf{Z} + \phi(C)) \tag{5.7}$$

where $\phi(C) \in \mathbb{R}^{1 \times d}$ is the context condition embedding that defines the context C. The context-naive embeddings are computed by passing the raw embedding alone through the GAT encoder: $f_0^{\text{CONE}} = g_\theta(\mathbf{Z})$. Finally, to form the full context-specific embedding for downstream evaluation, we concatenate it with the context-naive embedding and then project it down to d-dimension via PCA.

Context condition embedding The context condition embedding serves to provide low level semantics about each context, and two contexts with highly overlapping sets of nodes should have similar condition embeddings. To that end, we design an approach to encode condition embedding using the context similarity matrix $\mathbf{J} \in \mathbb{R}^{n_c \times n_c}$ constructed by taking the Jaccard index between

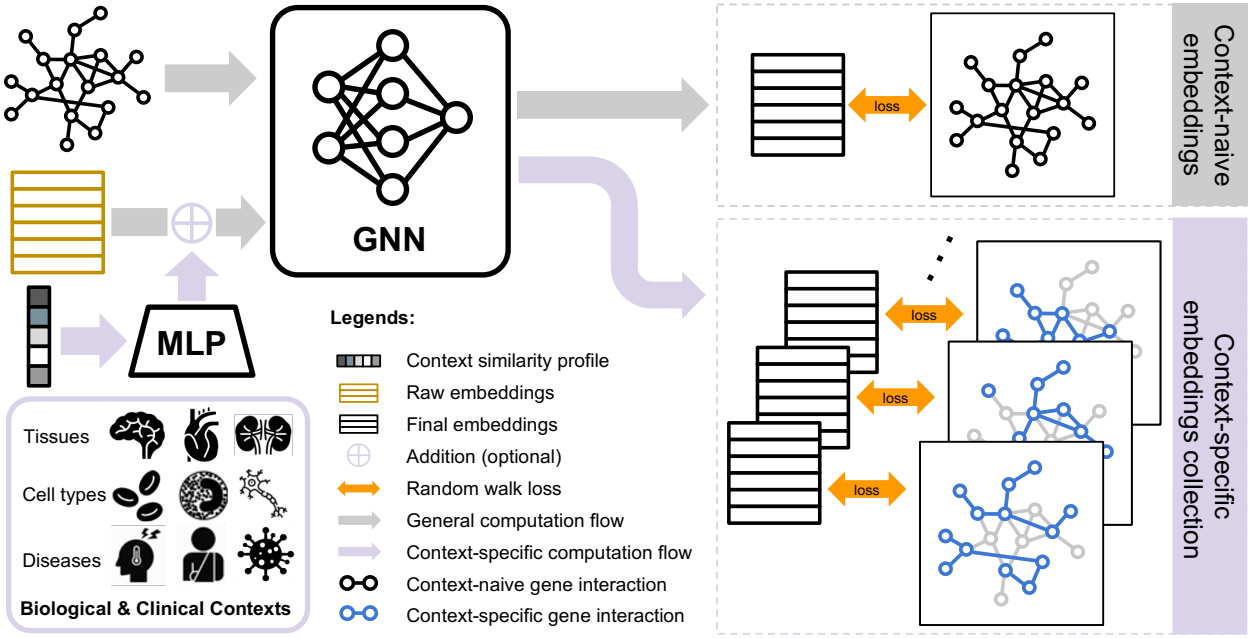


Figure 5.1: **Overview of CONE embedding collection training and inference.**

all pairwise contexts, $\mathbf{J}_{i,j} = \frac{|C_i \cap C_j|}{|C_i \cup C_j|}$. Finally, we use a two-layer Multi-Layer Perceptron (MLP) to project $\mathbf{J}$ into the condition embeddings, thus for each context C_i, its corresponding condition embedding is computed as $\phi(C_i) = \text{MLP}(\mathbf{J})_{[i,:]}$.

5.3.3 Training CONE

The loss function defined in equation 5.6 is implemented in practice by alternating between the context-naive random walk loss $\mathcal{L}^{RW}(f_0)$, and the context-specific random walk loss $\mathcal{L}^{RW}(f_C)$ for a randomly drawn context. We train the model for 120 epochs using the AdamW [158] optimizer with a constant learning rate of 0.001 and a weight decay of 0.01. We optimize CONE's hyperparameters using the primary main DisGeNET benchmark (Chapter 6.3) based on the averaged validation APOP scores of the context-naive CONE embeddings. The hyperparameter grid is shown in Table 5.1, where the final hyperparameter settings are bolded.

5.3.4 Complexity analysis

As the main module of CONE, GAT has the computational complexity of $\mathcal{O}(|V|d^2 + |E|d)$ [258, 32]. In addition, the context condition embedding encoder ϕ scales linearly with respect to the number of conditions as $\mathcal{O}(n_c d)$. However, in practice, since $n_c \ll |E|$, the computational

Table 5.1: **Hyperparameter search grid.** Final settings are **bolded**.

	Hyperparameter	Search grid
Architecture	Number of layers	[1, **2**, 3]
	Number of heads	[1, 2, **3**, 4, 5, 6]
Optimization	Learning rate	[0.01, 0.005, **0.001**, 0.0005, 0.0001]
	Weight decay	[1, 0.5, **0.1**, 0.05, 0.001, 0.0005, 0.0001]
Random walk	Walk length	[40, 80, **120**, 160, 200]
	Walks per node	[1, 5, **10**, 15, 20]
	Window size	[5, **10**, 15, 20, 25]

complexity of CONE should be equivalent to that of a single GAT network. This effective constant scaling with respect to the number of contexts is in stark contrast with the recently proposed method PINNACLE, which scales linearly with respect to the number of contexts due to the implementation of independent GAT module for each context. We provide empirical evidence for the scalability of CONE in Appendix 5.5.6.1.

5.4 Experiment setup

We devise diverse biomedical tasks to evaluate the capability of CONE against baseline methods to prioritize genes in the gene interaction network. These tasks are multi-label classification tasks, where the goal is to identify human genes that are related to certain diseases using the gene network embeddings generated by the models. We conduct our main analysis using the PINPPI network, which is a combined network using BioGRID [240], Menche [172], and HuRI [160], provided by the PINNACLE [140] paper. Specifically, we obtain the raw PINPPI network from the PINNACLE paper[2], which contains 15,461 nodes and 207,641 edges. We then convert the node IDs from gene symbol to Entrez ID [163] using the MyGeneInfo query service [277]. We only perserve genes that have exact one-to-one mapping from gene symbol to Entrez ID. After the above conversion, the final processed Entrez based PINPPI network contains 15,229 nodes and 206,835 edges.

For gene label information, we collect the two therapeutic target tasks (RA and IBD) from PINNACLE. Furthemore, we compile a comprehensive collection of disease-gene annotations from

[2]https://figshare.com/articles/software/PINNACLE/22708126

DisGeNET [206], following the processing steps detailed in Chapter 2.2.3. After filtering out diseases with less than ten positive genes intersecting with the PINPPI network, the final DisGeNET benchmark contains 167 diverse human diseases.

For each DisGeNET disease gene prioritization tasks, we randomly split the positive and negative genes into 6/2/2 train validation test sets. The final prediction performance are reported as the average test scores across five different random splits. For RA and IBD, we use the pre-defined train test split given by PINNACLE. Detailed dataset statistics are provided in Table 5.2

Table 5.2: **Gene set statistics.** First three gene set collections are used as prediction tasks, and the remaining three gene set collections are used as contexts.

	Collection	Type	# gene sets	Min.	Lower quart.	Med.	Upper quart.	Max.
Tasks	DisGeNET [206]	Disease	167	10	13	21	39	189
	IBD [140, 187]	Therapeutic Target	1			151		
	RA [140, 187]	Therapeutic Target	1			113		
Contexts	GTEx [157]	Tissue	30	430	1122.5	1593.5	3091	8604
	Tabula Sapiens [50, 140]	Cell type	156	1002	2012	2751	3051	3425
	CREEDS [271]	Disease	333	204	535	566	998	4452

5.4.1 Baseline methods

node2vec is a strong baseline method for network embedding-based gene prioritization method with superior performance on various benchmarks [16]. We also include embeddings generated by a two-layer GAT (v2) network [258, 32] trained in a standard graph autoencoder style [123] as a more direct baseline against CONE. Moreover, BIONIC [71] and Gemini [275] are two recent approaches that learn an integrated embedding across a collection of networks. We use them to test if embedding multiple context-specific subgraphs together gives an advantage over embedding a single context-naive network. All baselines and the CONE embeddings are evaluated in an *unsupervised* setting, where an ℓ_2 regularized logistic regression model is trained for each task using the embeddings that are learned without accessing any label information.

For context-specific network embeddings, we consider a recently proposed method, PINNACLE [140], which learns separate GAT modules for each context. We directly use the context-specific embeddings provided by the paper [3] to reanalyze the performance under our fair setting. We point

[3] https://figshare.com/articles/software/PINNACLE/22708126

out that PINNACLE context-specific embeddings differ slightly from CONE in that PINNACLE only generates embedding for the context-specific nodes. Conversely, CONE generates embeddings for all nodes regardless of if they are specific to the context. This enables us to evaluate all context-specific embeddings fairly across diverse disease gene prioritization tasks. Due to this limitation of PINNACLE, we exclude it from the primary DisGeNET gene prioritization benchmark. We set the embedding dimensions to 128 for all models.

5.4.2 Context-specific gene classification tasks

We primarily consider tissue-specificity for contextualizing the network embeddings. This leads to a natural understanding of the downstream disease gene prediction performance based on the association between tissues (contexts) and diseases (tasks). One widely adopted way of defining tissue- or cell type-specific genes is by *differential gene expression* [248], where genes that express significantly more in one context than others are considered relevant to the given context. Following this, we first obtain tissue-specific gene expression from the GTEx project [157], and then extract tissue-specific genes by taking genes with z-scores greater than one across tissues. In addition to tissue-specific genes, we also showcase CONE using other biological contexts, including cell types and diseases.

5.4.3 Evaluation metrics

We use the $\log_2$ fold-change of the average-precision over the prior (APOP) as the main metric for evaluating disease gene prioritization performance [154]. The area under the precision recall curve, which is closely related to the average precision, is a better choice for evaluating tasks with severe class imbalance [226]. The prior division, on the other hand, corrects for the expectation of the random guesser performance on different tasks with different number of positive examples. For the RA and IBD therapeutic target prediction tasks, we follow PINNACLE and report the average precision and recall at rank five (APR@5) in addition to APOP.

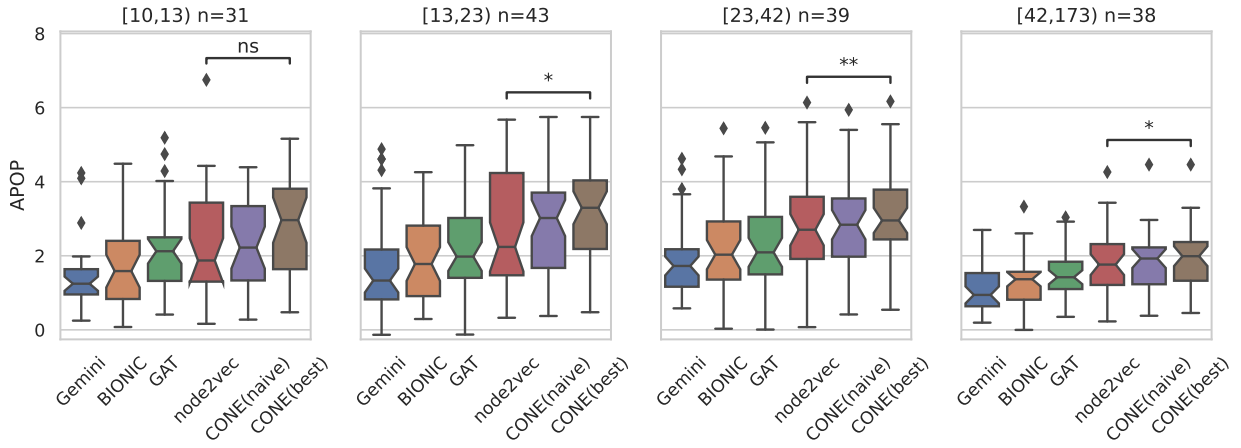


Figure 5.2: **DisGeNET disease gene predictions performance comparison between node2vec and CONE embeddings**. Each point in the box plot corresponds to the prediction test performance of a disease averaged across five random splits. Different panels show groups of diseases with different number of positive genes. For example, the left-most panel contains 31 diseases with at least 10 but less than 13 positive genes. ns, *, and ** indicate the significance level of the paired Wilcoxon test between the baseline *node2vec* and CONE (ns: not significant, *: $0.01 <$ p-value ≤ 0.05, **: p-value < 0.01).

5.5 Results and discussions

5.5.1 Context-specific embedding improves disease gene prediction performance

We first observe that, overall, picking the best context for each disease achieves noticeable performance improvement over the context-naive CONE embeddings, as indicated by the good performance of CONE (best) in Figure 5.2. Moreover, the advantage of using context-specific embedding is most apparent when the number of positive genes available for the disease is lacking, which might be attributable to the fact that diseases with more associated genes are more likely to contain more ubiquitous and less context-specific genes, and consequently reducing the effectiveness of using context-specific embeddings. In all cases, we note that either the context-naive or the context-specific CONE embedding consistently matches the performance achieved by the *node2vec* baseline.

5.5.2 CONE infer biologically relevant contexts for human diseases

Despite the performance improvement due to the CONE contextualized embeddings, it is still unclear whether the biological contexts that led to good performance on a particular disease are

associated with that disease. To address this question, we manually inspect six diseases where the connection between tissue and disease manifestation appears readily evident, as shown in Table 5.3. We hypothesize that the CONE embeddings derived from the disease-related tissue should have a higher APOP compared to the context-naive CONE or node2vec embeddings. Indeed, many of the top contexts found by CONE are biologically meaningful, whether as one of the main affected tissues or a related tissue. For example, the top-performing contexts for subvalvular aortic stenosis and familial bicuspid aortic valve, both diseases in which there is a problem with the aortic valve, which is the valve of an artery in the heart, included the artery for subvalvular aortic stenosis and heart for familial bicuspid aortic valve. More subtle top-performing but biologically relevant contexts are small intestine and adipose for hypochromic microcytic anemia. The cause of hypochromic microcytic anemia is typically decreased iron reserves in the body. This may be due to decreased dietary iron, poor absorption of iron from the gut, and acute or chronic blood loss [44]. Iron absorption is primarily carried out by cells in the small intestine [209], explaining why it would be a top context for anemia. Obesity, which is an excessive accumulation of adipose, has also been molecularly linked to iron deficiency in a way that shows the conditions mutually affect one another [3]. Also of note is that the primary tissue affected by pure red-cell aplasia, blood, was not identified in the top three contexts. However, patients with hepatitis, a symptom of liver inflammation, sometimes develop pure red-cell aplasia [145, 227]. CONE did manage to highlight cryptic associations between the liver and pure red-cell aplasia. Together, these results imply that CONE can help uncover subtle disease-tissue relationships. Thus, CONE contextualized embeddings can not only achieve good prediction performance, but the top-performing contexts typically show biological relevance.

5.5.3 Context-specific embedding enhances therapeutic target prediction

Inducing biological context information has been recently shown to be beneficial for predicting therapeutic targets in complex diseases such as Rheumatoid arthritis (RA) and Inflammatory Bowel Disease (IBD) [140]. Therapeutic target prediction for a particular disease is a binary classification problem that aims to predict whether a particular human gene can be used as a point of intervention

Table 5.3: **Top performing contexts for selected diseases in the DisGeNET benchmark.** Performance reported as test APOP scores averaged across five random splits. The top contexts are sorted descendingly from left two right. For example, the Heart context achieved the highest score for Nemaline myopathy.

Task	*node2vec*	CONE (naive)	CONE (best)	Top contexts
Familial hypertrophic cardiomyopathy	3.1481	2.6405	2.9609	Pancreas, Stomach, Muscle
Nemaline myopathy	4.9395	4.7192	5.1911	Heart, Stomach, Minor Salivary Gland
Subvalvular aortic stenosis	4.1582	3.6587	3.9863	Colon, Artery, Minor Salivary Gland
Hypochromic microcytic anemia	2.5056	1.3827	2.8930	Adipose, Skin, Small Intestine
Familial bicuspid aortic valve	1.2164	3.4816	4.1857	Pancreas, Stomach, Heart
Pure red-cell aplasia	6.0797	5.9410	6.1684	Stomach, Pancreas, Liver

for treating that disease. Compared to CONE, PINNACLE [140] takes an alternative approach by constructing a heterogeneous network that introduces different biological contexts as a type of node in the heterogeneous biomedical graph. Furthermore, the PINNACLE model learns individual sets of parameters for each biological context, contrasting with our unified model that shares the same set of parameters across all biological contexts. We hypothesize that our approach of tying weights induces more regularity from the underlying graph and, as a result, produces better-contextualized embeddings for predicting therapeutic targets.

To test this, we first use the cell-type specific genes processed by PINNACLE to generate cell-type specific CONE embeddings. We then follow the original evaluation and measure different contextualized embeddings' performance using APR@5. We note that the PINNACLE context-specific embeddings only contain embeddings of genes within the context. Conversely, CONE context-specific embeddings are genome-wide, meaning that embeddings in any context contain the same number of genes, spanning the whole network. Thus, to unify the setting between PINNACLE and CONE, we subset each context-specific CONE embedding to the corresponding context genes.

As shown in Figure 5.3, our CONE embeddings achieve significantly better performance than the PINNACLE embeddings when used in an unsupervised embedding setting, in which the training of the embedding does not involve node label information for the downstream prediction tasks. Furthermore, the highlighted immune cell contexts show that CONE better prioritizes the relevant cell context of IBD and RA, both of which are autoimmune diseases resulting from the malfunction of immune cells. Upon top-performing cell contexts in RA, CONE achieves better performance than

Table 5.4: **Top four performing contexts for predicting RA and IBD therapeutic targets.** The first block of rows show the top four cell types with highest APOP socres of predicting RA and IBD using PINNACLE embeddings. Similarly, the second block of rows show those for the CONE embeddings.

	Rheumatoid Arthritis (RA)			Inflammatory Bowel Disease (IBD)		
	Context	APOP	APR@5	Context	APOP	APR@5
PINNACLE	Large intestine goblet cell	1.734	0.333	Pulmonary ionocyte	3.145	0.333
	Intrahepatic cholangiocyte	1.634	0.200	Pancreatic acinar cell	3.051	0.250
	Retinal ganglion cell	1.537	0.200	Lymphatic endothelial cell	2.709	0.333
	Lung ciliated cell	1.525	0.040	Muscle cell	2.672	0.000
CONE	Retinal ganglion cell	3.186	0.760	Duct epithelial cell	5.977	1.000
	Pancreatic acinar cell	3.158	1.000	Luminal cell of prostate epithelium	5.954	1.000
	Pancreatic alpha cell	2.855	0.383	Tracheal goblet cell	5.883	1.000
	Club cell of prostate epithelium	2.710	0.383	Vascular associated smooth muscle cell	4.691	0.500

PINNACLE. Specifically, CONE reveals contexts that are biologically related to RA. For example, pancreatic acinar cells (rank of APOP=2, rank of APR@5=1) secrete digestive enzymes that are involved in the digestion process within the small intestine. The early activation of these digestive enzymes before they reach the duodenum can trigger the onset of acute pancreatitis [139]. On the other hand, acute pancreatitis is highly associated with RA. Clinical studies have shown that RA patients were 2.51 times more likely to develop acute pancreatitis [9]. CONE is also able to reveal hidden associations based on cell-type-specific networks. Similarly, CONE performs better than PINNACLE in identifying top relevant cell contexts related to IBD. CONE picked up duct epithelial cells (rank of APOP=1, rank of APR@5=1) as the top cell context. These cells are integral to the intestinal barrier, serving as the first line of defense against invading microorganisms [219]. However, in IBD patients, the proper function of the intestinal barrier is frequently compromised to varying degrees [12]. Overall, these examples demonstrate the superior power of CONE in predicting therapeutic targets using biologically relevant cell contexts.

5.5.4 CONE leverages diverse biological contexts beyond tissue and cell type

Since CONE takes contextual information in the form of node sets, it can be extended to biological contexts outside of traditionally used tissue and cell type contexts [297, 140]. Here, we explore the extendability of our CONE approach to different biological contexts in two other ways. First, we reevaluate the RA and IBD prediction performance using CONE trained on different disease contexts defined by differentially expressed genes obtained from CREEDS [271]. We find

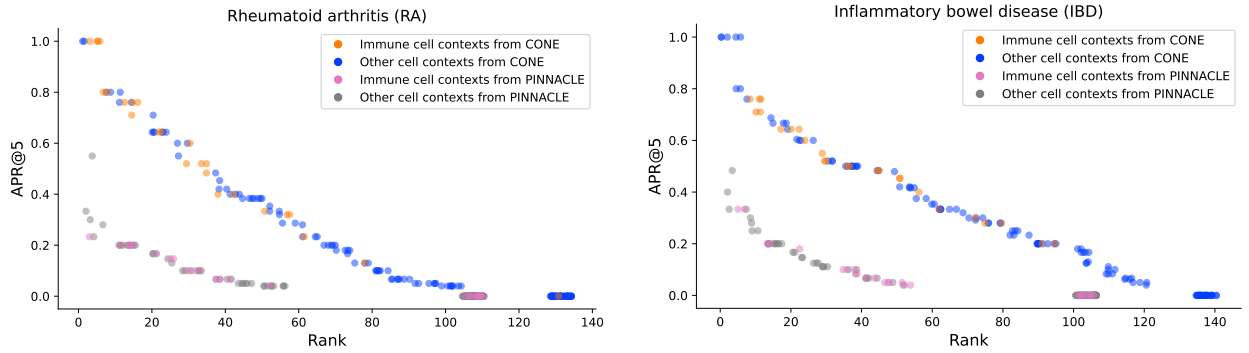


Figure 5.3: **Therapeutic area predictions for RA and IBD**. Each point represent the APR@5 score achieved when using a particular cell-type-specific embedding to predict the RA or IBD therapeutic targets. Immune cell contexts are highlighted in orange and pink for CONE and PINNACLE.

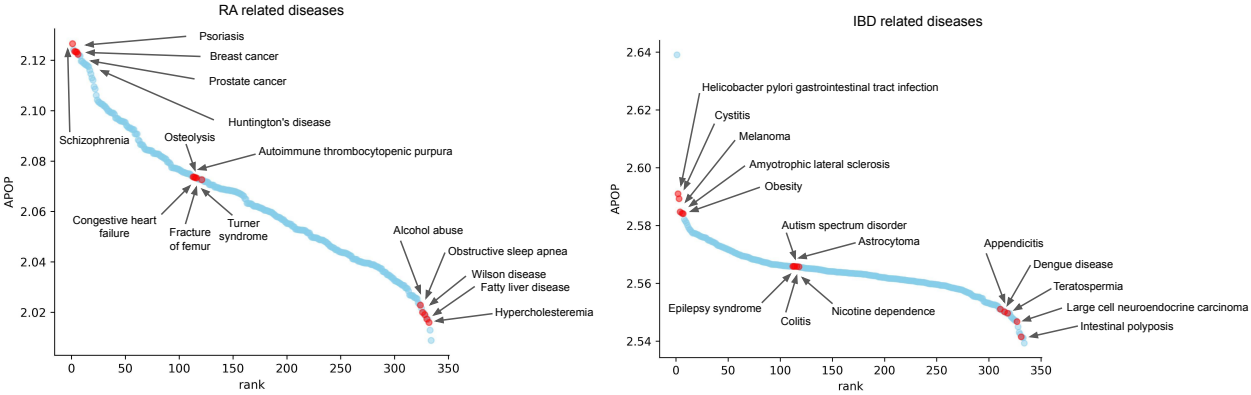


Figure 5.4: **Sorted disease contexts for predict the therapeutic area predictions for RA and IBD**.

that the top-ranked contexts for both RA and IBD are indeed highly relevant disorders for both diseases (Figure 5.4). For example, psoriasis is one of the top disease contexts related to RA (rank of APOP=3). A clear connection between these two conditions is psoriatic arthritis, a form of arthritis with a skin rash, which is a common symptom in psoriasis [284]. This indicates that there are similar genetic programs shared by both diseases [194], which is revealed by CONE using disease context networks. Furthermore, CONE also reveals several connections between RA and some seemingly unrelated diseases, such as heart failure (rank of APOP=115). Notably, a recent study has confirmed that RA patients have a two-fold higher risk of heart failure mortality than those without RA [193].

Similarly, CONE unveils meaningful relationships between IBD and other disease contexts. Cystitis (rank of APOP=3) is one of the top disease contexts identified by CONE. Clinical studies

have shown that cystitis, an inflammation of the bladder, leads to a significant increase in the risk of IBD [91, 48, 4]. For example, individuals with interstitial cystitis, a condition involving an inflamed or irritated bladder wall, are 100 times more likely to have IBD [4]. CONE also finds subtle relationships between IBD and neurological complications exemplified by epilepsy syndrome (rank of APOP=114) and autism spectrum disorder (rank of APOP=112) in the top list [42]. Neurological complications affect 0.25% to 47.50% IBD patients [69]. These IBD-related neurological complications are associated with neuroinflammation or increased risk of blood clots in brain veins [177]. Some diseases may have a protective effect on other diseases. CONE identified such protective relationships between *Helicobacter pylori* gastrointestinal tract infection (rank of APOP=2) and IBD, since *H. pylori* infection helps protect against IBD by inducing systematic immune tolerance and suppressing inflammatory responses [290]. Overall, these examples further confirm that CONE can leverage disease context to reveal both apparent and cryptic associations between complex diseases.

Finally, we reperform the DisGeNET benchmark using a diverse construction of context-specific gene sets, spanning from tissues, cell types, to diseases, and retrieved from various databases, including CellMarker2.0 [96], Human Protein Atlas [250], and the TISSUES database [191]. We observe that CONE performs similarly under all collections of contexts tested (Figure 5.5). Together with the fact that CONE performs competitively against baselines and captures biologically meaningful contextual information, we believe that CONE is a versatile and effective approach to scalably generate biological network embeddings conditioned on specific biological contexts.

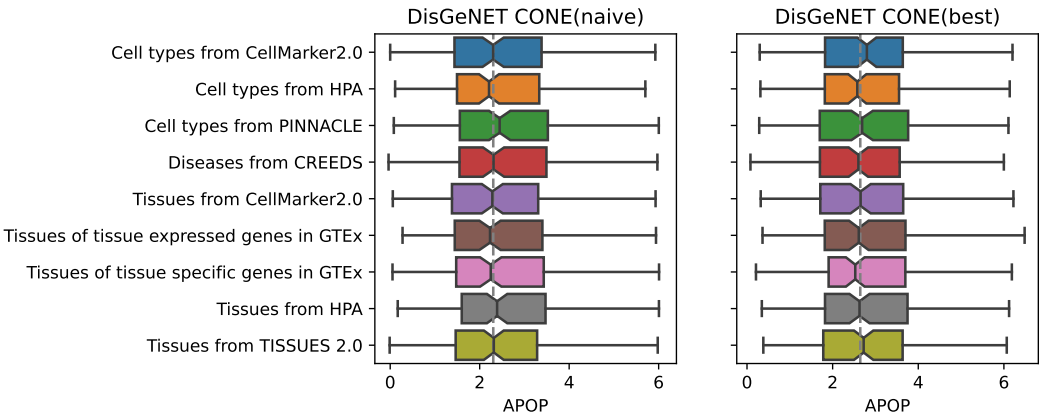


Figure 5.5: **Performance of CONE on the DisGeNET benchmark across contexts collections.**

5.5.5 CONE enables knowledge transfer to unseen contexts

The MLP context encoder used by CONE provides the possibility to generate embeddings for contexts that are not observed during training. This can be done by feeding the similarity profile of the new query context against all training contexts to the MLP encoder. To demonstrate the effectiveness of transferring to unseen context, we retrain the CONE GTEx tissue-specific embeddings but leave out the Heart context during training. We observe that holding out Heart context does not affect the disease gene classification performance significantly, with > 0.8 coefficient of correlation against the original performance. Furthermore, we compile two lists of diseases for which the Heart context achieved the top five performances for the original and held-out Heart versions of CONE. We found a significant overlap between the two lists (Hypergeometric p-val < 0.05), with seven common diseases (Table 5.5). One notable example is the familial bicuspid aortic valve, a known common congenital heart defect [29]. These results highlight the effective transferability of CONE to unseen contexts, with similar embedding quality as if the contexts were seen during training.

5.5.6 Additional experiments

5.5.6.1 Scalability

We empirically demonstrate the scalability of CONE against three other related methods, including GAT, BIONIC, and PINNACLE. All these methods use the GAT module as the main encoding component. CONE and GAT both employ a single GAT module, while BIONIC and

Table 5.5: **List of diseases for which the Heart context appears to be the top five performing contexts in terms of test APOP scores.** Last two columns indicate whether the disease shows up as top five context for the original CONE or the one trained without Heart context.

Disease ID	Disease Name	Original	Holdout Heart
MONDO:0005580	esophageal squamous cell carcinoma	✓	✓
MONDO:0001286	exotropia	✓	✓
MONDO:0007194	familial bicuspid aortic valve	✓	✓
MONDO:0018100	familial primary hypomagnesemia	✓	✓
MONDO:0017410	porencephaly	✓	✓
MONDO:0015194	sideroblastic anemia	✓	✓
MONDO:0006987	subvalvular aortic stenosis	✓	✓
MONDO:0007885	Legg-Calve-Perthes disease		✓
MONDO:0009532	Miller-Dieker lissencephaly syndrome		✓
MONDO:0015977	agammaglobulinemia		✓
MONDO:0016466	asbestosis		✓
MONDO:0002102	cheilitis		✓
MONDO:0017885	chromophobe renal cell carcinoma		✓
MONDO:0001342	dysgammaglobulinemia		✓
MONDO:0017610	epidermolysis bullosa simplex		✓
MONDO:0003435	microcystic adenoma		✓
MONDO:0018943	myofibrillar myopathy		✓
MONDO:0007243	Burkitt lymphoma	✓	
MONDO:0010269	Coats disease	✓	
MONDO:0000816	abdominal obesity-metabolic syndrome	✓	
MONDO:0007150	arcus senilis	✓	
MONDO:0015362	autosomal dominant distal hereditary motor neuropathy	✓	
MONDO:0000640	central nervous system primitive neuroectodermal neoplasm	✓	
MONDO:0004614	chronic monocytic leukemia	✓	
MONDO:0019942	distal arthrogryposis	✓	
MONDO:0002181	exostosis	✓	
MONDO:0003252	granular cell cancer	✓	
MONDO:0018778	intermediate Charcot-Marie-Tooth disease	✓	
MONDO:0020367	juvenile open angle glaucoma	✓	
MONDO:0011380	leukoencephalopathy with vanishing white matter	✓	
MONDO:0021637	low grade glioma	✓	
MONDO:0002478	mixed germ cell-sex cord-stromal tumor	✓	
MONDO:0004600	monocytic leukemia	✓	
MONDO:0018958	nemaline myopathy	✓	
MONDO:0019181	non-syndromic X-linked intellectual disability	✓	
MONDO:0019019	osteogenesis imperfecta	✓	
MONDO:0006335	ovarian endometrioid adenocarcinoma	✓	
MONDO:0005898	paronychia	✓	
MONDO:0005391	restless legs syndrome	✓	
MONDO:0006966	secondary Parkinson disease	✓	
MONDO:0008428	septooptic dysplasia	✓	
MONDO:0024268	superficial mycosis	✓	

PINNACLE use individual GAT modules for different contexts.

We consider two types of scaling experiments, *number of contexts* and *context node percentages*. For *number of contexts*, we fix the number of context-specific nodes to be about 5% of the total number of nodes and vary the number of contexts from 10 to 1000. Meanwhile, for *context node percentages*, we fix the number of contexts to be 100, and vary the context node percentages from 1% to 50%. For both sets of experiments, we use a synthetic network with $10,000$ nodes generated

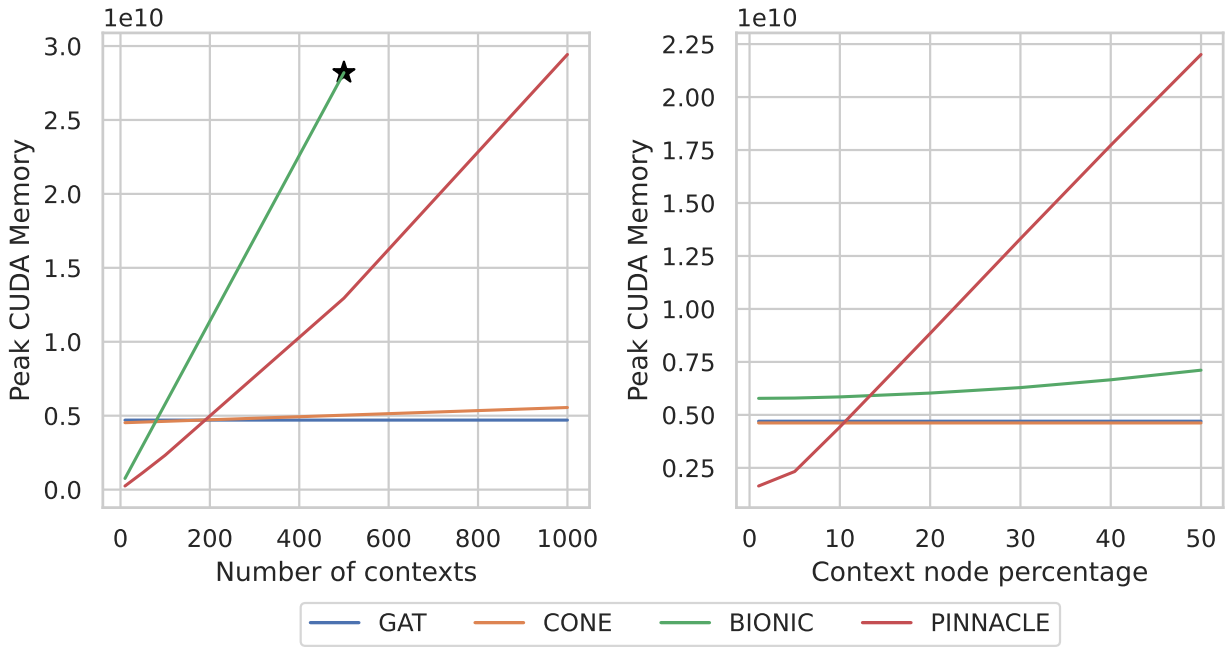


Figure 5.6: **Models scalability across different contextualization settings.** Star indicates the point beyond which the model will run out of memory.

using the Barabási–Albert model [18]. The network is generated so that the density is approximately 0.01. We report the peak CUDA memory usage (in bytes) of the forward pass for the model using the `torch.cuda.max_memory_allocated()` function. All experiments are conducted on compute nodes with 5 CPUs, 45GB memory, and a Tesla v100 GPU (32GB). We uniformly set the following hyperparameters across models: 128 dimensions and one layer. Furthermore, BIONIC and PINNACLE require subgraph batched training. We set the batch size to 2048 for both models. On the other hand, CONE and GAT employ full batched computation.

The empirical scalability results are shown in Figure 5.6. We first highlight that CONE shows great scalability, with very minimal overhead, as more contexts are introduced. Remarkably, CONE's memory consumption is comparable to the plain GAT model, which does not take context into account. This aligns well with our complexity analysis (Chapter 5.3.4). Conversely, BIONIC and PINNACLE react drastically to the number of contexts, with BIONIC running out of memory beyond 500 contexts. Furthermore, PINNACLE demonstrates a severe scalability issue with the context nodes percentage. In other words, PINNACLE memory consumption drastically increases

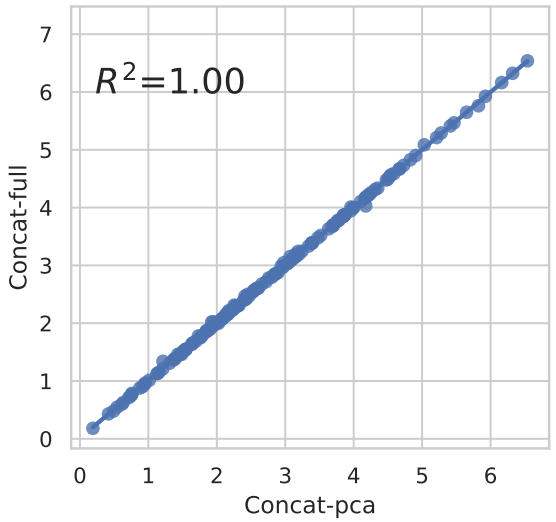


Figure 5.7: **CONE dimensionality reduction effect comparison.** Performance comparison between PCA-reduced CONE (x-axis) and not reduced CONE (y-axis) in terms of testing APOP on the DisGeNET disease gene classification benchmark.

as the context subgraphs increase. These results showcase the scalability advantage of CONE using a single shared GAT module to decode embeddings for all contexts.

5.5.6.2 Effects of PCA dimensionality reduction

The final CONE context-specific embeddings are obtained by first concatenating the context-naive with the context-specific embeddings and then applying PCA to project the dimension by half. Combining the context-naive and context-specific embeddings gives the final embedding a more comprehensive view of both the global and local (context-specific) semantics. Dimensionality reduction is applied so that the results for the final context-specific embeddings can be fairly compared to the context-naive embeddings. PCA is a common dimensionality reduction technique due to its simplicity and has been used in previous studies to combine multiple views of the embeddings, such as Walklets [203].

One question remaining is how does the performance change before and after applying PCA. Here, we compare the performance between the fully concatenated and PCA-reduced versions of CONE following the DisGeNET benchmarking setting. We observe little performance difference between the two versions of CONE (Figure 5.7). Thus, we set the final CONE to use PCA, as it provides a fairer setting in terms of the number of dimensions while having the same performance.

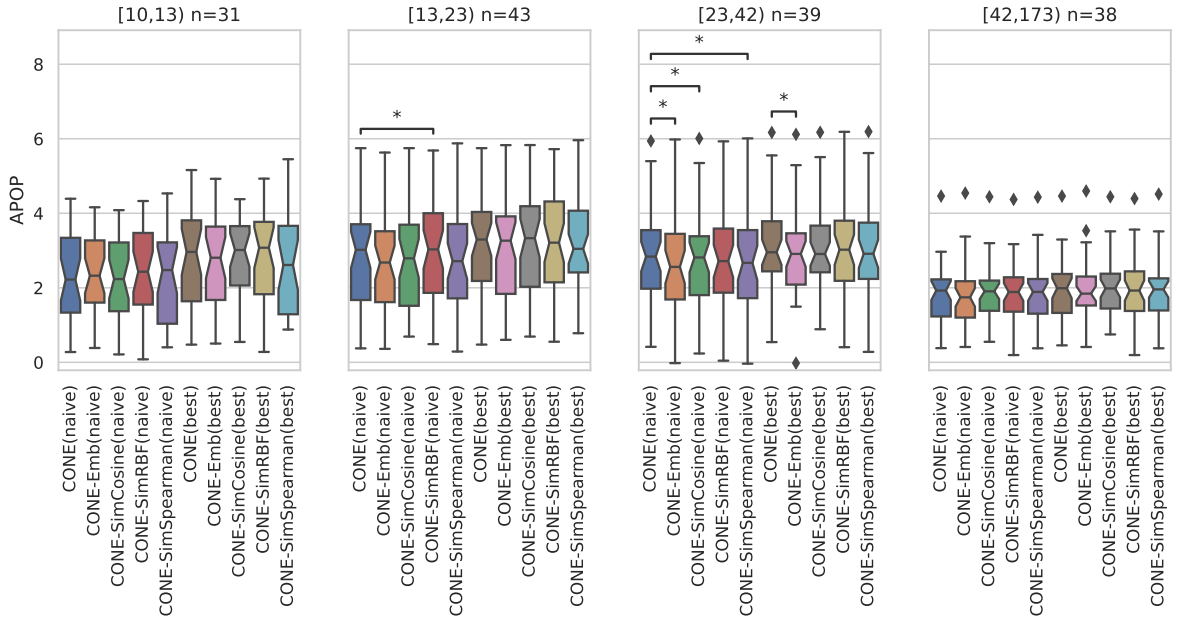


Figure 5.8: **Performance for different context similarity and context encoding strategies.** Each point in the box plot corresponds to the prediction test performance of a disease gene classification task from the DisGeNET benchmark, averaged across five random splits. Different panels show groups of diseases with different number of positive genes. * indicate that the performances between the two methods are significantly different (Wilcoxon p-value < 0.05).

5.5.6.3 Ablation studies

In the following, we investigate the effectiveness of our main design choices of CONE, including the context similarity measure and the MLP context encoder. We follow the same experimental settings as in our primary DisGeNET gene classification benchmark (Chapter 6.3).

Table 5.6: **Ablation study of context similarity and context encoding strategies using the DisGeNET benchmark.** Results are reported as APOP scores averaged across tasks within a group based on the number of positive examples.

	Similarity	Encoder	[10, 13)	[13, 23)	[23, 42)	[42, 173)
Default CONE setting	Jaccard	MLP	**2.813**	3.111	**3.116**	1.971
Context similarity ablations	Cosine	MLP	2.801	3.186	3.048	**2.003**
	RBF	MLP	2.805	3.189	3.038	1.957
	Spearman	MLP	2.705	**3.194**	3.050	1.966
Context encoder ablation	–	Embedding	2.697	3.083	2.911	1.932

Context similarity measures Besides the default Jaccard similarity measure, we consider three other similarity measures, including the cosine similarity, radial basis function, and Spearman

correlation coefficient. Table 5.6 shows that the choice of similarity has marginal effects on the performance, with the default Jaccard similarity consistently achieving better or equivalent performance against other choices of similarities. Figure 5.8 further indicates that there are no significant performance differences across the choice of similarity measures according to the paired Wilcoxon test.

Context encoder A trivial way to encode context embedding is one-hot encoding, which is equivalent to directly learning the embedding for each context. We call this approach *Embedding*. We observe that *Embedding* achieves the lowest average performance across all groups of tasks (Table 5.6). In the case of disease task group $[23, 42)$, the *Embedding* performance is significantly worse than that of the default CONE (Figure 5.8, Wilcoxon p-value < 0.05).

5.6 Conclusion

We have proposed CONE, a flexible and scalable approach that can inject diverse biological contextual information into the gene interaction network embeddings. Our study underscores the efficacy of the CONE method in enhancing the prioritization of genes within the gene interaction network. CONE consistently demonstrated superior performance in various biomedical tasks compared to baseline methods. Crucially, the introduction of context-specific embeddings, especially when positive genes for a disease were limited, resulted in significant performance gains. Moreover, the contexts identified by CONE were found to be biologically relevant, suggesting that the method not only boosts prediction accuracy but also provides biologically meaningful insights. This ability to integrate diverse biological contexts, from tissues and cell types to diseases, positions CONE as a versatile tool that can be used to uncover both explicit and cryptic relationships within biomedical datasets.

Limitations and future directions Constructing context-specific subnetworks solely based on the subgraph induced by context-specific genes is a reasonable but overly simplistic assumption. In reality, context-specific gene interactions are complicated and encompass diverse mechanisms, ranging from interactions mediated by non-coding RNAs to the influence of epigenetic modifications and signaling pathways. Consequently, a promising advancement would involve the carefully

constructed context-specific gene interaction networks that account for these intricate nuances, such as HumanBase [80], HIPPIE [5], IID [127], and many others [146, 43, 260].

CHAPTER 6

OPEN BIOMEDICAL NETWORK BENCHMARK: A PYTHON TOOLKIT FOR BENCHMARKING DATASETS WITH BIOMEDICAL NETWORKS

Over the past decades, network biology has been a major driver of computational methods developed to better understand the functional roles of each gene in the human genome in their cellular context. Following the application of traditional semi-supervised and supervised machine learning (ML) techniques, the next wave of advances in network biology will come from leveraging graph neural networks (GNN). However, to test new GNN-based approaches, a systematic and comprehensive benchmarking resource that spans a diverse selection of biomedical networks and gene classification tasks is lacking. Here, we present the Open Biomedical Network Benchmark (OBNB), a collection of node-classification benchmarking datasets derived using networks from 15 sources and tasks that include predicting genes associated with a wide range of functions, traits, and diseases. The accompanying Python package, `obnb`, contains reusable modules that enable researchers to download source data from public databases or archived versions and set up ML-ready datasets that are compatible with popular GNN frameworks such as PyG and DGL. Our work lays the foundation for novel GNN applications in network biology. `obnb` will also help network biologists easily set-up custom benchmarking datasets for answering new questions of interest and collaboratively engage with graph ML practitioners to enhance our understanding of the human genome. OBNB is released under the MIT license and is freely available on GitHub: https://github.com/krishnanlab/obnb

6.1 Introduction

Life is orchestrated by remarkably complex interactions between biomolecules, such as genes and their products. Network biology [22, 19, 108, 261] has demonstrated remarkable success over the past two decades in systematically uncovering genes' functions and their relations to human traits and diseases. Accurately identifying genes associated with a particular disease, for example, is a vital step towards understanding the biological mechanisms underlying the condition, which in turn could lead to novel and effective diagnostic and treatment strategies [268, 136]. Early work in network

biology focused on network diffusion-type methods based on the *guilt-by-association* principle [188], which states that genes interacting with each other likely participate in the same biological functions or pathways. The performance of these methods has subsequently been improved by the application of supervised learning [181]. In this trajectory, the next wave of improvements in network biology is likely to emanate from the surge of powerful graph machine learning (ML) techniques such as graph embeddings [78, 39] and graph neural networks [279, 295]. These methods have shown promising results in many graph-structured tasks, such as social networks [99], and have started to attract researchers to apply them to biological tasks [16, 291, 183]. To this end, accelerating the development and application of graph ML methods in network biology is of great importance.

However, there is a critical need for standardized benchmarks that allow reliable and reproducible assessment of the novel graph ML methods [232, 99]. Recent efforts such as MoleculeNet [278] and Therapeutics Data Commons [102] for molecular and therapeutics property predictions, and Benchmarking GNN [64] and Open Graph Benchmark [99, 98] for more general graph benchmarks, have proven valuable in advancing the field of graph ML by providing carefully-constructed benchmarking datasets for applying specialized methods. Meanwhile, such comprehensive benchmarking datasets and systems are currently lacking for network biology. Furthermore, setting up ML-ready datasets for network biology is incredibly tedious. Some necessary steps include converting gene identifiers (IDs), setting up labeled data from annotated biomedical ontologies, filtering labeled data based on network gene coverage, and constructing realistic data splits mimicking real-world biologically meaningful scenarios. As a result, despite the remarkable amount of publicly available data for biomedical networks [100, 111] and annotations [235], the only widely available ML-ready datasets for network biology are `PPI` dataset from OhmNet [297] and `PPA` from the Open Graph Benchmark [99].

Contributions In this work, we address this critical need. Our main contributions are as follows:

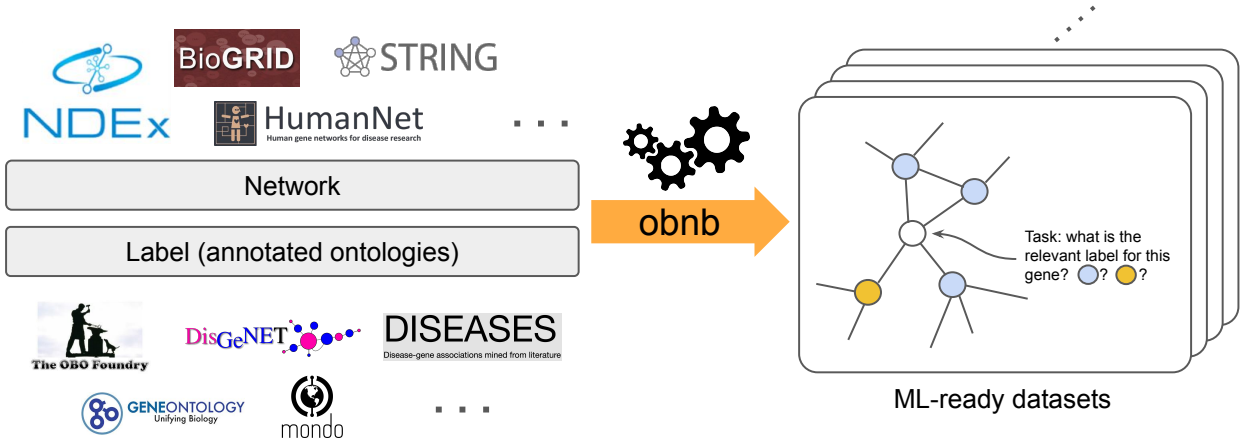
1. We present a Python package `obnb` that provides reusable modules for data downloading, processing, and split generation to set up node classification benchmarking datasets using publicly available biomedical networks and gene annotation data. The first release version

contains interfaces to networks from 15 sources and annotations from three sources.

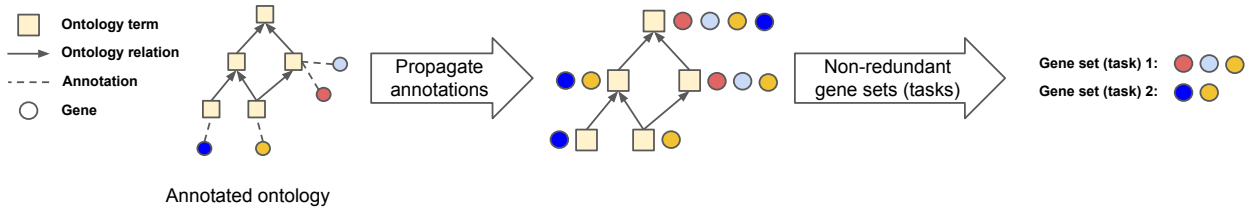
2. We present a comprehensive benchmarking study on the OBNB node classification datasets with a wide range of graph ML approaches and tricks to set up the baseline for future comparisons.

3. We analyze the benchmarking results and point out several exciting directions and the potential need for a special class of graph neural network to tackle the OBNB tasks.

Related work Several existing Python packages share similar goals with `obnb`, primarily focusing on establishing biomedical network datasets and facilitating their analyses. In these networks, nodes typically represent genes or their products, such as proteins, while edges represent the functional relationships between them [100], such as physical interactions. `PyGNA` [68] offers a suite of tools for analyzing and visualizing single or multiple gene sets using biological networks. `PyGenePlexus` [166] and `drug2ways` [218] specialize in network-based predictions of genes and drugs. The `OGB` [99] platform houses a variety of graph benchmarking datasets, which includes a PPI dataset akin to the STRING-GOBP dataset in OBNB (Table C.6). Nonetheless, all the aforementioned packages have a limited number of, if any, biomedical network and label data. `obnb`, on the other hand, provides an extensive number of biomedical networks and diverse gene set collections to facilitate the systematic evaluation of graph machine learning methods on diverse datasets. In a related domain, `PyKEEN` [7] provides a vast array of biomedical knowledge graph (KG) datasets and KG embedding methods. There, the main task of interest is link prediction, through which the missing knowledge link can be completed. Other notable works for constructing large-scale biomedical KG and setting up link-prediction benchmarks from them include `BioCypher` [156] and `OpenBioLink` [30]. While the tasks associated with KG [267] are orthogonal to the node classification settings, it is possible to reformulate gene classification problems as KG completion and vice versa [16, 291]. Nevertheless, the advantages and drawbacks of these two approaches are yet to be comprehensively evaluated.



(a) obnb downloads network and label data from public data sources, then process and combine them into ML-ready benchmarking graph datasets.



(b) Generating label data from annotated ontology through annotation propagation and non-redundant gene set extraction.

Figure 6.1: **Overview of obnb data processing and benchmarking dataset generation.**

6.2 OBNB system description

Making the process of setting up ML-ready network biology benchmarking datasets from publicly available data as effortlessly as possible is the core mission of `obnb`. We implement and package a suite of graph (`obnb.graph`) and label (`obnb.label`) processing functionalities and couple them with the high-level data object `obnb.data` to provide a simple interface for users to download and process biological networks and label information. For example, calling `obnb.data.BioGRID("datasets")` and `obnb.data.DisGeNET("datasets")` will download, process, and save the BioGRID network and DisGeNET label data under the `datasets/` directory, which can then be loaded directly next time the functions are called. Users can compose a dataset object using the network and the label objects, along with the split (Chapter 6.2.3), which can then be used by a model trainer to train and evaluate a particular graph learning method. Alternatively, the composed dataset object can be transformed into data objects for standard GNN

framework, including PyTorch Geometric (PyG) [70] and Deep Graph Library (DGL) [265].

In the following subsections, we provide details about the processing steps for the biomedical networks (Chapter 6.2.1), creation of node labels from annotated ontologies (Chapter 6.2.2), and preparation of data splits (Chapter 6.2.3). Finally, as we are dedicated to creating a valuable resource for the computational biology and graph machine learning communities, we closely follow several open-source package standards, as elaborated in Chapter 6.2.6.

6.2.1 Network

Downloading Currently, tens of genome-scale human gene interaction network databases are publicly available, each constructed and calibrated with different strategies and sources of interaction evidence [100]. Unlike in many other domains, such as chemoinformatics, where there are only a few ways to construct the graph (e.g., molecules), gene interaction networks can be defined and created in a wide range of manners, all of which capture different aspects of the functional relationships between genes. Some broad gene interaction mining strategies include experimentally captured physical protein interactions [240], gene co-expression [113], genetic interactions [60], and text-mined gene interactions [246]. We leverage the Network Data Exchange (NDEx) [210] to download the biological network data when possible (Figure 6.1a). The obtained `CX` stream format is then converted into a `obnb.graph` object for further processing.

Gene ID conversion In gene interaction networks, each node represents a gene or its gene product, such as a protein. There have been several standards in mapping gene identifiers, and different gene interaction networks might not use the same gene ID. For example, the STRING database [246] uses Ensembl protein ID [104], PCNet [100] uses HGNC [79], and BioGRID [240] uses Entrez gene ID [104]. Here, we use the NCBI Entrez gene ID for its advantages, such as supporting more species other than Human and being less ambiguous [94]. To convert other types of gene IDs into the Entrez gene, we use the MyGene query service [277], which provides the most up-to-date gene ID mapping across tens of gene ID types. Following, we remove any gene where more than one gene ID is mapped to the same Entrez gene, which indicates ambiguity in annotating the gene identifier currently. The gene interaction network after gene ID conversion will contain equal or less genes,

all of which are one-to-one mapped from the original gene IDs to the corresponding Entrez genes.

Connected component In practice, a small fraction (typically within 1%) of genes may be disconnected from the largest connected components. The disconnectedness of the network is typically due to missing information about gene interactions from the measurement; the more information a network uses to define the interactions, the denser and less likely there will be disconnected genes. Thus, we extract the largest connected component of the gene interaction network by default as it is more natural to have a single component for the transductive node classification tasks we consider. However, a user can also choose to take the full network without extracting the largest connected component by specifying the `largest_comp` option to `False` when initializing the data object.

6.2.2 Label

Many biological annotations are organized into abstract concept hierarchies, known as ontologies [23]. Each ontology term (a node in the ontology graph) is associated with a set of genes provided by current curated knowledge. By the hierarchical nature of the ontologies, if a gene is associated with an ontology term, then it must also be associated with any more general ontology terms. Thus, we first propagate gene annotations over the ontology by assigning a gene to all parent terms of its original annotated terms. This will result in a highly redundant gene set collection, where each ontology term necessarily contains a subset of its parent term's set of genes. We reduce the gene set collection's redundancy by we picking a representative and non-redundant subset of the gene sets. The final processed data results in the form of $\mathbf{Y} \in \{0, 1\}^{N \times p}$ with N number of genes and p number of labels. Below, we provide detailed description of (1) the gene annotation propagation, and (2) gene set filtering steps.

6.2.2.1 Annotated ontology

An ontology is a directed acyclic graph $H = (V_H, E_H)$, where $v \in V_H$ is an ontology term. The directed edge $(u, v) \in E_H$ indicates v is a parent of u, that is, v is more abstract or general concept containing u. Consider B as the set of all genes we are interested in, and $\mathcal{P}(B)$ be the power set of B. Given a gene annotation data, represented as a set of pairs $\mathcal{A} = \{(b, v)_i\}$, where

$b \in B$ and $v \in V_H$ are gene and ontology term, we represent the *raw annotation* $J_0 : V_H \to \mathcal{P}(B)$ as a map from an ontology term to a set of genes, so that $J_0(v) = \{b : (b, v) \in \mathcal{A}\}$. We then propagate the *raw annotation* $J_0(\cdot)$ over the ontology H into a *propagated annotation* $J(\cdot)$, where $J(v) = \cup_{v':\exists \text{path from} v' \text{ to } v} J_0(v')$, as depicted in Figure 6.1b.

Downloading All the ontology data is downloaded from the OBO Foundry [235] (Figure 6.1a), which actively maintains tens of large-scale biological ontologies. In the first release, we focus on three different annotations, Gene Ontology (GO) [14], DisGeNET [206], and DISEASES [81]. We naturally split GO into three different sub-collections, namely biological process (GOBP), cellular component (GOCC), and molecular function (GOMF), resulting in a total number of five label collections.

6.2.2.2 Filtering

The propagated annotations contain highly redundant gene sets, which may bias the benchmarking evaluations toward genes sets that commonly appear. To address this, we adapt the non-redundant gene set selection scheme used in [153]. In brief, we construct a graph of gene sets based on the redundancy between two gene sets, and recursively extract the most representative gene set in each connected component (Figure 6.1b). In addition, we provide several other filtering methods, such as filtering gene sets based on their sizes, number of occurrences, and their existence in the gene interaction network, to help further clean up the gene set collection to be used for final evaluation. Advanced users can alter these filtering functionalities to flexibly set up a custom gene set collection to better suit their biological interests besides using the default filtering steps provided by `obnb`.

6.2.3 Data splitting

A rigorous data splitting should closely mimic real-world scenarios to provide accurate and unbiased estimations of the models' performance. One stringent solution is temporal holdout, where the training input and label data are obtained before a specific time point, and the testing data is only available afterwards [58, 153]. In practice, this temporal setting is often too restrictive and leads to insufficient tasks for evaluation [153]. Thus, by default, we consider a closely related but less strict strategy called *study-bias holdout*.

6/2/2 study-bias holdout We use top 60% of the most studied genes with at least one associated label for training. The extend to which a gene is studied is based on its number of associated PubMed publications retrieved from NCBI[1]. The 20% least studied genes are used for testing, and the rest of the 20% genes are used for validation. Notice that by splitting up genes this way, some of the tasks may get very few positive examples in one of the splits. Hence, we remove any task whose minimum number of positive examples across splits falls below a threshold value (set to 10 by default). For completeness, we also provide functionalities in `obnb` to generate random splits.

6.2.4 Dataset construction

The network (Chapter 6.2.1) and label (Chapter 6.2.2) modules provide flexible solutions for processing and setting up datasets. Meanwhile, we provide a default dataset constructor that utilizes the above modules with reasonable settings to set up a dataset given particular selections of network and label. The default dataset construction workflow is as follows.

1. Select the network and label (task collection) of interest.
2. Remove genes in the task collection that are not present in the gene interaction network.
3. Remove tasks whose number of positive examples falls below 50 after the gene filtering above.
4. Setup 6/2/2 study-bias holdout (Chapter 6.2.3).

6.2.5 Example code

The `obnb` library provides high-level APIs for users to set up benchmarking datasets from diverse selections of gene interaction networks and gene annotations to study the performance of various graph learning methods. Below, we provide a simple code snippet demonstrating how one can easily set up a full dataset in a single function call. Further, we demonstrate how the constructed dataset can be directly used to evaluate the performance of a simple GNN model. More comprehensive example usage can also be found in the `obnb` README file: https://github.com/krishnanlab/obnb/blob/main/README.md.

[1] https://www.ncbi.nlm.nih.gov/

```
from obnb.dataset import OpenBiomedNetBench

# Download the current processed archive (set version to "latest" to
# download data from source directly and process them from scratch)
dataset = OpenBiomedNetBench(root="datasets", version="current", graph_name="BioGRID",
                             label_name="DisGeNET", auto_generate_feature="OneHotLogDeg")

# User built-in GNN trainer to train and evaluat the performance of a simple GCN
gcn_model = GCN(in_channels=1, hidden_channels=64,
                num_layers=5, out_channels=dataset.num_tasks)
gcn_trainer = SimpleGNNTrainer(device="cuda", metric_best="apop")
gcn_results = gcn_trainer.train(gcn_model, dataset)

# Alternatively, convert the dataset into your favorite GNN framework, i.e., PyG or DGL
# Or use OpenBiomedNetBenchPyG or OpenBiomedNetBenchDGL to directly instantiate the dataset
pyg_data = dataset.to_pyg_data()
dgl_data = dataset.to_dgl_data()
```

6.2.6 Community standards and maintenance plans

We follow several community standards to ensure sustainable and maintainable community-wide contributions, which is the key to the continual development of a code base. First, we release the code on GitHub https://github.com/krishnanlab/obnb under the permissive MIT license. Second, we use Sphinx to build the documentation of the code base and host it on ReadTheDocs.org. Several quick-start examples using the package are also provided on the GitHub README page. Third, code quality is ensured via various testing and code-linting automation using tox, pytest, pre-commit hooks, and GitHub actions. Fourth, we provide contribution guidelines and a code of conduct on the GitHub page. Finally, as a part of our commitment to the community, we also put in place a maintenance plan to address GitHub issues, merge pull requests, and release updates periodically to ensure the benchmarks remain adaptive to the evolving needs of the community.

6.3 Benchmarking experiment setup

To provide solid baselines for future references, we benchmark a wide range of graph ML methods on our comprehensive and diverse OBNB datasets, covering 15 gene interaction networks and four gene classification tasks (Appendix C.1). The classification performance is reported as the average

test precision over prior (APOP) score across five seeds (see Chapter 6.3.6 for the mathematical definition and the motivation of choosing APOP as our main metric).

6.3.1 Datasets

We present the primary results from the combination of two networks (BioGRID and HumanNet) and two gene classification tasks (GOBP and DisGeNET). The two networks are chosen for their distinct characteristics: BioGRID is an unweighted graph whose edges indicate protein interaction evidence from high throughput experiments [240], whereas HumanNet is a weighted graph whose edges indicate much more diverse types of interactions such as gene coexpression and associations derived via literature text-mining [107]. Meanwhile, the two selected label collections cover two broad classes of gene classification problems, namely, gene function prediction (GOBP) and disease gene association prediction (DisGeNET). The statistics about networks and task collections for the primary datasets are shown in Table 6.1 (see Table C.5 and Table C.6 for all dataset statistics).

Table 6.1: **Main dataset statistics.** The network density is computed as the ratio of existing edges over all possible undirected edges: $2(\#edges)/(\#nodes \times (\#nodes - 1))$.

(a) Network statistics

	# nodes	# edges	Density
BioGRID	18,951	1,103,298	0.0030
HumanNet	17,211	847,104	0.0029

(b) Label set collection statistics

	# tasks	# positives
DisGeNET	298	198.6 ± 135.0
GOBP	105	88.4 ± 35.3

6.3.2 Baselines

We consider three general types of methods: (1) **label propagation** [293] that directly propagates label information over the graph, (2) **graph embedding** that first extracts the vectorial representations of each node in the graph, which are then used to fit a simple classifier, such as logistic regression, and (3) **GNN** that learns the mapping from each node to its label space end-to-end.

Graph embeddings We include three distinct featurizations in the main result: using the rows of the adjacency matrix as the node features (Adj) [153], using the *node2vec* embedding (N2V) [83], or using the Laplacian EigenMap embedding (LapEigMap) [25]. Extended results using SVD, LINE [247], and Walklets [203] are available in the Appendix (Table C.2). Each of these features is

coupled with an ℓ_2 regularized logistic regression model (LogReg) for downstream prediction.

GNN We include multiple variations of two GNN, GCN [124] and GAT [257], in the main results. Extended results for SAGE [89], GIN [282], and GatedGCN [31] can be found in the Appendix (Table C.2). One major challenge in applying GNN to OBNB datasets is the lack of canonical node features. This is unlike networks in other domains, such as citation networks, where node features are naturally defined using the paper content, such as bag of words [99]. To tackle this problem, we experiment with a diverse selection of node featurization strategies, including using the one hot encoded log degree of the node, using *node2vec*, and many others. We provide detailed descriptions for the 15 different feature construction strategies in Appendix 6.3.3.1, with extended results for GCN and GAT in Table C.1.

Bag of Tricks (BoT) In addition to optimizing the model architectures, many tricks in model training and feature augmentations have also shown to be key factors for practically good performance in existing benchmarks [99]. Here, we further investigate the usefulness of several popular tricks, including reusing training labels as node features (LabelReuse) [269] and performing post-correction to the predictions at test time via correct and smooth (C&S) [103].

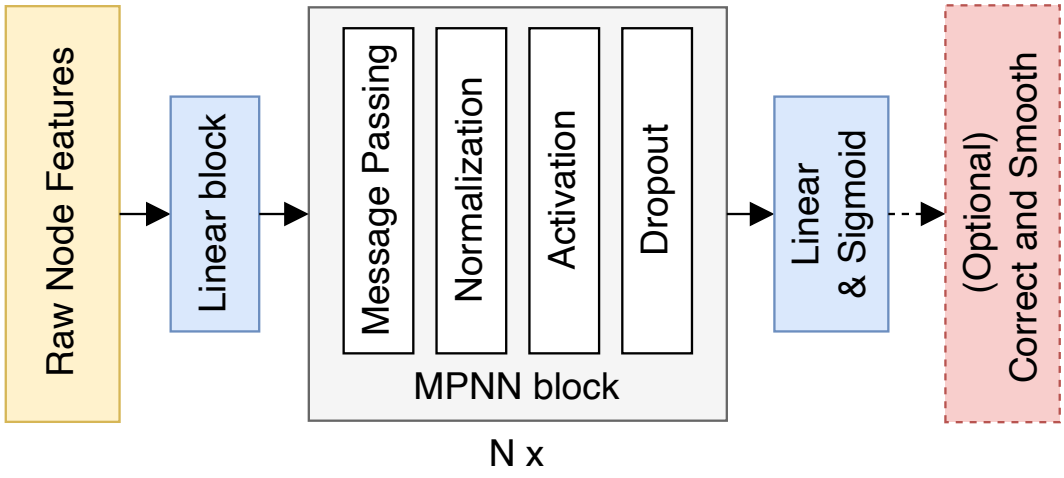


Figure 6.2: **GNN model architecture overview.**

6.3.3 GNN backbone design

The basic GNN architecture we used contains the following components:

- **Feature encoder** Since the genes do not come with canonical features, we need to derive initial node features for the GNN model. The default option we used is OneHotLogDeg with 32 bins (6.3.3.1). The raw features are first processed by a custom batch norm layer that is activated during testing in addition to training. Then, the processed features are projected into the hidden dimension ($d = 128$ by default) of the model via a linear layer, followed by batch normalization and ReLU activation.

- **Message passing layers** We use five message passing layers in the GNN model by default. Each message passing layer contains a convolution block (6.3.3.2), followed by normalization, non-linear activation, and dropouts. We apply residual connection by summing up the input and the output of the graph convolution block.

- **Prediction head and post-processing** Finally, we apply a linear layer to project the hidden dimension to the dimension matching the number of tasks and apply a sigmoid activation to turn the prediction into binary prediction probabilities. Optionally, we apply correct and smooth (C&S) [103] at test time to correct the predicted probabilities via a two-step label propagation (6.3.3.3).

6.3.3.1 Node features

In this section, we provide an overview of a diverse selection of node feature constructions we used in our benchmark. All node features are constructed using the whole network as input in a transductive setting, except for a few cases where the node features do not depend on the network structure, such as constant features. By default, any initial node feature will be 128-dimensional unless otherwise specified. We start by designing a collection of simple statistics that can be easily derived.

- **Constant** uses a one-dimensional trivial feature for every node in the graph.

- **RandomNormal** samples 32-dimensional features for each node independently via a standard normal distribution.

- **OneHotLogDeg** (short for **LogDeg**) first computes the log degree of each node in the graph and then uniformly bins the nodes into one of 32 bins based on their log degree. The one-hot encoded node degrees approach has recently been shown to be a great structure encoder, whose utilization can sometimes result in performance superior to using the original node features associated with the graph [52, 149]. Meanwhile, the design choice of using log-uniform grids stems from the scale-free nature of biological networks [6].

- **RandomWalkDiag** is the landing probability of a node back to itself after k hops, which is also commonly referred to as the random walk structural encodings (RWSE) [65]. It has been used widely in many graph transformer models due to its expressiveness in capturing graph structure [141]. Next, we consider several node feature options derived directly from the adjacency matrix.

- **Adj** uses the rows of the adjacency matrix as the node feature. It has been shown previously [153] that logistic regression using the adjacency matrix produced better prediction performance than the commonly used label propagation algorithm for diverse gene classification problems.

- **RandProjGaussian** and **RandProjSparse** are random projections of the adjacency matrix using two different but related approaches. We use the scikit-learn [196] implementations (`GaussianRandomProjection` and `SparseRandomProjection`) to compute these features.

- **SVD** uses the left singular vectors of the adjacency matrix as the node features.

- **LapEigMap [25]** uses the (ℓ_2 normalized) eigenvectors of the symmetric normalized graph Laplacian as the node features.

Node embeddings [183] are powerful approaches to extracting vectorial representations of each node in a graph and have shown promising results in many biomedical application [153, 291, 16]. Thus, we include a few popular and good-performing node embedding options in our benchmark.

- **LINE1** and **LINE2 [247]** extract first- and second-order proximity information from the graph to train the underlying embeddings. We use the GraPE [41] implementation to compute the LINE embeddings.

- **Node2vec [83]** extracts node representation using word2vec [174] on node sequences sampled from the graph via biased random walks. The biased random walks, in contrast to an earlier work, DeepWalk [202], which uses an unbiased search, allow the searching strategy to be more flexible, mimicking either breadth-first search or depth-first search in the random walk phase.

- **Walklets [203]** is similar to botch Node2vec and DeepWalk in that it runs word2vec using random walks sampled from the graph. However, wallets sample node pairs from the random walk with a specific number of hops, allowing a more explicit control of the multiscale relationships between nodes.

Finally, we experiment with options that let the model learn the node features freely.

- **Embedding** lets the model learn the node features freely.

- **AdjEmbBag** is similar to Embedding but with an additional aggregation step that sums up the raw embedding of each node from the central node's neighborhood.

6.3.3.2 Graph neural networks

In this section, summarize the five tested GNN models under the message-passing framework [74]. Let h_i^l be the *raw* representation of vertex i at layer j, and $\tilde{h}_i^l$ be the corresponding *processed* representation, e.g., after non-linear activation and normalization. The message-passing framework is written as

$$h_i^{l+1} = f^l\Big(\bigoplus_{j\in\mathcal{N}_i} \phi_n^l(\tilde{h}_j^l), \phi_s^l(\tilde{h}_i^l)\Big) \tag{6.1}$$

where $\bigoplus$ is the aggregation operator, f is the update function, ϕ_n and ϕ_s are message functions for neighbors and self. Different GNNs differ in the choices of $\bigoplus$, f, ϕ_n and ϕ_s they use.

GCN [124] uses a scaled linear transformation as the message function. The scaling factor is computed as the normalized edge weights with added self-loop. The aggregation is done by summing up the transformed message functions.

$$h_i^{l+1} = \sum_{j \in \mathcal{N}_i} \left(\frac{e_{i,j}}{\sqrt{\tilde{d}_i \tilde{d}_j}} \Theta^l \tilde{h}_j^l \right) + \frac{e_{i,i}}{\tilde{d}_i} \Theta^l \tilde{h}_i^l \tag{6.2}$$

where $\tilde{d}_i = d_i + 1 = \sum_{j \in \mathcal{N}_i} e_{i,j}$ is the degree of vertex i with self-loop added.

SAGE [89] uses two separate linear transformations as the message functions for neighbors and self. The aggregation is done by taking the sum[2] of the neighbors' messages and then adding it to the self-message. We additionally use an affine update function to transform the aggregated messages.

$$h_i^{l+1} = \Theta_u^l \left(\Theta_n^l \Big(\sum_{j \in \mathcal{N}_i} \tilde{h}_j^l \Big) + \Theta_s^l \tilde{h}_i^l \right) + b_u^l \tag{6.3}$$

GIN [282] uses a multi-layer perceptron (MLP) to update the aggregated messages, which is done by summing up neighbors' representations and self-representation from the last layer.

$$h_i^{l+1} = \mathrm{MLP}^l \Big(\sum_{j \in \mathcal{N}_i} \tilde{h}_j^l + (1 + \epsilon) \tilde{h}_i^l \Big) \tag{6.4}$$

GAT [257, 32] uses an attention mechanism to distribute the weights from which the linear transformed messages are aggregated.

$$h_i^{l+1} = \sum_{j \in \mathcal{N}_i} \left(\alpha_{i,j} \Theta_v^l \tilde{h}_j^l \right) + \alpha_{i,i} \Theta_v^l \tilde{h}_i^l \tag{6.5}$$

In particular, the original GAT paper [257] formulated the attention scores as follows

$$\alpha_{i,j} = \frac{\exp\Big(\mathrm{LeakyReLU}\big((a^l)^\top \Theta_a^l [\tilde{h}_i^l || \tilde{h}_j^l] \big) \Big)}{\sum_{k \in \mathcal{N}_i} \exp\Big(\mathrm{LeakyReLU}\big((a^l)^\top \Theta_a^l [\tilde{h}_i^l || \tilde{h}_k^l] \big) \Big)} \tag{6.6}$$

However, it was pointed out in a more recent work, GATv2 [32], that the above formulation was fundamentally limited in the types of attention the model can learn and proposed a correction that showed stronger performances follow.

$$\alpha_{i,j} = \frac{\exp\Big((a^l)^\top \mathrm{LeakyReLU}\big(\Theta_a^l [\tilde{h}_i^l || \tilde{h}_j^l] \big) \Big)}{\sum_{k \in \mathcal{N}_i} \exp\Big((a^l)^\top \mathrm{LeakyReLU}\big(\Theta_a^l [\tilde{h}_i^l || \tilde{h}_k^l] \big) \Big)} \tag{6.7}$$

[2]The original paper uses mean aggregation, but we found that sum aggregation works better in our benchmark.

We also observe a slight but significant improvement when using GATv2 attention as opposed to the original GAT. Thus, all reported GAT results are based on v2 attention.

GatedGCN [31] uses gating mechanisms to process neighbors' messages, with linear transformations as the message functions. The aggregation is done by summing up the gated messages from neighbors and adding them to the self-message.

$$h_i^{l+1} = \sum_{j \in \mathcal{N}_i} \left(\eta_{i,j} \odot \Theta_v^l \tilde{h}_j^l \right) + \Theta_s^l \tilde{h}_i^l \tag{6.8}$$

where $\odot$ is the elementwise multiplication operator, and the gating coefficients $\eta_{i,j}$ are computed as

$$\eta_{i,j} = \text{sigmoid}\left(\Theta_q^l \tilde{h^l}_i + \Theta_k^l \tilde{h^l}_j \right) \tag{6.9}$$

6.3.3.3 Network diffusion

Label propagation iteratively propagates the label information from the source nodes outwards through the network neighborhoods [126, 51, 180]. Let $\mathbf{Y} \in \{0, 1\}^{n \times d}$ be the (training) label matrix, and $\mathbf{M} \in \mathbb{R}^{n \times n}$ be the diffusion operator. The propagated information at step t, denoted $\mathbf{F}_t \in \mathbb{R}^{n \times d}$, is defined as

$$\mathbf{F}_t = \alpha \mathbf{M} \mathbf{F}_{t-1} + (1 - \alpha) \mathbf{F}_0 \tag{6.10}$$

where $\mathbf{F}_0 = \mathbf{F}$ is the features to be propagated, which is the label matrix $\mathbf{Y}$ in the case of label propagation. The propagated feature is computed by repeating the propagation above until convergence:

$$\text{PROPAGATE}(\mathbf{F}) = \lim_{t \to \infty} \mathbf{F}_t \tag{6.11}$$

In practice, equation 6.11 is approximated by applying the propagation until the changes are small enough.

The original label propagation paper [293] uses the symmetric normalized adjacency matrix $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ as the diffusion operator, where $\mathbf{A}$ and $\mathbf{D}$ are the adjacency matrix and the corresponding diagonal degree matrix. Here, we use the column stochastic matrix $\mathbf{D}^{-1}\mathbf{A}$ as the diffusion operator to resemble the random walk with restart (RWR) implementation that is more commonly used in the network biology literature [51]. We use a propagation parameter α of 0.1 (equivalent to a restart parameter of 0.9) for label propagation.

C&S implements the idea of using label propagation[3] to (1) correct for the error made by the model and (2) smooth out the corrected predictions. Specifically, let $\mathbf{E} = \mathbf{Y}_{\text{train}} - \mathbf{Z}$ be the prediction error matrix, where $\mathbf{Z}$ is the predicted probabilities resulting from the model. C&S can be summarized as follows.

1. Propagate the error matrix: $\tilde{\mathbf{E}} = \text{PROPAGATE}(\mathbf{E})$.
2. Correct the original prediction with a fixed scale s: $\tilde{\mathbf{Z}} = \mathbf{Z} + s\tilde{\mathbf{E}}$.
3. Smooth out the corrected predictions: $\mathbf{Z}_{\text{C\&S}} = \text{PROPAGATE}(\tilde{\mathbf{Z}})$

We use $\alpha = 1.0$ for the correction step, $\alpha = 0.8$ for the smoothing step, and a scaling factor of $s = 1.5$.

6.3.4 Training and hyperparameter details

Table 6.2: **Default hyperparameter settings.**

	Parameter	**Value**
General architecture	Hidden dimensions	128
	Number of layers	5
	Activation	ReLU
	Dropout	0.2
	Normalization	DiffGroupNorm [296]
GAT specific	Heads	1
	Attention dropout	0.05
GIN specific	MLP layers	2
	MLP dimensions	256
	ϵ	0.0
Optimizer (AdamW [158])	Learning rate	0.01
	Weight decay	1e-6
	Max epochs	50,000
	Early stopping patience	500
Learning rate scheduler (ReduceLROnPlateau)	Scheduler patience	100
	Scheduler reduce factor	0.5
	Minimum learning rate	1e-5

[3] Uses the symmetric normalized adjacency matrix for propagation.

Table 6.3: **Hyperparameter search space for GCN and GAT on the primary datasets.**

Parameter	Search space
Number of layers	{1, 2, 3, 4, 5, 6, 7, 8}
Hidden dimensions	{64, 128, 192, 256}
Number of heads (GAT)	{1, 2, 3, 4}
Attention dropout (GAT)	{0.0, 0.05, 0.1}
Dropout	{0.1, 0.2, 0.3}
Normalization	{none, BatchNorm, LayerNorm, PairNorm, DiffGroupNorm}
Activation	{ReLU, PReLU, GELU, SELU, ELU}

All hyperparameters are listed in the configuration files for each model in the benchmarking repository[4]. We summarize the primary default hyperparameters for GNN in Table 6.2. For logistic regression baselines, we use the SGD optimizer with a constant learning rate of 200, weight decay of 1e-5, and momentum of 0.9.

Fully tuned GNN models In addition to the baseline GNN experiments where we used the default settings elaborated above, we also tuned GCN and GAT specifically for the four primary benchmarks presented in the main paper with a bag of tricks (BoT). Specifically, we construct node features by concatenating Node2vec embeddings ($d = 128$), OneHotLogDeg encodings ($d = 32$), and LabelReuse. We also apply correct and smooth (C&S) to the GNN predictions as a post-processing step. To search for the dataset-specific optimal hyperparameters for GCN and GAT each dataset, we use the Bayesian search optimization strategy provided by Weights&Biases [28] based on the validation APOP scores. The search space is listed in Table 6.3 and the final hyperparameter settings are summarized in Table 6.4, 6.5.

Table 6.4: **Tuned GCN model on the four primary benchmarks.**

	BioGRID		HumanNet	
	GOBP	DisGeNET	GOBP	DisGeNET
Hidden dimension	192	256	192	64
Number of layers	3	1	4	4
Activation	SELU	PReLU	GELU	ReLU
Dropout	0.3	0.3	0.3	0.3
Normalization	DiffGroupNorm	DiffGroupNorm	–	DiffGroupNorm

[4]https://github.com/krishnanlab/obnbench/tree/main/conf

Table 6.5: **Tuned GAT (v2) model on the four primary benchmarks.**

	BioGRID		HumanNet	
	GOBP	DisGeNET	GOBP	DisGeNET
Hidden dimension	128	64	128	64
Number of layers	4	2	4	8
Number of heads	1	4	1	1
Attention dropout	0.05	0.1	0.1	0.1
Activation	SELU	PReLU	PReLU	GELU
Dropout	0.2	0.3	0.2	0.1
Normalization	BatchNorm	DiffGroupNorm	DiffGroupNorm	DiffGroupNorm

6.3.5 Implementation information

The `obnbench` benchmarking codebase utilizes PyTorch [195] and PyTorch Geometric (PyG) [70] for building deep (graph) neural network models. In addition, we leverage Weights & Bias [28] for tracking training results and Hydra [283] for managing experiment configurations. All benchmarking experiments are run using computational nodes with five CPUs, 45GB memory, and a Tesla V100 GPU (32GB).

6.3.6 Evaluation metric

We use the log_2 fold change of average precision over the prior (APOP) as the primary metric for our benchmarking study. The prior is computed as the ratio between the number of positives and the total number of samples, which corresponds to the probability that a randomly drawn sample is positive. Thus, APOP indicates how much better (> 0), or worse (< 0), a prediction is than a random guess, in two-fold change. More precisely,

$$\text{APOP} = \log_2\left(\frac{\text{Average Precision}}{\text{prior}}\right) = \log_2\left(\frac{\sum_n (R_n - R_{n-1})P_n}{(\#\text{ positives})/(\#\text{ positives} + \#\text{ negatives})}\right) \tag{6.12}$$

where R_n and P_n are the recall and precision and precision at the n^{th} prediction score threshold.

The average precision is related to the the area under the precision and recall curve (AUPRC), which has been shown to be more suitable than the the area under the receiver operator characteristic curve (AUROC) in the case of class imbalance [226, 238]. In our dataset, class imbalance is prevalent, where each class has only one or two hundred of positive examples but thousands of negative examples (Table 6.1b, C.6). Moreover, AUROC is more tolerant of errors made on top

predictions and generally only requires that the global distribution of the predictions made is consistent with the true labels. AUPRC and AP, on the other hand, penalize the top predictions' errors more.

Practical relevance The OBNB benchmarks cover diverse gene prioritization tasks, such as pinpointing relevant genes for a particular disease. This formulation can also be straightforwardly extend to drug recommendation or drug repurposing tasks, by considering known drug targets (genes) as positive examples. For such biomedicine applications, in practice, only a few top predictions made by a predictive method can be experimentally verified by experimentalists due to the high experimental costs. Thus, APOP's emphasis on top predictions aligns well with this practical constraint and encourages methods to make highly accurate top predictions.

6.3.7 Corrected Homophily Ratio

We propose a novel measure of the node classification task's homophily, called *corrected homophily ratio*. This measure extends the traditional definition of homophily [197] to more general setting of multilabel classification with an emphasis of class imbalance.

Let $G = (V, E, w)$ be a weighted graph (unweighted graphs can be treated as weighted graphs with identical edge weights), with node set V, edge set E, and the edge weight function $w : V \times V \to \mathbb{R}$. Let $y_i : V \to \{0, 1\}^p$ be the label function for the i^{th} class (or label), for $i \in \{1, \ldots, p\}$. We define the set of *labeled nodes* V_{labeled} as the ones that are part of at least one label, i.e., $V_{\text{labeled}} = \{v \in V | \max_{i \in 1,\ldots,p} y_i(v) = 1\}$. Furthermore, we define the *positively* and *negatively labeled nodes* for the i^{th} class as $V^{(i+)}_{\text{labeled}} = \{v \in V | y_i(v) = 1\}$ and $V^{(i-)}_{\text{labeled}} = \{v \in V | y_i(v) = 0\}$, respectively

Definition 1 (Node homophily ratio) *The node homophily ratio of the i^{th} class for node v is defined as the ratio of its neighborhood that is positively labeled in the i^{th} class.*

$$h_i(v) = \frac{|\{u \in \mathcal{N}(v) | y_i(u) = 1\}|}{|\mathcal{N}(v)|} \tag{6.13}$$

Definition 2 (Positive and negative homophily ratio) *The positive and negative homophily ratios of the i^{th} class (or label) are defined as the ratio of a node's neighborhood that are positively labeled*

for the i^{th} class averaged over the positively and negatively labeled nodes, respectively

$$h_{i+} = \frac{1}{|V^{(i+)}_{labeled}|} \sum_{v \in V^{(i+)}_{labeled}} h_i(v), \quad h_{i-} = \frac{1}{|V^{(i-)}_{labeled}|} \sum_{v \in V^{(i-)}_{labeled}} h_i(v) \tag{6.14}$$

We refer to the *positive homophily ratio* as the *homophily ratio* for short. Note that definition 1 is slightly different from the typical definition of node homophily that is used in previous works targeting multiclass classification settings [197]. Specifically, (1) we only count positive nodes from the neighborhood instead of nodes having the same class as the central node since we are dealing with (multilabel) binary classification, which will also come in handy later for defining the corrected homophily ratio; (2) we only average the node homophily ratios over the positive node sets since our datasets have a notable class imbalance, where most nodes are negatively labeled. This way, the homophily ratio is less skewed towards the majority of negatively labeled nodes.

Definition 3 (Corrected homophily ratio) *The corrected homophily ratio of the i^{th} class is defined as the expected fold change of node homophily between positively and negatively labeled nodes for class i*

$$\tilde{h}_i = \log_2 \left(\frac{h_{i+}}{h_{i-}} \right) \tag{6.15}$$

This corrected homophily ratio provides an answer to the following question: *How much more likely are nodes labeled in class i to be connected with other nodes labeled in class i compared to nodes not labeled in class i*? A positive value of the corrected homophily ratio indicates that positive nodes in class i have a higher likelihood of interacting with other positive nodes of class i than with negative nodes.

6.4 Results and discussions

6.4.1 Traditional ML methods perform comparably to GNNs overall

Table 6.6 shows the overall performance for the selected model on the four primary benchmarking datasets (full baseline results in Table C.2). Surprisingly, even after a rather extensive hyperparameter search and GNN architecture tuning (Appendix 6.3.4), logistic regression using graph-derived

features still performs comparably to GNNs. For example, *node2vec* achieves the best performance for GOBP prediction when using both BioGRID and HumanNet. Similar results are obtained when using the other networks (Table C.2) and using the more standard random splitting strategy (Table C.4). The superior performance of traditional ML methods over GNNs is in stark contrast with results reported in recent benchmarking studies [232, 64, 99], highlighting the unique characteristics of biological networks and the challenges in modeling them. In addition, we demonstrate that the proposed study-bias holdout splitting provides more stringent evaluations than random splitting (Table C.4).

Table 6.6: **Main performance summary on the four primary OBNB benchmarking datasets.** Performance is evaluated in APOP ↑ aggregated over five random seeds. **Bold** indicates the best-performing method within the method class (top group: traditional ML; bottom group: GNN). Green indicates best performing-method across both classes. See Table C.3 for extended baseline performance references.

			BioGRID		HumanNet	
Model	Features	C&S?	DisGeNET	GOBP	DisGeNET	GOBP
LabelProp	–	✗	**0.931 ± 0.000**	1.885 ± 0.000	**3.059 ± 0.000**	3.806 ± 0.000
LogReg	Adj	✗	0.743 ± 0.000	2.528 ± 0.000	3.053 ± 0.000	3.964 ± 0.006
LogReg	N2V	✗	0.836 ± 0.029	**2.571 ± 0.015**	2.433 ± 0.029	**4.036 ± 0.019**
LogReg	LapEigMap	✗	0.864 ± 0.002	2.149 ± 0.000	2.301 ± 0.000	3.778 ± 0.001
GCN	LogDeg	✗	0.773 ± 0.035	2.022 ± 0.100	2.452 ± 0.107	3.524 ± 0.061
GCN	LogDeg	✓	1.026 ± 0.013	2.201 ± 0.035	2.777 ± 0.031	3.743 ± 0.015
GCN†	N2V+LogDeg+Label	✓	1.014 ± 0.020	2.411 ± 0.044	3.053 ± 0.078	3.921 ± 0.045
GCN‡	N2V+LogDeg+Label	✓	1.012 ± 0.040	2.572 ± 0.066	**3.116 ± 0.017**	3.812 ± 0.071
GAT	LogDeg	✗	0.552 ± 0.111	1.592 ± 0.408	2.547 ± 0.207	3.571 ± 0.159
GAT	LogDeg	✓	1.002 ± 0.018	2.227 ± 0.024	3.007 ± 0.037	3.876 ± 0.053
GAT†	N2V+LogDeg+Label	✓	1.037 ± 0.036	**2.624 ± 0.070**	3.100 ± 0.031	3.908 ± 0.086
GAT‡	N2V+LogDeg+Label	✓	**1.063 ± 0.023**	2.562 ± 0.070	3.065 ± 0.021	**3.963 ± 0.082**

† recommended BoT, ‡ recommended BoT with fully tuned GNNs (see Section 6.3.4 for tuning details)

Carefully designed BoT significantly boosts GNN performance While plain GNNs without BoT yield relatively unsatisfactory performances, carefully crafted BoT can bring significant enhancement to GNNs in our benchmark. First, a systematic benchmark on GCN and GAT with 15 different feature construction reveals that using *node2vec* as node features consistently achieve top performance across datasets (Figure C.1). Second, reusing training labels as features also elevates overall performance, most notably for GAT (Figure C.2). Third, C&S post-processing universally improves performance for both GNN and logistic regression methods, with, on average, 0.27 improvement in test APOP scores (Figure C.2,C.3). In light of these observations, we recommend an optimized BoT as follows:

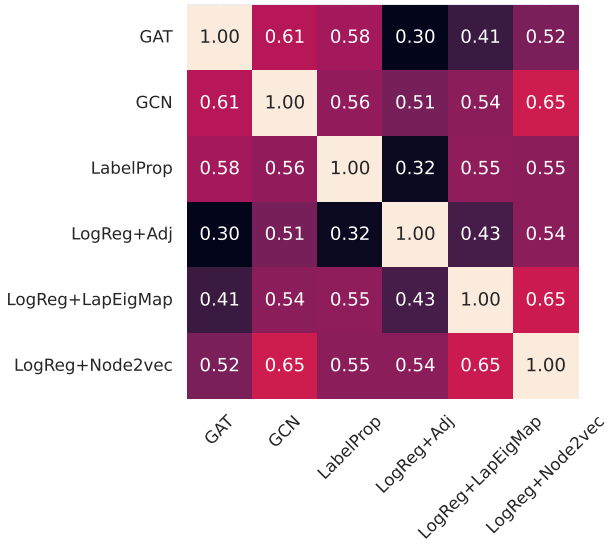


Figure 6.3: **Correlations between methods across tasks in BioGRID-DisGeNET.**

(1) use a combination of LogDeg encodings, *node2vec* embeddings, and training labels as input node features, (2) apply C&S post-processing to refine predictions further. Our results indicate that the recommended BoT helps improve GNNs' performances by 0.36 APOP test scores on average across datasets.

6.4.2 Different models have their own strength

Despite the overall rankings of different methods indicating the superiority of their gene classification capabilities, no single method can achieve the best results across all tasks. We demonstrate this in Figure 6.3, which shows that the performance of most methods does not correlate with each other across different tasks on the BioGRID-DisGeNET dataset. For example, while LogReg+Adj performs better than GAT+LogDeg overall on the BioGRID-DisGeNET dataset, GAT+LogDeg achieves significantly better predictions (t-test p-value < 0.01) for a handful of tasks (Table 6.7), such as iris disorder and ocular vascular disorder.

Focusing on the optimal model for one or a few tasks of interest is utterly essential. This is because, in practice, experimental biologists often come with only one or a few gene sets and want to either obtain new related genes or reprioritize them based on their relevance to the whole gene set [164]. Therefore, understanding the characteristics of the task that for which a particular model

Table 6.7: **Prediction performance discrepancy across tasks.** Task-specific model prediction performance difference on the BioGRID-DisGeNET dataset between GAT and LogReg+Adj.

Task ID	Task Name	GAT	LogReg+Adj	Difference
MONDO:0002289	Iris disorder	2.160	0.149	2.011
MONDO:0001703	Color-vision disease	2.177	0.230	1.946
MONDO:0001926	Ureteral disorder	1.543	-0.336	1.879
MONDO:0005552	Ocular vascular disease	1.507	-0.175	1.682
MONDO:0018470	Renal agenesis	1.273	-0.347	1.620
...				
MONDO:0020018	Cranial malformation	0.699	4.487	-3.788
MONDO:0019743	Nephropathy secondary to a storage or other metabolic disease	0.773	4.876	-4.103
MONDO:0024757	Cardiovascular neoplasm	0.316	4.443	-4.127
MONDO:0045011	Keratinization disease	0.264	4.619	-4.355
MONDO:0015160	Multiple congenital anomalies/dysmorphic syndrome-variable intellectual disability syndrome	0.280	4.828	-4.549

works well over others is the key to making better architectural decisions for a new task of interest and, ultimately, designing new specialized GNN for network biology.

6.4.3 Homophily is a driving factor for good predictions

Homophily describes the tendency of a node's neighborhood to have similar labels to itself. Such effects are prevalent in many real-world graphs, such as citation networks, and have been studied extensively recently in the GNN communities as a way to understand *what* information can or cannot be captured by GNN effectively [159, 162]. Intuitively, homophily aligns well with the *guilt-by-association* principle in network biology, which similarly states that genes that interact with each other are likely functionally related. Despite this clear connection, existing homophily measures, such as the homophily ratio [162], do not straightforwardly apply to the OBNB datasets for two reasons.

1. Most established work on homophily consider *multiclass* classification task, where each node has exactly one class label, contrasting with the *multilabel* classification tasks in OBNB.

2. Label information in OBNB datasets is incredibly sparse. On average, there are only hundreds of positive genes per class out of the total number of 20K genes.

One attempt to resolve the first issue is to compute the average homophily ratio for each class independently. However, due to the label scarcity, the resulting metric can not be readily interpreted.

In particular, the highest homophily ratio observed in our main benchmarking study is about 0.2 (Figure 6.4). Datasets with this value are typically categorized into heterophily (not homophily), contradicting the guilt-by-association principle. To address this inconsistency, we propose a corrected version of the homophily ratio that takes label scarcity into account (Chapter 6.3.7). The main idea is to measure homophily as a relative measurement: *how much more likely nodes labeled in class i are connected with nodes also labeled in class i than those not in class i*? This measure is directly interpretable as it is the log fold change of the homophily between positive and negative examples.

Corrected homophily ratio characterizes prediction performance of graph ML methods As shown in the bottom two panels of Figure 6.4, the corrected homophily ratio is well correlated with the prediction performance across different networks and tasks. This positive correlation unveils the fundamental principle that graph ML methods rely on: the underlying graph must provide sufficient contrast between the neighborhoods of the positive and negative examples. Conversely, the top two panels of Figure 6.4 indicate that the standard homophily ratio fails to adequately account for the performance nuances in low homophily tasks As a result, OBNB opens up new research opportunities in understanding the effects of homophily in GNN by introducing a new type of homophily that differs significantly from those traditionally studied [162].

Graph based augmentation tricks offer most advantages on intermediate homophilic tasks We next investigate the relationship between the corrected homophily ratio and the performance difference between a pair of methods by asking: *is one method systematically better than another method on a particular range of homophily?* The first and second rows of Figure C.4 indicate that GNN augmented with different grpah-based tricks, such as C&S and *node2vec* features, provide most apparent advantage at intermediate homophily (corrected homophily ratio of ~ 2.5). However, as the homophily increases further, the performance differences diminish, suggesting that all methods would perform equally perfectly as the graph provide more clear clues about the labels. Similar trends are observed for the comparison between GAT+BoT and baseline label propagation and logistic regression methods (Figure C.4 third row), where GAT+BoT most significantly outperforms the baselines at intermediate homophily.

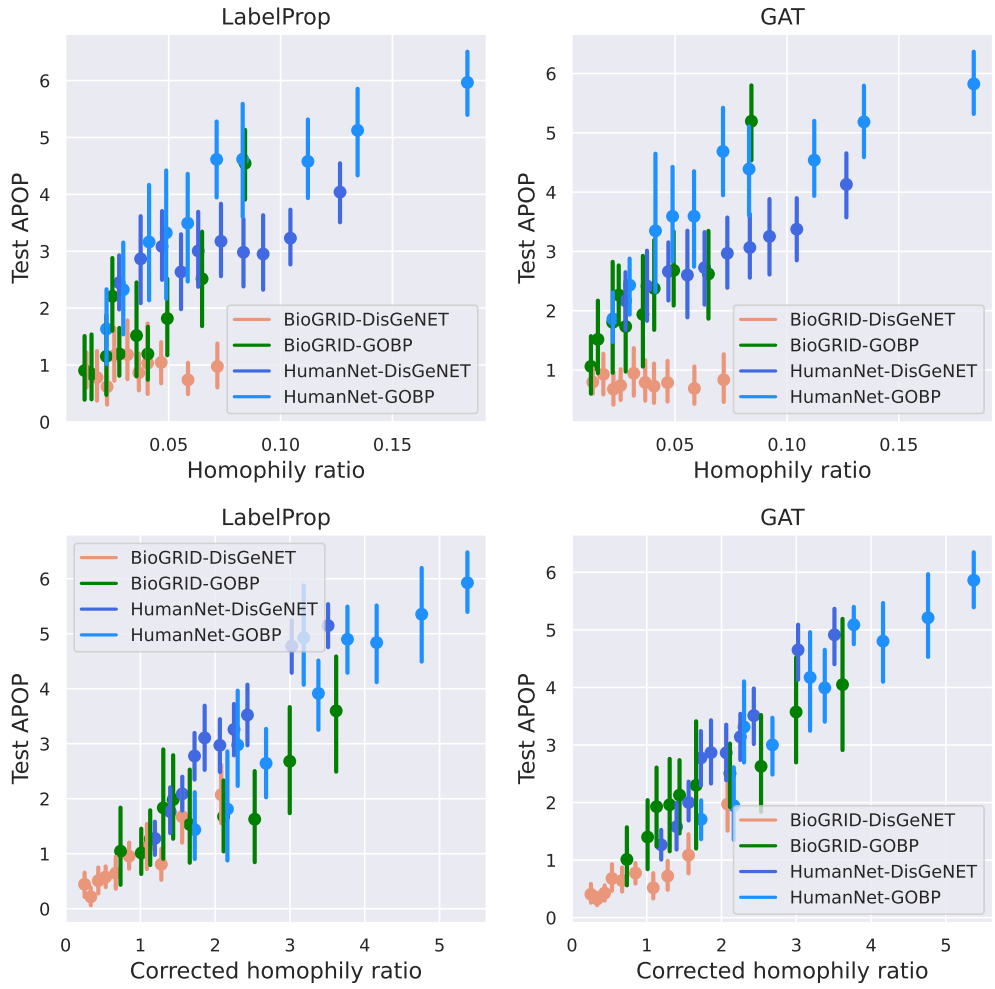


Figure 6.4: **Relationship between the (corrected) homophily ratios of gene classification tasks and the prediction methods' performances.**

6.4.4 Challenges for the OBNB benchmarks and potential future directions

Our systematic benchmarking study paves the way for numerous subsequent explorations. Below, we outline the primary challenges associated with OBNB and suggest potential areas for future research.

- **Challenge 1. Lack of canonical node features** Traditional network biology studies solely rely on biological network structures to gain insights [19, 22, 126]. Meanwhile, the presence of rich node features in many existing graph benchmarks is crucial for the success of GNNs [149], posing a challenge for GNNs in learning without meaningful node features. An exciting and promising future direction for obtaining meaningful node features is by leveraging the

sequential or structural information of the gene product (e.g., protein) using large-scale biological pre-trained language models like ESM-2 [147].

- **Challenge 2. Dense and weighted graphs** Many gene interaction networks are dense and weighted by construction, such as coexpression [113] or integrated functional interactions [246]. In more extreme cases, the weighted networks can be fully connected [80]. The networks used in the OBNB benchmark are orders of magnitude denser than citation networks [232] (Table 6.1a). Thus, scalably and effectively learning from densely weighted gene interaction networks is an area of research to be further explored in the future [150]. One potential solution would be first sparsifying the original dense network before proceeding to train GNN models on them, either by straightforwardly applying an edge cutoff threshold or by using more intricated methods such as spectral sparsification [239].

- **Challenge 3. Scarce labeled data** Curated biological annotations are scarce, posing the challenge of effectively training powerful and expressive models with limited labeled examples. Furthermore, a particular gene can be labeled with more than one biological annotation (multi-label setting), which is much more complex than the multi-class settings in popular benchmarking graph datasets such as Cora and CiteSeer [232]. The data scarcity issue naturally invites the usage of self-supervised [280] graph learning techniques, such as DGI [259], and knowledge transfer via pre-training, such as TxGNN [101]. However, these methods still face the challenge of missing node features (challenge 1).

- **Challenge 4. Low corrected homophily ratio tasks** Our results in Figure 6.4 show that tasks with low corrected homophily ratios tend to be more challenging to predict, indicating that current tested methods are limited to local information of the underlying graph. This naturally opens up opportunities for designing models that (1) better capture long-range dependencies [66] and (2) exploit higher-order structural information [178].

6.5 Conclusion

We have developed `obnb`, an open-source Python package for rapidly setting up ML-ready benchmarking graph datasets using diverse biomedical networks and gene annotations from publicly available sources. `obnb` takes care of tedious data (pre-)processing steps so that network biologists can easily set up particular datasets with their desired settings, and graph ML researchers can directly use the ML-ready datasets for model development. We have established a comprehensive set of baseline performances on OBNB using a wide range of graph ML methods for future reference and pointed out potential improvements that could further enhance performances. Together, OBNB will help accelerate the development of advanced graph ML methods in network biology toward furthering our understanding of the complex genetic basis of human traits and diseases.

CHAPTER 7

CONCLUSION

In this chapter, we summarize the primary results from this dissertation, reflect on current limitations, and highlight promising future directions.

7.1 Summary

This dissertation aimed to further our understanding of human genomics using genome-scale gene interaction networks by identifying genes associated with different biological functions, diseases, and traits. The accurate and complete mapping of these associations is pivotal for developing effective treatment strategies for complex human diseases. My dissertation projects have contributed significantly toward this goal by developing (1) computational frameworks and software for network-based gene classification, (2) efficient and effective algorithms capable of embedding dense genome-scale networks to enable fast and accurate gene classification in low-dimensional spaces, and (3) a biological contextual informed network embedding approach that highlights context-specificity of complex diseases. These results have demonstrated the immense potential for graph deep learning to shed light on human genetics with remarkable accuracy and context-specificity. The open-source software, data, methods, and algorithms developed in this dissertation lay the foundation toward deeper insights into human genomics and can ultimately facilitate the design of effective therapeutic strategies for complex diseases.

7.2 Reflections and limitations

Despite the tremendous progress made in this dissertation toward computational systems leveraging molecular interaction networks to gain insights into human biology, there are some key limitations to be further explored. From a *computational* perspective, challenges persist in (1) analyzing gene sets exhibiting unique characteristics and (2) adapting models to accommodate dynamic changes in the underlying network. On the *biological* side, the current definition of context-specificity is rather simplified, potentially obstructing our exploration of the subtle interplay between diseases and their biological contexts. We discuss these limitations in greater detail below.

Lack of inductive embedding capabilities In GenePlexus (Chapter 2), we have demonstrated immense opportunities for using network-based features to classify genes, which have led to our extended efforts to develop and deploy software and web-services, GenePlexusWeb (Chapter 2.5), to help biomedical researchers explore potential relevant genes given the query gene set of interest. This online service necessitates highly efficient model training and network feature extractions. PecanPy (Chapter 3) has contributed toward this goal by providing a fast and efficient network embedding implementation. Nevertheless, critical challenges remain regarding efficiently adjusting existing network embeddings for minor changes in the network. Particularly, we need to regenerate the network embeddings and subsequently retrain the gene classification models even under the slightest modification of the input gene interaction network, such as adding or removing edges, either by adapting newly released data or accounting for the biological context-specificity. Addressing this limitation will significantly speed up the online training and inference time. *Inductive representation learning* [89] aims to tackle this by generating the network embeddings inductively given the sampled neighborhoods. However, its straightforward application is hindered by the lack of canonical features for genes (Chapter 6.4.4).

Low homophily As we have first observed in GenePlexus and later formally analyzed in OBNB via the notion of *corrected homophily ratio* (Chapter 6.3.7), network-based approaches suffered from accurately predicting genes that are "scattered over the network." More precisely, these are the gene sets whose positive and negative examples do not show enough contrast in their neighborhood compositions. Interestingly, such low-homophily gene sets are typically related to rare diseases and complex traits, while functional gene sets are typically more localized in the underlying network (Chapter 2.3.2). One explanation for this is that molecular interaction networks are primarily intended, either through curation or construction, to reflect biological relationships between genes and proteins as they pertain to "normal" cellular functions, whereas complex diseases and traits are often products of (partially) affecting several biological pathways and modules spanning across the genome-scale network [19, 72]. Addressing this computational challenge will greatly enhance our understanding of complex diseases by providing an accurate and complete view of their associated

genes. Some recent works have explored solutions to the low homophily problem in general graph learning [162, 149], but their effectiveness in the biological network domain remains elusive.

Simplistic context-specificity definition Biological context-specificity, such as tissues and cell types, is vital for studying complex human diseases [80, 93, 125]. In CONE (Chapter 5), we stepped toward this direction by defining and using biological contextual information as gene sets, such as tissue-specific genes. The core idea was that a pair of genes interact under a particular biological context if (1) they interact in the context-naive setting and (2) both genes are expressive in that context. Despite its efficacy in studying the context-specific gene interactions, as demonstrated by CONE and other related works [228, 140], this simplistic approach disregards intricacies in context-specific gene and protein interactions in real cellular systems, limiting our ability to fully investigate the molecular mechanisms underlying complex diseases in their respective biological contexts. For example, interactions between two proteins could depend on (1) the presence of another protein, such as scaffold proteins [59] and small ubiquitin-like modifier (SUMO) proteins [92, 169], (2) their subcellular localizations [105, 288], and (3) post-translational modifications [45, 143]. Several works have presented context-specific gene interaction networks that are constructed computationally [80, 175, 43], experimentally [215], or via hybrid approaches [211]. However, how to efficiently learn contextualized embeddings on them with a versatile applicability to any "context-naive" backbone network, as CONE does, needs to be further explored.

7.3 Future directions

We plan to continuously enhance the GenePlexus tools and services (Chapter 2) to better assist experimental biologists in making insightful discoveries. First, we aim to expand the tutorials and resources for GenePlexus, ensuring it is straightforward for experimental biologists to utilize. The extended efforts will include organizing workshops to present GenePlexus tutorials and showcasing case studies. Second, we intend to improve user experiences by enabling users to upload their list of gene interactions, which can be superimposed on the existing genome-scale interaction networks. This feature will allow biologists with specialized knowledge of specific gene interactions in particular biological contexts to use this information to make context-specific predictions. We

also plan to incorporate a feedback mechanism where users can evaluate the predictions as correct or incorrect. This feedback can serve as crowd-sourced data to further enhance the accuracy of GenePlexus predictions.

In addition, the advancements made in this dissertation are not limited to gene classifications. In the following, we discuss promising future directions adapting this dissertation by integrating with cutting-edge foundation models and extending to critical applications in transnational biomedical research.

Integrating structural and sequential information through foundation models The recent surge of foundation models for proteins and genomics represents a groundbreaking shift in how genomics data can be analyzed and understood [115, 147, 185]. These models have shown remarkable success in extracting meaningful representations from proteins and genomic sequences. Integrating these models with gene interaction networks will enable many exciting research opportunities. First, this allows one to train *inductive* models by using the extracted features as raw gene features, paving the way for dynamic updates to network embeddings. Second, using genomic foundation model representations presents the possibility of mitigating the *low homophily* challenge by providing complementary information about genes from a different modality than gene interactions. Consequently, embracing this innovative approach could advance our understanding of complex diseases and traits.

Understanding transcriptional and phenotypic effects of genes This dissertation primarily focused on unveiling genes' functional and disease associations. Yet, the *transcriptional* and *phenotypic* effects of the dysfunctioning or dysregulation of every gene (and their combinations) represent a critical, and even more significant, area of study. Recent works, such as GEARS [222], have pioneered in this direction. In GEARS, the integration of a specially constructed functional gene interaction network is instrumental to the success of GEARS in predicting the transcriptional outcomes of combinatorial gene perturbations that were not encountered during its training phase. Future research leveraging biologically contextualized gene interaction network embeddings, such as CONE (Chapter 5), holds immense promise for deepening our understanding of transcriptional

dynamics under more physiologically relevant conditions, extending beyond the confines of cell line studies [222]. Such advancements present profound implications for developing personalized therapeutic strategies against complex diseases.

Annotating functions and diseases for long non-coding RNAs Aside from the 20K protein-coding genes in the human genome that we primarily focused on in this dissertation, there has been growing interest in studying long non-coding RNAs (lncRNAs). Accumulating evidence indicates that many lncRNAs play pivotal functional roles, with their dysregulation linked to numerous rare diseases [8, 179]. Yet, our current knowledge of lncRNAs remains much scarcer than that of protein-coding genes – a gap that is further complicated by their incredibly context-specific manifestations and functions [170, 11]. Recent advancements in network biology, particularly through utilizing molecular interaction networks that integrate lncRNA interactions, offer promising solutions [208, 184]. Similar to proteins, lncRNAs follow the *guilt-by-association* principle: lncRNAs are likely to co-function with their network interaction partners [243]. Therefore, the GDL methods developed in this work, especially the one that is biological context-specific, pave the way toward demystifying lncRNAs and expanding therapeutic options for rare diseases through them.

Aiding drug repurposing and *de novo* design for treating complex diseases The precise mapping of gene functions and their associations with diseases and traits marks a pivotal stride toward a complete picture of human biology, presenting opportunities for effective and personalized treatments for human diseases. Several recent studies have highlighted the efficacy of network biology in elucidating disease mechanisms [19, 72] and proposing viable therapeutic strategies [95, 13, 225]. The advancements presented in this dissertation lay the groundwork for future research to develop predictive models using GDL on molecular interaction networks to suggest treatments tailored to individuals, which can be seen as a form of biological context. Additionally, integrating these approaches with genomics [147, 185] and molecular [221, 223] foundation models, as discussed above, will further open exciting avenues for repurposing existing [234] and designing *de novo* drugs [244, 176], presenting promises for tackling rare and complex diseases that currently lack viable treatment options.

BIBLIOGRAPHY

[1] Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alexander A Alemi. Watch your step: Learning node embeddings via graph attention. *Advances in neural information processing systems*, 31, 2018.

[2] Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B Aditya Prakash. Sub2vec: Feature learning for subgraphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 170–182. Springer, 2018.

[3] Elmar Aigner, Alexandra Feldman, and Christian Datz. Obesity as an emerging risk factor for iron deficiency. *Nutrients*, 6(9):3587–3600, 2014.

[4] Madhu Alagiri, Sherman Chottiner, Vicki Ratner, Debra Slade, and Philip M Hanno. Interstitial cystitis: unexplained associations with other chronic disease and pain syndromes. *Urology*, 49(5):52–57, 1997.

[5] Gregorio Alanis-Lobato, Miguel A Andrade-Navarro, and Martin H Schaefer. Hippie v2. 0: enhancing meaningfulness and reliability of protein–protein interaction networks. *Nucleic acids research*, page gkw985, 2016.

[6] Réka Albert. Scale-free networks in cell biology. *Journal of cell science*, 118(21):4947–4957, 2005.

[7] Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Sahand Sharifzadeh, Volker Tresp, and Jens Lehmann. Pykeen 1.0: A python library for training and evaluating knowledge graph embeddings. *J. Mach. Learn. Res.*, 22(82):1–6, 2021.

[8] Shabana A Ali, Mandy J Peffers, Michelle J Ormseth, Igor Jurisica, and Mohit Kapoor. The non-coding rna interactome in joint health and disease. *Nature Reviews Rheumatology*, 17(11):692–705, 2021.

[9] Motasem Alkhayyat, Mohannad Abou Saleh, Mehnaj Kaur Grewal, Mohammad Abureesh, Emad Mansoor, C Roberto Simons-Linares, Abby Abelson, and Prabhleen Chahal. Pancreatic manifestations in rheumatoid arthritis: a national population-based study. *Rheumatology*, 60(5):2366–2374, 2021.

[10] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[11] Paulo Amaral, Silvia Carbonell-Sala, Francisco M De La Vega, Tiago Faial, Adam Frankish, Thomas Gingeras, Roderic Guigo, Jennifer L Harrow, Artemis G Hatzigeorgiou, Rory Johnson, et al. The status of the human gene catalogue. *Nature*, 622(7981):41–47, 2023.

[12] Lena Antoni, Sabine Nuding, Jan Wehkamp, and Eduard F Stange. Intestinal barrier in inflammatory bowel disease. *World journal of gastroenterology: WJG*, 20(5):1165, 2014.

[13] DK Arrell and Andre Terzic. Network systems biology for drug discovery. *Clinical Pharmacology & Therapeutics*, 88(1):120–125, 2010.

[14] Michael Ashburner, Catherine A Ball, Judith A Blake, David Botstein, Heather Butler, J Michael Cherry, Allan P Davis, Kara Dolinski, Selina S Dwight, Janan T Eppig, et al. Gene ontology: tool for the unification of biology. *Nature genetics*, 25(1):25–29, 2000.

[15] Sezin Kircali Ata, Le Ou-Yang, Yuan Fang, Chee-Keong Kwoh, Min Wu, and Xiao-Li Li. Integrating node embeddings and biological annotations for genes to predict disease-gene associations. *BMC systems biology*, 12(9):31–44, 2018.

[16] Sezin Kircali Ata, Min Wu, Yuan Fang, Le Ou-Yang, Chee Keong Kwoh, and Xiao-Li Li. Recent advances in network-based methods for disease gene prediction. *Briefings in bioinformatics*, 22(4):bbaa303, 2021.

[17] Awais Athar, Anja Füllgrabe, Nancy George, Haider Iqbal, Laura Huerta, Ahmed Ali, Catherine Snow, Nuno A Fonseca, Robert Petryszak, Irene Papatheodorou, et al. Arrayexpress update–from bulk to single-cell expression data. *Nucleic acids research*, 47(D1):D711–D715, 2019.

[18] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[19] Albert-László Barabási, Natali Gulbahce, and Joseph Loscalzo. Network medicine: a network-based approach to human disease. *Nature reviews genetics*, 12(1):56–68, 2011.

[20] Albert-László Barabási, Natali Gulbahce, and Joseph Loscalzo. Network medicine: a network-based approach to human disease. *Nature reviews genetics*, 12(1):56–68, 2011.

[21] Albert-László Barabási and Zoltán N. Oltvai. Network biology: understanding the cell's functional organization. *Nature Reviews Genetics*, 5(2):101–113, February 2004.

[22] Albert-Laszlo Barabasi and Zoltan N Oltvai. Network biology: understanding the cell's functional organization. *Nature reviews genetics*, 5(2):101–113, 2004.

[23] Jonathan BL Bard and Seung Y Rhee. Ontologies in biology: design, applications and future challenges. *nature reviews genetics*, 5(3):213–222, 2004.

[24] Zafer Barutcuoglu, Robert E Schapire, and Olga G Troyanskaya. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7):830–836, 2006.

[25] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and

data representation. *Neural computation*, 15(6):1373–1396, 2003.

[26] Yoav Benjamini, Abba M Krieger, and Daniel Yekutieli. Adaptive linear step-up procedures that control the false discovery rate. *Biometrika*, 93(3):491–507, 2006.

[27] Mary Lauren Benton, Abin Abraham, Abigail L LaBella, Patrick Abbot, Antonis Rokas, and John A Capra. The influence of evolutionary history on human health and disease. *Nature Reviews Genetics*, 22(5):269–283, 2021.

[28] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[29] Jonathan JH Bray, Rosie Freer, Alex Pitcher, and Rajesh Kharbanda. Family screening for bicuspid aortic valve and aortic dilatation: a meta-analysis. *European Heart Journal*, page ehad320, 2023.

[30] Anna Breit, Simon Ott, Asan Agibetov, and Matthias Samwald. Openbiolink: a benchmarking framework for large-scale biomedical link prediction. *Bioinformatics*, 36(13):4097–4098, 2020.

[31] Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

[32] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2021.

[33] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

[34] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, July 2017.

[35] Garth R Brown, Vichet Hem, Kenneth S Katz, Michael Ovetsky, Craig Wallin, Olga Ermolaeva, Igor Tolstoy, Tatiana Tatusova, Kim D Pruitt, Donna R Maglott, et al. Gene: a gene-centered information resource at ncbi. *Nucleic acids research*, 43(D1):D36–D42, 2015.

[36] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[37] Carol J Bult, Judith A Blake, Cynthia L Smith, James A Kadin, and Joel E Richardson. Mouse genome database (mgd) 2019. *Nucleic acids research*, 47(D1):D801–D806, 2019.

[38] Annalisa Buniello, Jacqueline A L MacArthur, Maria Cerezo, Laura W Harris, James Hayhurst, Cinzia Malangone, Aoife McMahon, Joannella Morales, Edward Mountjoy, Elliot Sollis, et al. The nhgri-ebi gwas catalog of published genome-wide association studies, targeted arrays and summary statistics 2019. *Nucleic acids research*, 47(D1):D1005–D1012, 2019.

[39] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[40] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900, 2015.

[41] Luca Cappelletti, Tommaso Fontana, Elena Casiraghi, Vida Ravanmehr, Tiffany J. Callahan, Carlos Cano, Marcin P. Joachimiak, Christopher J. Mungall, Peter N. Robinson, Justin Reese, and Giorgio Valentini. GRAPE for fast and scalable graph processing and random-walk-based embedding. *Nature Computational Science*, 3(6):552–568, June 2023.

[42] Giovanni Casella, Gian Eugenio Tontini, Gabrio Bassotti, Luca Pastorelli, Vincenzo Villanacci, Luisa Spina, Vittorio Baldini, and Maurizio Vecchi. Neurological disorders and inflammatory bowel diseases. *World Journal of Gastroenterology: WJG*, 20(27):8764, 2014.

[43] Junha Cha, Jiwon Yu, Jae-Won Cho, Martin Hemberg, and Insuk Lee. schumannet: a single-cell network analysis platform for the study of cell-type specificity of disease genes. *Nucleic acids research*, 51(2):e8–e8, 2023.

[44] Hammad S. Chaudhry and Madhukar Reddy Kasarla. *Microcytic Hypochromic Anemia*. StatPearls Publishing, Treasure Island (FL), 2022.

[45] Nana Cheng, Mingzhu Liu, Wanting Li, BingYue Sun, Dandan Liu, Guoqing Wang, Jingwei Shi, and Lisha Li. Protein post-translational modification in sars-cov-2 and host interaction. *Frontiers in Immunology*, 13:1068449, 2023.

[46] Hyunghoon Cho, Bonnie Berger, and Jian Peng. Compact integration of multi-network topology for functional analysis of genes. *Cell systems*, 3(6):540–548, 2016.

[47] Sarvenaz Choobdar, Mehmet E Ahsen, Jake Crawford, Mattia Tomasoni, Tao Fang, David Lamparter, Junyuan Lin, Benjamin Hescott, Xiaozhe Hu, Johnathan Mercer, et al. Assessment of network module identification across complex diseases. *Nature methods*, 16(9):843–852, 2019.

[48] Doreen E Chung, Lesley K Carr, Linda Sugar, Michelle Hladunewich, and Leslie A Deane. Xanthogranulomatous cystitis associated with inflammatory bowel disease. *Canadian Urological Association Journal*, 4(4):E91, 2010.

[49] Gene Ontology Consortium. The gene ontology resource: 20 years and still going strong. *Nucleic acids research*, 47(D1):D330–D338, 2019.

[50] Tabula Sapiens Consortium*, Robert C Jones, Jim Karkanias, Mark A Krasnow, Angela Oliveira Pisco, Stephen R Quake, Julia Salzman, Nir Yosef, Bryan Bulthaup, Phillip Brown, et al. The tabula sapiens: A multiple-organ, single-cell transcriptomic atlas of humans. *Science*, 376(6594):eabl4896, 2022.

[51] Lenore Cowen, Trey Ideker, Benjamin J Raphael, and Roded Sharan. Network propagation: a universal amplifier of genetic associations. *Nature Reviews Genetics*, 18(9):551–562, 2017.

[52] Hejie Cui, Zijie Lu, Pan Li, and Carl Yang. On positional and structural node features for graph neural networks on non-attributed graphs. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 3898–3902, 2022.

[53] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. A survey on network embedding. *IEEE transactions on knowledge and data engineering*, 31(5):833–852, 2018.

[54] Patrick Danaher, Pei Wang, and Daniela M Witten. The joint graphical lasso for inverse covariance estimation across multiple classes. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 76(2):373–397, 2014.

[55] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240, 2006.

[56] Andrew Davison and Morgane Austern. Asymptotics of network embeddings learned via subsampling. *Journal of Machine Learning Research*, 24(138):1–120, 2023.

[57] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016.

[58] Christophe Dessimoz, Nives Škunca, and Paul D Thomas. Cafa and the open world of protein function predictions. *Trends in Genetics*, 29(11):609–610, 2013.

[59] DN Dhanasekaran, K Kashef, CM Lee, H Xu, and EP Reddy. Scaffold proteins of map-kinase modules. *Oncogene*, 26(22):3185–3202, 2007.

[60] Scott J Dixon, Michael Costanzo, Anastasia Baryshnikova, Brenda Andrews, Charles Boone, et al. Systematic mapping of genetic interaction networks. *Annual review of genetics*, 43(1):601–625, 2009.

[61] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *Proceedings of the 23rd ACM SIGKDD*

international conference on knowledge discovery and data mining, pages 135–144, 2017.

[62] Kevin Drew, John B Wallingford, and Edward M Marcotte. hu. map 2.0: integration of over 15,000 proteomic experiments builds a global compendium of human multiprotein assemblies. *Molecular Systems Biology*, 17(5):e10016, 2021.

[63] Dongsheng Duan, Nathalie Goemans, Shin'ichi Takeda, Eugenio Mercuri, and Annemieke Aartsma-Rus. Duchenne muscular dystrophy. *Nature Reviews Disease Primers*, 7(1):13, 2021.

[64] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023.

[65] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph neural networks with learnable structural and positional representations. In *International Conference on Learning Representations*, 2021.

[66] Vijay Prakash Dwivedi, Ladislav Rampášek, Michael Galkin, Ali Parviz, Guy Wolf, Anh Tuan Luu, and Dominique Beaini. Long range graph benchmark. *Advances in Neural Information Processing Systems*, pages 22326–22340, 2022.

[67] Ron Edgar, Michael Domrachev, and Alex E Lash. Gene expression omnibus: Ncbi gene expression and hybridization array data repository. *Nucleic acids research*, 30(1):207–210, 2002.

[68] Viola Fanfani, Fabio Cassano, and Giovanni Stracquadanio. Pygna: a unified framework for geneset network analysis. *BMC bioinformatics*, 21(1):1–22, 2020.

[69] José M Ferro and Miguel Oliveira Santos. Neurology of inflammatory bowel disease. *Journal of the Neurological Sciences*, 424:117426, 2021.

[70] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[71] Duncan T Forster, Sheena C Li, Yoko Yashiroda, Mami Yoshimura, Zhijian Li, Luis Alberto Vega Isuhuaylas, Kaori Itto-Nakama, Daisuke Yamanaka, Yoshikazu Ohya, Hiroyuki Osada, et al. Bionic: biological network integration using convolutions. *Nature Methods*, 19(10):1250–1261, 2022.

[72] Laura I Furlong. Human diseases through the lens of network biology. *Trends in genetics*, 29(3):150–159, 2013.

[73] Jesse Gillis and Paul Pavlidis. The impact of multifunctional genes on" guilt by association" analysis. *PloS one*, 6(2):e17258, 2011.

[74] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, Proceedings of Machine Learning Research, 2017.

[75] Vladimir Gligorijević, Meet Barot, and Richard Bonneau. deepnf: deep network fusion for protein function prediction. *Bioinformatics*, 34(22):3873–3881, 2018.

[76] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[77] Kwang-Il Goh, Michael E Cusick, David Valle, Barton Childs, Marc Vidal, and Albert-László Barabási. The human disease network. *Proceedings of the National Academy of Sciences*, 104(21):8685–8690, 2007.

[78] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[79] Kristian A Gray, Bethan Yates, Ruth L Seal, Mathew W Wright, and Elspeth A Bruford. Genenames. org: the hgnc resources in 2015. *Nucleic acids research*, 43(D1):D1079–D1085, 2015.

[80] Casey S Greene, Arjun Krishnan, Aaron K Wong, Emanuela Ricciotti, Rene A Zelaya, Daniel S Himmelstein, Ran Zhang, Boris M Hartmann, Elena Zaslavsky, Stuart C Sealfon, et al. Understanding multicellular function and disease with human tissue-specific networks. *Nature genetics*, 47(6):569–576, 2015.

[81] Dhouha Grissa, Alexander Junge, Tudor I Oprea, and Lars Juhl Jensen. Diseases 2.0: a weekly updated database of disease–gene associations from text mining and data integration. *Database*, 2022, 2022.

[82] Martin Grohe. word2vec, node2vec, graph2vec, x2vec: Towards a theory of vector embeddings of structured data. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 1–16, 2020.

[83] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, 2016.

[84] Yuanfang Guan, Cheryl L Ackert-Bicknell, Braden Kell, Olga G Troyanskaya, and Matthew A Hibbs. Functional genomics complements quantitative genetics in identifying disease-gene associations. *PLoS computational biology*, 6(11):e1000991, 2010.

[85] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 297–304. JMLR Workshop and Conference Proceedings, 2010.

[86] Celia Hacker. k-simplex2vec: a simplicial extension of node2vec. *arXiv preprint arXiv:2010.05636*, 2020.

[87] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[88] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *ArXiv preprint*, 2017.

[89] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2017.

[90] Myat Noe Han, David I Finkelstein, Rachel M McQuade, and Shanti Diwakarla. Gastrointestinal dysfunction in parkinson's disease: current and potential therapeutics. *Journal of Personalized Medicine*, 12(2):144, 2022.

[91] SM Mahmudul Hasan and Baljinder S Salh. Emphysematous cystitis as a potential marker of severe crohn's disease. *BMC gastroenterology*, 22(1):181, 2022.

[92] Ronald T Hay. Sumo: a history of modification. *Molecular cell*, 18(1):1–12, 2005.

[93] Idan Hekselman and Esti Yeger-Lotem. Mechanisms of tissue and cell-type specificity in heritable traits and diseases. *Nature Reviews Genetics*, 21(3):137–150, 2020.

[94] Daniel Himmelstein, Casey Greene, and Alexander Pico. Using entrez gene as our gene vocabulary, February 2015.

[95] Andrew L Hopkins. Network pharmacology: the next paradigm in drug discovery. *Nature chemical biology*, 4(11):682–690, 2008.

[96] Congxue Hu, Tengyue Li, Yingqi Xu, Xinxin Zhang, Feng Li, Jing Bai, Jing Chen, Wenqi Jiang, Kaiyue Yang, Qi Ou, et al. Cellmarker 2.0: an updated database of manually curated cell markers in human/mouse and web tools based on scrna-seq data. *Nucleic Acids Research*, 51(D1):D870–D876, 2023.

[97] Fang Hu, Jia Liu, Liuhuan Li, and Jun Liang. Community detection in complex networks using node2vec with spectral clustering. *Physica A: Statistical Mechanics and its Applications*,

545:123633, 2020.

[98] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021.

[99] 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. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[100] Justin K Huang, Daniel E Carlin, Michael Ku Yu, Wei Zhang, Jason F Kreisberg, Pablo Tamayo, and Trey Ideker. Systematic evaluation of molecular networks for discovery of disease genes. *Cell systems*, 6(4):484–495, 2018.

[101] Kexin Huang, Payal Chandak, Qianwen Wang, Shreyas Havaldar, Akhil Vaid, Jure Leskovec, Girish Nadkarni, Benjamin S Glicksberg, Nils Gehlenborg, and Marinka Zitnik. Zero-shot prediction of therapeutic use with geometric deep learning and clinician centered design. *medRxiv*, pages 2023–03, 2023.

[102] Kexin Huang, Tianfan Fu, Wenhao Gao, Yue Zhao, Yusuf Roohani, Jure Leskovec, Connor Coley, Cao Xiao, Jimeng Sun, and Marinka Zitnik. Therapeutics data commons: Machine learning datasets and tasks for drug discovery and development. *Advances in neural information processing systems*, 2021.

[103] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R. Benson. Combining label propagation and simple models out-performs graph neural networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

[104] T Hubbard, D Andrews, Mario Cáccamo, Graham Cameron, Yuan Chen, M Clamp, Laura Clarke, Guy Coates, Tony Cox, Fiona Cunningham, et al. Ensembl 2005. *Nucleic acids research*, 33(suppl_1):D447–D453, 2005.

[105] Mien-Chie Hung and Wolfgang Link. Protein localization in disease and therapy. *Journal of cell science*, 124(20):3381–3392, 2011.

[106] Edward L Huttlin, Raphael J Bruckner, Jose Navarrete-Perea, Joe R Cannon, Kurt Baltier, Fana Gebreab, Melanie P Gygi, Alexandra Thornock, Gabriela Zarraga, Stanley Tam, et al. Dual proteome-scale networks reveal cell-specific remodeling of the human interactome. *Cell*, 184(11):3022–3040, 2021.

[107] Sohyun Hwang, Chan Yeong Kim, Sunmo Yang, Eiru Kim, Traver Hart, Edward M Marcotte, and Insuk Lee. Humannet v2: human gene networks for disease research. *Nucleic acids research*, 47(D1):D573–D580, 2019.

[108] Trey Ideker and Roded Sharan. Protein networks in disease. *Genome research*, 18(4):644–652, 2008.

[109] Elisabetta Indelicato and Sylvia Boesch. From genotype to phenotype: expanding the clinical spectrum of cacna1a variants in the era of next generation sequencing. *Frontiers in Neurology*, 12:263, 2021.

[110] Hawoong Jeong, Sean P Mason, A-L Barabási, and Zoltan N Oltvai. Lethality and centrality in protein networks. *Nature*, 411(6833):41–42, 2001.

[111] Junzhong Ji, Aidong Zhang, Chunnian Liu, Xiaomei Quan, and Zhijun Liu. Survey: Functional module detection from protein-protein interaction networks. *IEEE Transactions on Knowledge and Data Engineering*, 26(2):261–277, 2012.

[112] Yuxiang Jiang, Tal Ronnen Oron, Wyatt T Clark, Asma R Bankapur, Daniel D'Andrea, Rosalba Lepore, Christopher S Funk, Indika Kahanda, Karin M Verspoor, Asa Ben-Hur, et al. An expanded evaluation of protein function prediction methods shows an improvement in accuracy. *Genome biology*, 17:1–19, 2016.

[113] Kayla A Johnson and Arjun Krishnan. Robust normalization and transformation techniques for constructing gene coexpression networks from rna-seq data. *Genome biology*, 23(1):1–26, 2022.

[114] Pall F Jonsson and Paul A Bates. Global topological features of cancer proteins in the human interactome. *Bioinformatics*, 22(18):2291–2297, 2006.

[115] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[116] Indika Kahanda, Christopher S Funk, Fahad Ullah, Karin M Verspoor, and Asa Ben-Hur. A close look at protein function prediction evaluation protocols. *GigaScience*, 4(1):s13742–015, 2015.

[117] Atanas Kamburov, Konstantin Pentchev, Hanna Galicka, Christoph Wierling, Hans Lehrach, and Ralf Herwig. Consensuspathdb: toward a more complete picture of cell biology. *Nucleic acids research*, 39(suppl_1):D712–D717, 2011.

[118] Minoru Kanehisa. The kegg database. In *'In Silico'Simulation of Biological Processes: Novartis Foundation Symposium 247*, volume 247, pages 91–103. Wiley Online Library, 2002.

[119] Minoru Kanehisa, Miho Furumichi, Mao Tanabe, Yoko Sato, and Kanae Morishima. Kegg: new perspectives on genomes, pathways, diseases and drugs. *Nucleic acids research*,

45(D1):D353–D361, 2017.

[120] Ulas Karaoz, TM Murali, Stan Letovsky, Yu Zheng, Chunming Ding, Charles R Cantor, and Simon Kasif. Whole-genome annotation by using evidence integration in functional-linkage networks. *Proceedings of the National Academy of Sciences*, 101(9):2888–2893, 2004.

[121] Chan Yeong Kim, Seungbyn Baek, Junha Cha, Sunmo Yang, Eiru Kim, Edward M Marcotte, Traver Hart, and Insuk Lee. Humannet v3: an improved database of human gene networks for disease research. *Nucleic acids research*, 50(D1):D632–D639, 2022.

[122] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[123] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *ArXiv preprint*, 2016.

[124] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[125] Maksim Kitsak, Amitabh Sharma, Jörg Menche, Emre Guney, Susan Dina Ghiassian, Joseph Loscalzo, and Albert-László Barabási. Tissue specificity of human disease module. *Scientific reports*, 6(1):35241, 2016.

[126] Sebastian Köhler, Sebastian Bauer, Denise Horn, and Peter N Robinson. Walking the interactome for prioritization of candidate disease genes. *The American Journal of Human Genetics*, 82(4):949–958, 2008.

[127] Max Kotlyar, Chiara Pastrello, Nicholas Sheahan, and Igor Jurisica. Integrated interactions database: tissue-specific view of the human and model organism interactomes. *Nucleic acids research*, 44(D1):D536–D541, 2016.

[128] Arjun Krishnan, Ran Zhang, Victoria Yao, Chandra L Theesfeld, Aaron K Wong, Alicja Tadych, Natalia Volfovsky, Alan Packer, Alex Lash, and Olga G Troyanskaya. Genome-wide prediction and functional characterization of the genetic basis of autism spectrum disorder. *Nature neuroscience*, 19(11):1454–1462, 2016.

[129] Arjun Krishnan, Ran Zhang, Victoria Yao, Chandra L Theesfeld, Aaron K Wong, Alicja Tadych, Natalia Volfovsky, Alan Packer, Alex Lash, and Olga G Troyanskaya. Genome-wide prediction and functional characterization of the genetic basis of autism spectrum disorder. *Nature neuroscience*, 19(11):1454–1462, 2016.

[130] Georg Kustatscher, Tom Collins, Anne-Claude Gingras, Tiannan Guo, Henning Hermjakob, Trey Ideker, Kathryn S Lilley, Emma Lundberg, Edward M Marcotte, Markus Ralser, et al. Understudied proteins: opportunities and challenges for functional proteomics. *Nature*

Methods, 19(7):774–779, 2022.

[131] Georg Kustatscher, Piotr Grabowski, Tina A Schrader, Josiah B Passmore, Michael Schrader, and Juri Rappsilber. Co-regulation map of the human proteome enables identification of protein functions. *Nature biotechnology*, 37(11):1361–1371, 2019.

[132] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6, 2015.

[133] Gert RG Lanckriet, Minghua Deng, Nello Cristianini, Michael I Jordan, and William Stafford Noble. Kernel-based data fusion and its application to protein function prediction in yeast. In *Biocomputing 2004*, pages 300–311. World Scientific, 2003.

[134] Peter Laslo, Jagan MR Pongubala, David W Lancki, and Harinder Singh. Gene regulatory networks directing myeloid and lymphoid cell fates within the immune system. In *Seminars in immunology*, volume 20, pages 228–235. Elsevier, 2008.

[135] Jeffrey N Law, Shiv D Kale, and TM Murali. Accurate and efficient gene function prediction using a multi-bacterial network. *Bioinformatics*, 37(6):800–806, 2021.

[136] Insuk Lee, U Martin Blom, Peggy I Wang, Jung Eun Shim, and Edward M Marcotte. Prioritizing candidate disease genes by network-based boosting of genome-wide association data. *Genome research*, 21(7):1109–1121, 2011.

[137] Rasko Leinonen, Hideaki Sugawara, Martin Shumway, and International Nucleotide Sequence Database Collaboration. The sequence read archive. *Nucleic acids research*, 39(suppl_1):D19–D21, 2010.

[138] Mark DM Leiserson, Fabio Vandin, Hsin-Ta Wu, Jason R Dobson, Jonathan V Eldridge, Jacob L Thomas, Alexandra Papoutsaki, Younhun Kim, Beifang Niu, Michael McLellan, et al. Pan-cancer network analysis identifies combinations of rare somatic mutations across pathways and protein complexes. *Nature genetics*, 47(2):106–114, 2015.

[139] Po Sing Leung and Siu Po Ip. Pancreatic acinar cell: its role in acute pancreatitis. *The international journal of biochemistry & cell biology*, 38(7):1024–1030, 2006.

[140] Michelle M Li, Yepeng Huang, Marissa Sumathipala, Man Qing Liang, Alberto Valdeolivas, Ashwin N Ananthakrishnan, Katherine Liao, Daniel Marbach, and Marinka Zitnik. Contextualizing protein representations using deep learning on protein networks and single-cell data. *bioRxiv*, pages 2023–07, 2023.

[141] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning. *Advances in Neural Information Processing Systems*, 33:4465–4478, 2020.

[142] Taibo Li, Rasmus Wernersson, Rasmus B Hansen, Heiko Horn, Johnathan Mercer, Greg Slodkowicz, Christopher T Workman, Olga Rigina, Kristoffer Rapacki, Hans H Stærfeldt, et al. A scored human protein–protein interaction network to catalyze genomic interpretation. *Nature methods*, 14(1):61–64, 2017.

[143] Wei Li, Hong-Lian Li, Jian-Zhi Wang, Rong Liu, and Xiaochuan Wang. Abnormal protein post-translational modifications induces aggregation and abnormal deposition of protein, mediating neurodegenerative diseases. *Cell & Bioscience*, 14(1):1–14, 2024.

[144] Arthur Liberzon, Aravind Subramanian, Reid Pinchback, Helga Thorvaldsdóttir, Pablo Tamayo, and Jill P Mesirov. Molecular signatures database (msigdb) 3.0. *Bioinformatics*, 27(12):1739–1740, 2011.

[145] Pyoung Suk Lim, In Hee Kim, Seong Hun Kim, Seung Ok Lee, and Sang Wook Kim. A case of severe acute hepatitis a complicated with pure red cell aplasia. *The Korean Journal of Gastroenterology*, 60(3):177–181, 2012.

[146] Cui-Xiang Lin, Hong-Dong Li, Chao Deng, Yuanfang Guan, and Jianxin Wang. Tissuenexus: a database of human tissue functional gene networks built with a large compendium of curated rna-seq data. *Nucleic acids research*, 50(D1):D710–D718, 2022.

[147] Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, et al. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130, 2023.

[148] Li Liu, Qian-Zhong Li, Wen Jin, Hao Lv, and Hao Lin. Revealing gene function and transcription relationship by reconstructing gene-level chromatin interaction. *Computational and structural biotechnology journal*, 17:195–205, 2019.

[149] Renming Liu, Semih Cantürk, Frederik Wenkel, Sarah McGuire, Xinyi Wang, Anna Little, Leslie O'Bray, Michael Perlmutter, Bastian Rieck, Matthew Hirn, et al. Taxonomy of benchmarks in graph representation learning. In *Learning on Graphs Conference*, pages 6–1. PMLR, 2022.

[150] Renming Liu, Matthew Hirn, and Arjun Krishnan. Accurately modeling biased random walks on weighted networks using node2vec+. *Bioinformatics*, 39(1):btad047, 2023.

[151] Renming Liu and Arjun Krishnan. Pecanpy: a fast, efficient and parallelized python implementation of node2vec. *Bioinformatics*, 37(19):3377–3379, 2021.

[152] Renming Liu and Arjun Krishnan. Open biomedical network benchmark: A python toolkit for benchmarking datasets with biomedical networks. In *Machine Learning in Computational Biology*, pages 23–59. PMLR, 2024.

[153] Renming Liu, Christopher A Mancuso, Anna Yannakopoulos, Kayla A Johnson, and Arjun

Krishnan. Supervised learning is an accurate method for network-based gene classification. *Bioinformatics*, 36(11):3457–3465, 2020.

[154] Renming Liu, Christopher A Mancuso, Anna Yannakopoulos, Kayla A Johnson, and Arjun Krishnan. Supervised learning is an accurate method for network-based gene classification. *Bioinformatics*, 36(11):3457–3465, 2020.

[155] Renming Liu, Hao Yuan, Kayla A Johnson, and Arjun Krishnan. Cone: Context-specific network embedding via contextualized graph attention. *bioRxiv*, pages 2023–10, 2023.

[156] Sebastian Lobentanzer, Patrick Aloy, Jan Baumbach, Balazs Bohar, Vincent J Carey, Pornpimol Charoentong, Katharina Danhauser, Tunca Doğan, Johann Dreo, Ian Dunham, et al. Democratizing knowledge representation with biocypher. *Nature Biotechnology*, pages 1–4, 2023.

[157] John Lonsdale, Jeffrey Thomas, Mike Salvatore, Rebecca Phillips, Edmund Lo, Saboor Shad, Richard Hasz, Gary Walters, Fernando Garcia, Nancy Young, et al. The genotype-tissue expression (gtex) project. *Nature genetics*, 45(6):580–585, 2013.

[158] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018.

[159] Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. Is heterophily a real nightmare for graph neural networks to do node classification? *arXiv preprint arXiv:2109.05641*, 2021.

[160] Katja Luck, Dae-Kyum Kim, Luke Lambourne, Kerstin Spirohn, Bridget E Begg, Wenting Bian, Ruth Brignall, Tiziana Cafarelli, Francisco J Campos-Laborie, Benoit Charloteaux, et al. A reference map of the human binary protein interactome. *Nature*, 580(7803):402–408, 2020.

[161] Ping Luo, Yuanyuan Li, Li-Ping Tian, and Fang-Xiang Wu. Enhancing the prediction of disease–gene associations with multimodal deep learning. *Bioinformatics*, 35(19):3735–3742, 2019.

[162] Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. Is homophily a necessity for graph neural networks? In *International Conference on Learning Representations*, 2021.

[163] Donna Maglott, Jim Ostell, Kim D Pruitt, and Tatiana Tatusova. Entrez gene: gene-centered information at ncbi. *Nucleic acids research*, 33(suppl_1):D54–D58, 2005.

[164] Christopher A Mancuso, Patrick S Bills, Douglas Krum, Jacob Newsted, Renming Liu, and Arjun Krishnan. Geneplexus: a web-server for gene discovery using network-based machine learning. *Nucleic Acids Research*, 50(W1):W358–W366, 2022.

[165] Christopher A Mancuso, Kayla A Johnson, Renming Liu, and Arjun Krishnan. Joint representation of molecular networks from multiple species improves gene classification. *PLOS Computational Biology*, 20(1):e1011773, 2024.

[166] Christopher A Mancuso, Renming Liu, and Arjun Krishnan. Pygeneplexus: A python package for gene discovery using network-based machine learning. *bioRxiv*, 2022.

[167] Christopher A Mancuso, Renming Liu, and Arjun Krishnan. Pygeneplexus: a python package for gene discovery using network-based machine learning. *Bioinformatics*, 39(2):btad064, 2023.

[168] Daniel Marbach, David Lamparter, Gerald Quon, Manolis Kellis, Zoltán Kutalik, and Sven Bergmann. Tissue-specific regulatory circuits reveal variable modular perturbations across complex diseases. *Nature methods*, 13(4):366–370, 2016.

[169] Stéphane Martin, Kevin A Wilkinson, Atsushi Nishimune, and Jeremy M Henley. Emerging extranuclear roles of protein sumoylation in neuronal function and dysfunction. *Nature Reviews Neuroscience*, 8(12):948–959, 2007.

[170] John S Mattick, Paulo P Amaral, Piero Carninci, Susan Carpenter, Howard Y Chang, Ling-Ling Chen, Runsheng Chen, Caroline Dean, Marcel E Dinger, Katherine A Fitzgerald, et al. Long non-coding rnas: definitions, functions, challenges and recommendations. *Nature reviews Molecular cell biology*, 24(6):430–447, 2023.

[171] Patrick McGillivray, Declan Clarke, William Meyerson, Jing Zhang, Donghoon Lee, Mengting Gu, Sushant Kumar, Holly Zhou, and Mark Gerstein. Network analysis as a grand unifier in biomedical data science. *Annual Review of Biomedical Data Science*, 1:153–180, 2018.

[172] Jörg Menche, Amitabh Sharma, Maksim Kitsak, Susan Dina Ghiassian, Marc Vidal, Joseph Loscalzo, and Albert-László Barabási. Uncovering disease-disease relationships through the incomplete interactome. *Science*, 347(6224):1257601, 2015.

[173] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *ArXiv preprint*, 2013.

[174] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

[175] Shahin Mohammadi, Jose Davila-Velderrain, and Manolis Kellis. Reconstruction of cell-type-specific interactomes at single-cell resolution. *Cell systems*, 9(6):559–568, 2019.

[176] Michael Moret, Irene Pachon Angona, Leandro Cotos, Shen Yan, Kenneth Atz, Cyrill Brunner, Martin Baumgartner, Francesca Grisoni, and Gisbert Schneider. Leveraging molecular structure and bioactivity with chemical language models for de novo drug design.

Nature Communications, 14(1):114, 2023.

[177] Germán Morís. Inflammatory bowel disease: an increased risk factor for neurologic complications. *World Journal of Gastroenterology: WJG*, 20(5):1228, 2014.

[178] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, pages 4602–4609, 2019.

[179] Deisy Morselli Gysi and Albert-László Barabási. Noncoding rnas improve the predictive power of network medicine. *Proceedings of the National Academy of Sciences*, 120(45):e2301342120, 2023.

[180] Sara Mostafavi, Debajyoti Ray, David Warde-Farley, Chris Grouios, and Quaid Morris. Genemania: a real-time multiple association network integration algorithm for predicting gene function. *Genome biology*, 9:1–15, 2008.

[181] Giulia Muzio, Leslie O'Bray, and Karsten Borgwardt. Biological network analysis with deep learning. *Briefings in bioinformatics*, 22(2):1515–1530, 2021.

[182] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *arXiv preprint arXiv:1606.08928*, 2016.

[183] Walter Nelson, Marinka Zitnik, Bo Wang, Jure Leskovec, Anna Goldenberg, and Roded Sharan. To embed or not: network embedding as a paradigm in computational biology. *Frontiers in genetics*, 10:381, 2019.

[184] Kinga Nemeth, Recep Bayraktar, Manuela Ferracin, and George A Calin. Non-coding rnas in disease: from mechanisms to therapeutics. *Nature Reviews Genetics*, pages 1–22, 2023.

[185] Eric Nguyen, Michael Poli, Matthew G Durrant, Armin W Thomas, Brian Kang, Jeremy Sullivan, Madelena Y Ng, Ashley Lewis, Aman Patel, Aaron Lou, et al. Sequence modeling and design from molecular to genome scale with evo. *bioRxiv*, pages 2024–02, 2024.

[186] Sergey Nurk, Sergey Koren, Arang Rhie, Mikko Rautiainen, Andrey V Bzikadze, Alla Mikheenko, Mitchell R Vollger, Nicolas Altemose, Lev Uralsky, Ariel Gershman, et al. The complete sequence of a human genome. *Science*, 376(6588):44–53, 2022.

[187] David Ochoa, Andrew Hercules, Miguel Carmona, Daniel Suveges, Asier Gonzalez-Uriarte, Cinzia Malangone, Alfredo Miranda, Luca Fumis, Denise Carvalho-Silva, Michaela Spitzer, et al. Open targets platform: supporting systematic drug–target identification and prioritisation. *Nucleic acids research*, 49(D1):D1302–D1310, 2021.

[188] Stephen Oliver. Guilt-by-association goes global. *Nature*, 403(6770):601–602, 2000.

[189] Rose Oughtred, Jennifer Rust, Christie Chang, Bobby-Joe Breitkreutz, Chris Stark, Andrew Willems, Lorrie Boucher, Genie Leung, Nadine Kolas, Frederick Zhang, et al. The biogrid database: A comprehensive biomedical resource of curated protein, genetic, and chemical interactions. *Protein Science*, 30(1):187–200, 2021.

[190] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[191] Oana Palasca, Alberto Santos, Christian Stolte, Jan Gorodkin, and Lars Juhl Jensen. Tissues 2.0: an integrative web resource on mammalian tissue expression. *Database*, 2018:bay003, 2018.

[192] Christopher Y Park, Aaron K Wong, Casey S Greene, Jessica Rowland, Yuanfang Guan, Lars A Bongo, Rebecca D Burdine, and Olga G Troyanskaya. Functional knowledge transfer for high-accuracy prediction of under-studied biological processes. *PLoS computational biology*, 9(3):e1002957, 2013.

[193] Elizabeth Park, Jan Griffin, and Joan M Bathon. Myocardial dysfunction and heart failure in rheumatoid arthritis. *Arthritis & Rheumatology*, 74(2):184–199, 2022.

[194] Liz Parrish. Psoriasis: symptoms, treatments and its impact on quality of life. *British Journal of Community Nursing*, 17(11):524–528, 2012.

[195] 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, 2019.

[196] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[197] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[198] Lourdes Peña-Castillo, Murat Tasan, Chad L Myers, Hyunju Lee, Trupti Joshi, Chao Zhang, Yuanfang Guan, Michele Leone, Andrea Pagnani, Wan Kyu Kim, et al. A critical assessment of mus musculus gene function prediction using integrated genomic evidence. *Genome biology*, 9:1–19, 2008.

[199] J. Peng, J. Guan, and X. Shang. Predicting parkinson's disease genes based on node2vec and

autoencoder. *Fontiers in Genetics*, 2019.

[200] Jiajie Peng, Jiaojiao Guan, and Xuequn Shang. Predicting parkinson's disease genes based on node2vec and autoencoder. *Frontiers in genetics*, 10:226, 2019.

[201] Livia Perfetto, Leonardo Briganti, Alberto Calderone, Andrea Cerquone Perpetuini, Marta Iannuccelli, Francesca Langone, Luana Licata, Milica Marinkovic, Anna Mattioni, Theodora Pavlidou, et al. Signor: a database of causal relationships between biological entities. *Nucleic acids research*, 44(D1):D548–D554, 2016.

[202] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, 2014.

[203] Bryan Perozzi, Vivek Kulkarni, Haochen Chen, and Steven Skiena. Don't walk, skip! online learning of multi-scale network embeddings. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, pages 258–265, 2017.

[204] Emma Persson, Miguel Castresana-Aguirre, Davide Buzzao, Dimitri Guala, and Erik LL Sonnhammer. Funcoup 5: functional association networks in all domains of life, supporting directed links and tissue-specificity. *Journal of Molecular Biology*, 433(11):166835, 2021.

[205] Sergio Picart-Armada, Steven J Barrett, David R Willé, Alexandre Perera-Lluna, Alex Gutteridge, and Benoit H Dessailly. Benchmarking network propagation methods for disease gene identification. *PLoS computational biology*, 15(9):e1007276, 2019.

[206] Janet Piñero, Àlex Bravo, Núria Queralt-Rosinach, Alba Gutiérrez-Sacristán, Jordi Deu-Pons, Emilio Centeno, Javier García-García, Ferran Sanz, and Laura I Furlong. Disgenet: a comprehensive platform integrating information on human disease-associated genes and variants. *Nucleic acids research*, page gkw943, 2016.

[207] Rosario M Piro and Ferdinando Di Cunto. Computational approaches to disease-gene prediction: rationale, classification and successes. *The FEBS journal*, 279(5):678–696, 2012.

[208] Rosario Michael Piro and Annalisa Marsico. Network-based methods and other approaches for predicting lncrna functions and disease associations. *Computational Biology of Non-Coding RNA: Methods and Protocols*, pages 301–321, 2019.

[209] Elif Piskin, Danila Cianciosi, Sukru Gulec, Merve Tomas, and Esra Capanoglu. Iron absorption: factors, limitations, and improvement methods. *ACS omega*, 7(24):20441–20456, 2022.

[210] Dexter Pratt, Jing Chen, David Welker, Ricardo Rivas, Rudolf Pillich, Vladimir Rynkov, Keiichiro Ono, Carol Miello, Lyndon Hicks, Sandor Szalma, et al. Ndex, the network data

exchange. *Cell systems*, 1(4):302–305, 2015.

[211] Yue Qin, Edward L Huttlin, Casper F Winsnes, Maya L Gosztyla, Ludivine Wacheul, Marcus R Kelly, Steven M Blue, Fan Zheng, Michael Chen, Leah V Schaffer, et al. A multi-scale map of cell structure fusing protein images and interactions. *Nature*, 600(7889):536–542, 2021.

[212] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *Proceedings of the eleventh ACM international conference on web search and data mining*, pages 459–467, 2018.

[213] Predrag Radivojac, Wyatt T Clark, Tal Ronnen Oron, Alexandra M Schnoes, Tobias Wittkop, Artem Sokolov, Kiley Graim, Christopher Funk, Karin Verspoor, Asa Ben-Hur, et al. A large-scale evaluation of computational protein function prediction. *Nature methods*, 10(3):221–227, 2013.

[214] Ricardo N Ramirez, Nicole C El-Ali, Mikayla Anne Mager, Dana Wyman, Ana Conesa, and Ali Mortazavi. Dynamic gene regulatory networks of human myeloid differentiation. *Cell systems*, 4(4):416–429, 2017.

[215] Tavis J Reed, Matthew D Tyl, Alicja Tadych, Olga G Troyanskaya, and Ileana M Cristea. Tapioca: a platform for predicting de novo protein–protein interactions in dynamic contexts. *Nature Methods*, pages 1–13, 2024.

[216] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

[217] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 385–394, 2017.

[218] Daniel Rivas-Barragan, Sarah Mubeen, Francesc Guim Bernat, Martin Hofmann-Apitius, and Daniel Domingo-Fernández. Drug2ways: Reasoning over causal paths in biological networks for drug discovery. *PLoS computational biology*, 16(12):e1008464, 2020.

[219] Giulia Roda, Alessandro Sartini, Elisabetta Zambon, Andrea Calafiore, Margherita Marocchi, Alessandra Caponi, Andrea Belluzzi, and Enrico Roda. Intestinal epithelial cells in inflammatory bowel diseases. *World journal of gastroenterology: WJG*, 16(34):4264, 2010.

[220] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[221] Yu Rong, Yatao Bian, Tingyang Xu, Weiyang Xie, Ying Wei, Wenbing Huang, and Junzhou

Huang. Self-supervised graph transformer on large-scale molecular data. *Advances in neural information processing systems*, 33:12559–12571, 2020.

[222] Yusuf Roohani, Kexin Huang, and Jure Leskovec. Predicting transcriptional outcomes of novel multigene perturbations with gears. *Nature Biotechnology*, pages 1–9, 2023.

[223] Jerret Ross, Brian Belgodere, Vijil Chenthamarakshan, Inkit Padhi, Youssef Mroueh, and Payel Das. Large-scale chemical language representations capture molecular structure and properties. *Nature Machine Intelligence*, 4(12):1256–1264, 2022.

[224] Juliana S Bernardes. A review of protein function prediction under machine learning perspective. *Recent patents on biotechnology*, 7(2):122–141, 2013.

[225] Sepideh Sadegh, James Skelton, Elisa Anastasi, Judith Bernett, David B Blumenthal, Gihanna Galindez, Marisol Salgado-Albarrán, Olga Lazareva, Keith Flanagan, Simon Cockell, et al. Network medicine for disease module identification and drug repurposing with the nedrex platform. *Nature Communications*, 12(1):6848, 2021.

[226] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):e0118432, 2015.

[227] Akira Sato, Fumiaki Sano, Toshiya Ishii, Kayo Adachi, Ryujirou Negishi, Nobuyuki Matsumoto, and Chiaki Okuse. Pure red cell aplasia associated with autoimmune hepatitis successfully treated with cyclosporine a. *Clinical Journal of Gastroenterology*, 7:74–78, 2014.

[228] Martin H Schaefer, Tiago JS Lopes, Nancy Mah, Jason E Shoemaker, Yukiko Matsuoka, Jean-Fred Fontaine, Caroline Louis-Jeune, Amie J Eisfeld, Gabriele Neumann, Carol Perez-Iratxeta, et al. Adding protein context to the human protein-protein interaction network to reveal meaningful interactions. *PLoS computational biology*, 9(1):e1002860, 2013.

[229] Lynn Marie Schriml, Cesar Arze, Suvarna Nadendla, Yu-Wei Wayne Chang, Mark Mazaitis, Victor Felix, Gang Feng, and Warren Alden Kibbe. Disease ontology: a backbone for disease semantic integration. *Nucleic acids research*, 40(D1):D940–D946, 2012.

[230] Benno Schwikowski, Peter Uetz, and Stanley Fields. A network of protein–protein interactions in yeast. *Nature biotechnology*, 18(12):1257–1261, 2000.

[231] Roded Sharan, Igor Ulitsky, and Ron Shamir. Network-based prediction of protein function. *Molecular systems biology*, 3(1):88, 2007.

[232] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *ArXiv preprint*, 2018.

[233] Harinder Singh, Kay L Medina, and Jagan MR Pongubala. Contingent gene regulatory networks and b cell fate specification. *Proceedings of the National Academy of Sciences*, 102(14):4949–4953, 2005.

[234] Rohit Singh, Samuel Sledzieski, Bryan Bryson, Lenore Cowen, and Bonnie Berger. Contrastive learning in protein language space predicts interactions between drugs and protein targets. *Proceedings of the National Academy of Sciences*, 120(24):e2220778120, 2023.

[235] Barry Smith, Michael Ashburner, Cornelius Rosse, Jonathan Bard, William Bug, Werner Ceusters, Louis J Goldberg, Karen Eilbeck, Amelia Ireland, Christopher J Mungall, et al. The obo foundry: coordinated evolution of ontologies to support biomedical data integration. *Nature biotechnology*, 25(11):1251–1255, 2007.

[236] Cynthia L Smith and Janan T Eppig. The mammalian phenotype ontology: enabling robust annotation and comparative analysis. *Wiley Interdisciplinary Reviews: Systems Biology and Medicine*, 1(3):390–399, 2009.

[237] Cynthia L Smith, Carroll-Ann W Goldsmith, and Janan T Eppig. The mammalian phenotype ontology as a tool for annotating, analyzing and comparing phenotypic information. *Genome biology*, 6:1–9, 2005.

[238] Helen R Sofaer, Jennifer A Hoeting, and Catherine S Jarnevich. The area under the precision-recall curve as a performance metric for rare binary events. *Methods in Ecology and Evolution*, 10(4):565–577, 2019.

[239] Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

[240] Chris Stark, Bobby-Joe Breitkreutz, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, and Mike Tyers. Biogrid: a general repository for interaction datasets. *Nucleic acids research*, 34(suppl_1):D535–D539, 2006.

[241] Joshua M Stuart, Eran Segal, Daphne Koller, and Stuart K Kim. A gene-coexpression network for global discovery of conserved genetic modules. *science*, 302(5643):249–255, 2003.

[242] Aravind Subramanian, Pablo Tamayo, Vamsi K Mootha, Sayan Mukherjee, Benjamin L Ebert, Michael A Gillette, Amanda Paulovich, Scott L Pomeroy, Todd R Golub, Eric S Lander, et al. Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences*, 102(43):15545–15550, 2005.

[243] Marissa Sumathipala, Enrico Maiorino, Scott T Weiss, and Amitabh Sharma. Network diffusion approach to predict lncrna disease associations using multi-type biological networks: Lion. *Frontiers in physiology*, 10:446144, 2019.

[244] Kyle Swanson, Gary Liu, Denise B Catacutan, Autumn Arnold, James Zou, and Jonathan M Stokes. Generative ai for designing and validating easily synthesizable and structurally novel antibiotics. *Nature Machine Intelligence*, 6(3):338–353, 2024.

[245] Damian Szklarczyk, Andrea Franceschini, Stefan Wyder, Kristoffer Forslund, Davide Heller, Jaime Huerta-Cepas, Milan Simonovic, Alexander Roth, Alberto Santos, Kalliopi P Tsafou, et al. String v10: protein–protein interaction networks, integrated over the tree of life. *Nucleic acids research*, 43(D1):D447–D452, 2015.

[246] Damian Szklarczyk, Annika L Gable, David Lyon, Alexander Junge, Stefan Wyder, Jaime Huerta-Cepas, Milan Simonovic, Nadezhda T Doncheva, John H Morris, Peer Bork, et al. String v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. *Nucleic acids research*, 47(D1):D607–D613, 2019.

[247] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, 2015.

[248] Cole Trapnell. Defining cell types and states with single-cell genomics. *Genome research*, 25(10):1491–1498, 2015.

[249] Dénes Türei, Tamás Korcsmáros, and Julio Saez-Rodriguez. Omnipath: guidelines and gateway for literature-curated signaling pathway resources. *Nature methods*, 13(12):966–967, 2016.

[250] Mathias Uhlén, Linn Fagerberg, Björn M Hallström, Cecilia Lindskog, Per Oksvold, Adil Mardinoglu, Åsa Sivertsson, Caroline Kampf, Evelina Sjöstedt, Anna Asplund, et al. Tissue-based map of the human proteome. *Science*, 347(6220):1260419, 2015.

[251] Giorgio Valentini, Elena Casiraghi, Luca Cappelletti, Tommaso Fontana, Justin Reese, and Peter Robinson. Het-node2vec: second order random walk sampling for heterogeneous multigraphs embedding. *arXiv preprint arXiv:2101.01425*, 2021.

[252] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in science & engineering*, 13(2):22–30, 2011.

[253] Oron Vanunu, Oded Magger, Eytan Ruppin, Tomer Shlomi, and Roded Sharan. Associating genes and protein complexes with disease via network propagation. *PLoS computational biology*, 6(1):e1000641, 2010.

[254] Nicole A Vasilevsky, Nicolas A Matentzoglu, Sabrina Toro, Joe E Flack, Harshad Hegde, Deepak R Unni, Gioconda Alyea, Joanna S Amberger, Larry Babb, James P Balhoff, et al. Mondo: Unifying diseases for the world, by the world. *medRxiv*, 2022.

[255] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[256] Alexei Vazquez, Alessandro Flammini, Amos Maritan, and Alessandro Vespignani. Global protein function prediction from protein-protein interaction networks. *Nature biotechnology*, 21(6):697–700, 2003.

[257] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

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

[259] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep graph infomax. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[260] Daniel V Veres, Dávid M Gyurkó, Benedek Thaler, Kristof Z Szalay, Dávid Fazekas, Tamás Korcsmáros, and Peter Csermely. Comppi: a cellular compartment-specific database for protein–protein interaction network analysis. *Nucleic acids research*, 43(D1):D485–D493, 2015.

[261] Marc Vidal, Michael E Cusick, and Albert-László Barabási. Interactome networks and human disease. *Cell*, 144(6):986–998, 2011.

[262] Marc Vidal, Michael E Cusick, and Albert-László Barabási. Interactome networks and human disease. *Cell*, 144(6):986–998, 2011.

[263] Hao Wang, Jiaxin Yang, and Jianrong Wang. Leverage large-scale biological networks to decipher the genetic basis of human diseases using machine learning. *Artificial Neural Networks*, pages 229–248, 2021.

[264] Hao Wang, Jiaxin Yang, Yu Zhang, and Jianrong Wang. Discover novel disease-associated genes based on regulatory networks of long-range chromatin interactions. *Methods*, 189:22–33, 2021.

[265] Minjie Yu Wang. Deep graph library: Towards efficient and scalable deep learning on graphs. In *ICLR workshop on representation learning on graphs and manifolds*, 2019.

[266] Nian Wang, Min Zeng, Yiming Li, Fang-xiang Wu, and Min Li. Essential protein prediction based on node2vec and xgboost. *Journal of Computational Biology*, 28(7):687–700, 2021.

[267] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[268] Xiujuan Wang, Natali Gulbahce, and Haiyuan Yu. Network-based methods for human disease gene prediction. *Briefings in functional genomics*, 10(5):280–293, 2011.

[269] Yangkun Wang, Jiarui Jin, Weinan Zhang, Yong Yu, Zheng Zhang, and David Wipf. Bag of tricks for node classification with graph neural networks. *arXiv preprint arXiv:2103.13355*, 2021.

[270] YueQun Wang, LiYan Dong, XiaoQuan Jiang, XinTao Ma, YongLi Li, and Hao Zhang. Kg2vec: A node2vec-based vectorization model for knowledge graph. *Plos one*, 16(3):e0248552, 2021.

[271] Zichen Wang, Caroline D Monteiro, Kathleen M Jagodnik, Nicolas F Fernandez, Gregory W Gundersen, Andrew D Rouillard, Sherry L Jenkins, Axel S Feldmann, Kevin S Hu, Michael G McDermott, et al. Extraction and analysis of signatures from the gene expression omnibus by the crowd. *Nature communications*, 7(1):1–11, 2016.

[272] David Warde-Farley, Sylva L Donaldson, Ovi Comes, Khalid Zuberi, Rashad Badrawi, Pauline Chao, Max Franz, Chris Grouios, Farzana Kazi, Christian Tannus Lopes, et al. The genemania prediction server: biological network integration for gene prioritization and predicting gene function. *Nucleic acids research*, 38(suppl_2):W214–W220, 2010.

[273] James C Whisstock and Arthur M Lesk. Prediction of protein function from protein sequence and structure. *Quarterly reviews of biophysics*, 36(3):307–340, 2003.

[274] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

[275] Adelaide Woicik, Mingxin Zhang, Hanwen Xu, Sara Mostafavi, and Sheng Wang. Gemini: memory-efficient integration of hundreds of gene networks with high-order pooling. *Bioinformatics*, 39:i504 – i512, 2023.

[276] Chunlei Wu, Ian MacLeod, and Andrew I Su. Biogps and mygene. info: organizing online, gene-centric information. *Nucleic acids research*, 41(D1):D561–D565, 2013.

[277] Chunlei Wu, Adam Mark, and Andrew I Su. Mygene. info: gene annotation query as a service. *bioRxiv*, page 009332, 2014.

[278] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

[279] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[280] Yaochen Xie, Zhao Xu, Jingtun Zhang, Zhengyang Wang, and Shuiwang Ji. Self-supervised learning of graph neural networks: A unified review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[281] Jiwen Xin, Adam Mark, Cyrus Afrasiabi, Ginger Tsueng, Moritz Juchler, Nikhil Gopal, Gregory S Stupp, Timothy E Putman, Benjamin J Ainscough, Obi L Griffith, et al. High-performance web services for querying gene and variant annotation. *Genome biology*, 17:1–7, 2016.

[282] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[283] Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019.

[284] Keiichi Yamanaka, Osamu Yamamoto, and Tetsuya Honda. Pathophysiology of psoriasis: A review. *The Journal of dermatology*, 48(6):722–731, 2021.

[285] Xiao-Ying Yan, Shao-Wu Zhang, and Song-Yao Zhang. Prediction of drug–target interaction by label propagation with mutual interaction information derived from heterogeneous network. *Molecular BioSystems*, 12(2):520–531, 2016.

[286] Kuo Yang, Ruyu Wang, Guangming Liu, Zixin Shu, Ning Wang, Runshun Zhang, Jian Yu, Jianxin Chen, Xiaodong Li, and Xuezhong Zhou. Hergepred: heterogeneous network embedding representation for disease gene prediction. *IEEE journal of biomedical and health informatics*, 23(4):1805–1815, 2018.

[287] Victoria Yao, Rachel Kaletsky, William Keyes, Danielle E Mor, Aaron K Wong, Salman Sohrabi, Coleen T Murphy, and Olga G Troyanskaya. An integrative tissue-network approach to identify and test human disease genes. *Nature biotechnology*, 36(11):1091–1099, 2018.

[288] Osman N Yogurtcu and Margaret E Johnson. Cytosolic proteins can exploit membrane localization to trigger functional assembly. *PLoS computational biology*, 14(3):e1006031, 2018.

[289] Haiyuan Yu, Philip M Kim, Emmett Sprecher, Valery Trifonov, and Mark Gerstein. The importance of bottlenecks in protein networks: correlation with gene essentiality and expression dynamics. *PLoS computational biology*, 3(4):e59, 2007.

[290] Yang Yu, Shengtao Zhu, Peng Li, Li Min, and Shutian Zhang. Helicobacter pylori infection

and inflammatory bowel disease: a crosstalk between upper and lower digestive tract. *Cell death & disease*, 9(10):961, 2018.

[291] Xiang Yue, Zhen Wang, Jingong Huang, Srinivasan Parthasarathy, Soheil Moosavinasab, Yungui Huang, Simon M Lin, Wen Zhang, Ping Zhang, and Huan Sun. Graph embedding on biomedical networks: methods, applications and evaluations. *Bioinformatics*, 36(4):1241–1251, 2020.

[292] Min Zeng, Min Li, Zhihui Fei, Fang-Xiang Wu, Yaohang Li, Yi Pan, and Jianxin Wang. A deep learning framework for identifying essential proteins by integrating multiple types of biological information. *IEEE/ACM transactions on computational biology and bioinformatics*, 18(1):296–305, 2019.

[293] Dengyong Zhou, Olivier Bousquet, Thomas Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. *Advances in neural information processing systems*, 16, 2003.

[294] Dongyan Zhou, Songjie Niu, and Shimin Chen. Efficient graph computation for node2vec. *arXiv preprint arXiv:1805.00280*, 2018.

[295] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[296] Kaixiong Zhou, Xiao Huang, Yuening Li, Daochen Zha, Rui Chen, and Xia Hu. Towards deeper graph neural networks with differentiable group normalization. *Advances in neural information processing systems*, 33:4917–4928, 2020.

[297] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. *Bioinformatics*, 33(14):i190–i198, 2017.

[298] Marinka Zitnik, Michelle M Li, Aydin Wells, Kimberly Glass, Deisy Morselli Gysi, Arjun Krishnan, TM Murali, Predrag Radivojac, Sushmita Roy, Anaïs Baudot, et al. Current and future directions in network biology. *arXiv preprint arXiv:2309.08478*, 2023.

APPENDIX A

GENEPLEXUS

A.1 Additional information

A.1.1 Networks

The networks used in this study are BioGRID, STRING-EXP, InBioMap, GIANT-TN, and STRING. Detailed information about the network properties and sources can be seen in Table A.1, with the network construction method and interaction type information coming from [100]. BioGRID (version 3.4.136) is a low-throughput network that includes both genetic interactions, as well as physical protein-protein interactions [240]. InBioMap (version 2016_09_12) is a high-throughput, scored network that contains physical protein-protein interactions as well as pathway database annotations incorporated as edges [142]. We used the "final-scores" as the edge weights. STRING (version 10.0) is a high-throughput, scored network that aggregates information from many data sources [245]. We used two different STRING networks. First, we used the "combined" network that directly includes database annotations, text-mining, ortholog information, co-expression, and physical protein interactions (referred to as STRING in this study). We also used a subset of edges in STRING that only contains experimental evidences, thus restricting the network to one constructed just from physical protein interactions in humans (referred to as STRING-EXP in this study). For both networks, we used the corresponding relationship scores as edge weights, after normalizing them to lie between 0 and 1. The GIANT-TN (version 1.0) network is the tissue-naïve network from GIANT [80], referred to as the "Global" network on the website, and is constructed from both low- and high-throughput data, and includes information from co-expression, non-protein sources, regulatory data, and physical protein-protein interactions. The GIANT-TN network is a fully connected, scored network. To add sparsity to the GIANT-TN network, we removed all edges with scores below 0.01 (equal to the prior the Bayesian model used to construct the network). It is worth noting here that the purpose of this study is not to compare networks against each other, but rather to determine the performance of SL methods vs LP methods on various types of networks.

Table A.1: **Gene interaction networks information.** LT : low-throughput, HT : high-throughput, GE : genetic, PH : physical, DA : database annotations, CE : co-expression, NP : non-protein, RE : regulatory, CC : co-citation, OR : orthologous.

Network	Num. nodes	Num. edges	Density	Info	Weighted	Interaction types
BioGRID	20,558	238,474	1.13E-3	LT	✗	GE, PH
STRING-EXP	14, 089	141,629	7.08E-4	HT	✓	PH
InBioMap	17,399	644,862	1.58E-3	HT	✓	PE, DA
GIANT-TN	25,689	38,904,929	1.92E-3	LT, HT	✓	CE, NP, PH, RE
STRING	17,352	3,540,737	7.20E-3	HT	✓	CC, CE, OR, DA, PH

A.1.2 Model selection and hyperparameter tuning

The restart hyperparameter used in generating an influence matrix was determined by doing a grid search over values between 0.55 and 0.95 in 0.1 steps for all networks and gene set collections, optimizing for auPRC using label propagation (Figure A.1). In general, there was not a strong dependence on . It can be seen in Figure A.1 that a higher restart probability resulted in marginally better performance for the larger networks (STRING and GLOBAL), whereas as a smaller restart probability led to nominally better performance for the smaller networks such as BioGRID. In this study, we used = 0.85 for every gene set collection–network combination, as = 0.85 offered good performance and had low variance. This was used for both LP-I and SL-I. We stress that the tuning of was never done for SL-I, and thus, our finding that SL methods generally outperform LP methods is not biased by this parameter tuning.

Model selection of the supervised learning classifier was done by comparing four popular classifiers that are implemented in Python package `scikit-learn` [196]. To determine the best supervised learning classifier, we compared their performance over every gene set collection–network combination using their default hyperparameters in version 0.19 of `scikit-learn` (Figure A.2). Logistic regression with ℓ_2 regularization is marginally better than linear support vector machines and both these classifiers outperform random forest and logistic regression with ℓ_1 regularization.

For the model selection of the embedding technique, we chose node2vec [83] because its competitive performance and ease of use [78]. The following hyperparameters were tuned based on the aggregated performance across all gene set collections–network combinations: p - the breadth first search parameter, q - the depth first search parameter, d - embedding size, r - number of walks

per node, l - walk length, and k - context window size. Since p and q are coupled, we performed a grid search for these two parameters leaving all others constant. Each of the other hyperparameters was tuned by leaving the rest at their default values as described in the original node2vec publication. This procedure resulted in the following hyperparameter values – $p = 0.1$, $q = 0.1$, $d = 512$, $k = 8$, $l = 120$, and $r = 10$ – which were used for every gene set collection–network combination.

A.1.3 Gene set collections

The gene set collections used in this work are from the Gene Ontology (from version 2 of `MyGene.info` API with data retrieved on 2018-05-18, GOBPtmp, GOBP) [49, 14, 276, 281], Kyoto Encyclopedia of Genes and Genomes (from version 6.1 of MSigDB, KEGGBP) [119], DisGeNet (version 5.0, DisGeNet, BeFree) [206], GWAS from a community challenge at https://www.synapse.org/#!Synapse:syn11944948 [47], and the Mouse Gene Informatics database (data retrieved on 2018-10-01, MGI) [37].

A.1.3.1 Pre-processing gene sets based on specificity, redundancy, and multi-functionality

Each of these six gene set collections contained anywhere from about a hundred to tens of thousands of gene sets (Table A.2) that varied widely in specificity and redundancy. The first pre-processing step we did after downloading the data was to convert the original gene/protein IDs to entrez gene IDs, which was done using gene ID conversions found in `MyGene.info` [276, 281]. If the original ID mapped to more than one entrez ID, all of them were included for further analysis. Next, whenever applicable, annotations to gene sets corresponding to terms in a curated ontology were propagated along the is_a and part_of relationships to ancestor terms in the corresponding ontologies: Gene Ontology [14] for GOBP, Disease Ontology [229] for DisGeNet and BeFree, and Mammalian Phenotype Ontology [236] for MGI.

Subsequent preprocessing steps were designed to ensure that the final set of gene sets from each source are specific, largely non-overlapping, and not driven by multi-attribute genes.

Specificity To select specific biologically-meaningful gene sets in each collection, we sorted all the gene sets in a collection from the largest to smallest based on the number of annotated genes (gene set size), manually examined their descriptions, and chose a size threshold that roughly separated

Table A.2: **Gene set collection information.** The last four columns reflect the fact each gene set collection is slightly different for every network and these values are presented as either a range, a median value, or number of genes in a union across all networks used in this study.

Gene set collection	Num. full gene sets	Num. non-redundant gene sets	Num. gene sets after holdout preprocessing	Gene set sizes	Median gene set sizes	Total num. genes
GOBPtmp	754†	166	(115, 160)	(27, 452)	174	9,464
GOBP	11,574	313	(84, 96)	(20, 181)	76	5,301
KEGGBP	149	138	(63, 96)	(24, 181)	51	3,454
DisGeNet	4,030	334	(89, 104)	(21, 368)	67	4,689
BeFree	2,891	207	(49, 57)	(20, 223)	80	2,692
GWAS	169	74	(30, 37)	(24, 431)	94	2,134
MGI	10,264	492	(90, 121)	(20, 132)	41	2,716

† The GOBP temporal holdout additionally ensures at least ten genes are in the training and testing sets.

large, generic gene sets from the smaller, specific ones. This threshold was 200 for GOBP and KEGGBP, 300 for MGI, 400 for BeFree, 500 for GWAS and GOBPtmp, and 600 for DisGeNet.

Redundancy To remove redundant gene sets within a collection, first, we calculated the Jaccard index ($|A \cap B| / |A \cup B|$) and the overlap index ($|A \cap B| / \min(|A|, |B|)$) between all pairs of gene sets, where A and B represent two sets of genes annotated to the gene sets. Then, we built a graph with the gene sets as the nodes, and added edges between gene sets pairs if their Jaccard index was >0.5 and their overlap index was >0.7. The gene set graph constructed in this manner contained many connected components, each representing a set of highly overlapping gene sets. Finally, we used the following procedure to pick representative gene sets within each component:

1. calculate a score for each gene set equal to the sum of the proportions of genes in other linked gene sets that are contained within it (higher this score, more representative that gene set is)
2. create a sorted list of all the gene sets in decreasing order of this score
3. pick the first gene set in the list, remove every subsequent gene set that is connected to it in the graph, and repeat this step until the sorted list is empty.

This procedure resulted in a reasonable number of non-redundant gene sets within each collection. The same Jaccard and overlap thresholds were used for all collections except MGI. For MGI, since an overlap cutoff of 0.7 still resulted in thousands of gene sets, it was further lowered to 0.5.

Multi-attribute genes Given the set of largely non-overlapping gene sets in a collection, individual genes were removed from all gene sets if they appeared in more than 10 gene sets in that collection. This step ensures that the evaluations are not biased by multi-attribute genes that can potentially be easily predicted in a non-specific manner [73].

We also note that we did not include the cellular component (CC) or molecular function (MF) classes of the gene ontology as part of the function classification tasks because two genes that are annotated to the same CC or MF need not be related to each other functionally.

A.1.3.2 Calculating the network properties of the gene sets

To determine how the performance of a given gene set depends on the network, for each gene set we calculated three different properties:

1. For a given gene set, T, the number of genes annotated it is given by T.

2. For a given gene set, T, the edge density, D_T, is given by

$$D_T = \frac{2\sum_{(u,v)\in T} W_{uv}}{|T|(|T|-1)} \tag{A.1}$$

   where W_{uv} is the edge weight between genes u and v. The edge density is a measure of how tightly connected the gene set is within itself.

3. For a given gene set, T, the segregation, ST, is given by

$$S_T = \frac{\sum_{(u,v)\in T} W_{uv}}{\sum_{u\in T, t\in V} W_{ut}} \tag{A.2}$$

   Segregation is a measure of how isolated the gene set is from the rest of the network.

The three gene set properties are shown for all gene set collection–network combinations in Figure A.3. In general, there is little difference in the number of genes across the different prediction tasks (i.e. function, disease and trait), except for GOBPtmp, which has the largest number of genes due to the fact the gene sets need to be larger to have enough with at least 10 testing genes. Edge density and segregation are highest for the function gene sets (GOBPtmp, GOBP, KEGGBP) and lowest for the disease and trait gene sets (DisGeNet, BeFree, GWAS, MGI).

A.1.4 Evaluation data split

We used three different validation schemes to evaluate gene classification.

Temporal holdout validation Temporal holdout is the most stringent evaluation scheme for gene classification since it mimics the practical scenario of using current knowledge to predict the future. Since Gene Ontology was the only source with clear date-stamps for all its annotations, temporal holdout was applied only to the GOBP gene set collection. Since the goal of this study is to use relatively recent and widely-used molecular networks, as this would reflect how these models would be deployed in practice, we chose a temporal cutoff point of Jan 1st, 2017. Then, for each gene set collection, genes that only had an annotation to any gene set in the collection after 2017-01-01 were assigned to the testing set and the remaining genes were assigned to the training set. Since this resulted in the testing set having far fewer genes than the testing set for the other validation schemes, we made the following minor modifications to the gene set pre-processing procedure: GOBP gene set collection was first filtered to remove any gene set with fewer than ten training genes or had fewer than ten testing genes based on the temporal split and the specificity threshold (maximum number of genes annotated to a gene set) was increased from 200 to 500. Redundancy filtering and multi-attribute gene filtering were unchanged. As noted in Section 1.1, from each network resource considered in this study, we chose the most recent version of the network that was released before 2017-01-01 to ensure no data leakage. Finally, genes were removed from gene sets if they were not present in a given network, gene sets with fewer than ten training genes or fewer than ten testing genes were filtered out, and the remaining gene sets were used to perform the temporal holdout validation.

Study-bias holdout validation The goal of study-bias validation is to evaluate the scenario that is close to the real-world situation of learning from well-characterized genes to predict novel un(der)-characterized genes. Here, we defined study-bias for each gene as the number of articles in PubMed (http://www.pubmed.gov/) in which that gene was referenced in, as determined in the gene2pubmed file (downloaded on 2018-10-30) from the NCBI Gene database [35]. Using this definition, for each gene set collection–network combination, we created training-testing splits in

the following manner: Genes were removed from gene sets if they were not present in the given network. Then, among the remaining genes, a gene was assigned to the training set if it was in the top two-thirds of the list of genes sorted by their PubMed count. The remaining genes were assigned to the testing set. Finally, gene sets with fewer than ten training genes or fewer than ten testing genes were filtered out and the remaining gene sets were used to perform the study-bias holdout validation.

Five-fold cross-validation To ensure comparability, we performed 5-fold cross-validation using the same gene sets that were used in study-bias holdout, splitting each gene set randomly into five approximately equal folds (with similar proportions of positive and negative examples) and, in rotation, using one fold as the testing set and the remaining four as the training set.

A.1.5 Evaluation metrics

We present results in three metrics, the area under the precision recall curve (auPRC), the area under the receiver operator curve(auROC), and the precision of the top K ranked predictions (P@TopK). Since each gene set collection–network combination has a different number of positive examples (and, hence, different positive:negative proportions), we normalized auPRC and P@topK by the prior. Specifically, auPRC is given by:

$$\text{auPRC} = \log_2\left(\frac{\text{auPRC}_\text{S}}{\text{prior}}\right) \tag{A.3}$$

where $auPRC_S$ is the standard area under the precision-recall curve, and the prior is $P/P+N$ with P being the number of positive ground truth labels, and N being the number of negative ground truth labels. The log_2 in Equation A.3 allows for the following interpretation: the number of 2-fold increases of the measured auPRCSover what is expected given the ground truth labels (e.g., a value of 1 indicates a 2-fold increase, a value of 2 indicates a 4-fold increase). Similarly, P@topK is given by:

$$\text{P@topK} = \log_2\left(\frac{\text{TP}_\text{K}}{\text{K} \times \text{prior}}\right) \tag{A.4}$$

where K is the number of top-predictions to consider, TPK is the number true-positives of the top-K predictions, and the prior is the same as in Equation A.3. We set K to be the number of ground truth positives in the testing set. P@topK can be thought of as what is the 2-fold increase in the percent

of the top-K predictions that were predicted true over the expected value. Of note, it is possible that TPk=0 if no true positive is captured within the first K predictions. This causes P@topK to become $-\infty$. To address this issue, we set such values to be the minimum score obtained across all predictions for that given gene set collection–network combination.

Precision-based metrics – auPRC and P@topK – are more suitable than the more popular area under the receiver-operating characteristic curve (auROC) for two reasons. First, gene classification is a highly imbalanced problem with many more negative samples than positive samples, and auROC is ill-suited for imbalanced problems [226]. Second, precision can control for Type-1 error (false positives) [55]. Since the foremost reason for gene classification is to provide a list of candidate genes for further experimental study, it is more important to make sure the top predictions are as correct as possible, as opposed to ensuring that, on average, positive examples are ranked higher than negative examples. However, for completeness, we have provided auROC results in this Supplemental Material (Section 2).

A.2 Additional results

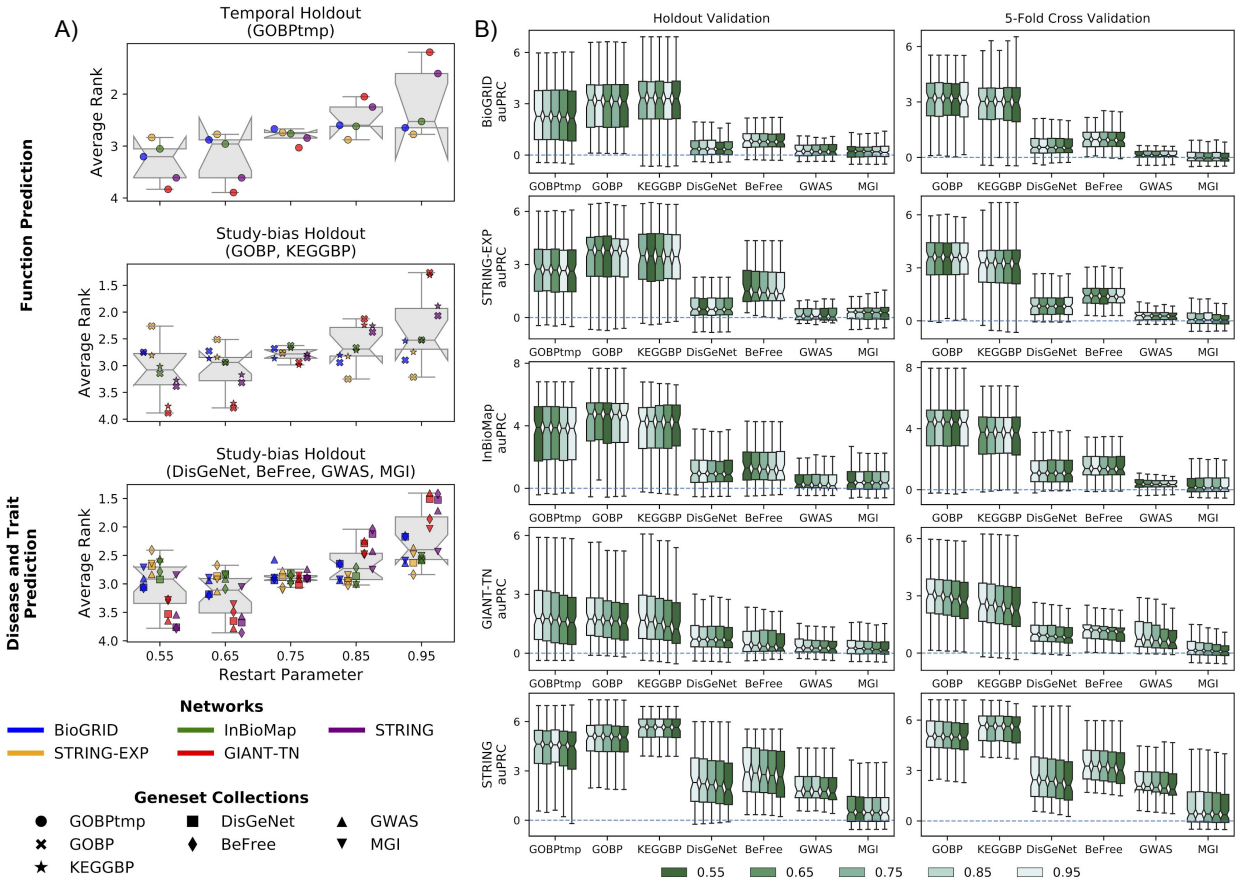


Figure A.1: **Restart probability hyperparameter tuning for label propagation.** (A) Each point in each boxplot represents the average rank for a gene set collection–network combination, where the five restart probabilities that were tried were ranked in terms of performance (auPRC) for each gene set in a gene set collection using the standard competition ranking. A restart probability of 0.85 was chosen for this study as it resulted in good overall performance as well as low variance in performance for the different genset-collection–network combinations. (B) The performance for each individual gene set collection–network combination is compared across the five restart probabilities: 0.55, 0.65, 0.75, 0.85 and 0.95. The methods are ranked by median value of auRPC with the highest scoring method on the left. There is no strong dependence of auRPC on the restart probability.

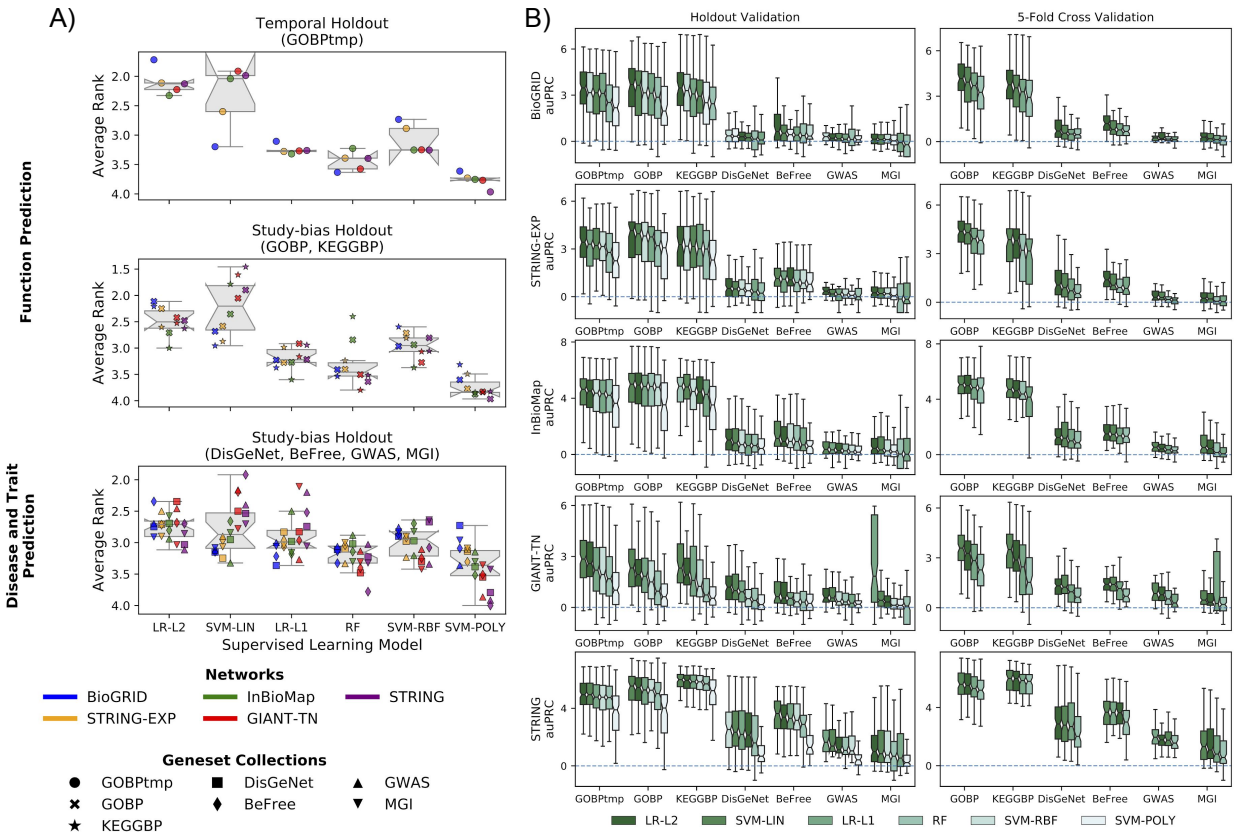


Figure A.2: **Comparison of classifiers for supervised learning.** (A) Each point in each boxplot represents the average rank for a gene set collection–network combination, where the four classifiers were ranked by the auPRC for each gene set in a gene set collection using the standard competition ranking. Logistic regression with L2 regularization (LR-L2) was chosen as the classifier for supervised learning as it had slightly better overall performance than a linear support vector machine (SVM). (B) The auRPC for each individual gene set collection–network combination is compared across the four supervised learning classifiers: logistic-regression with L1 regularization (LR-L1), LR-L2, SVM, and a random forest (RF). The classifiers are ranked by median value with the best performing one on the left.

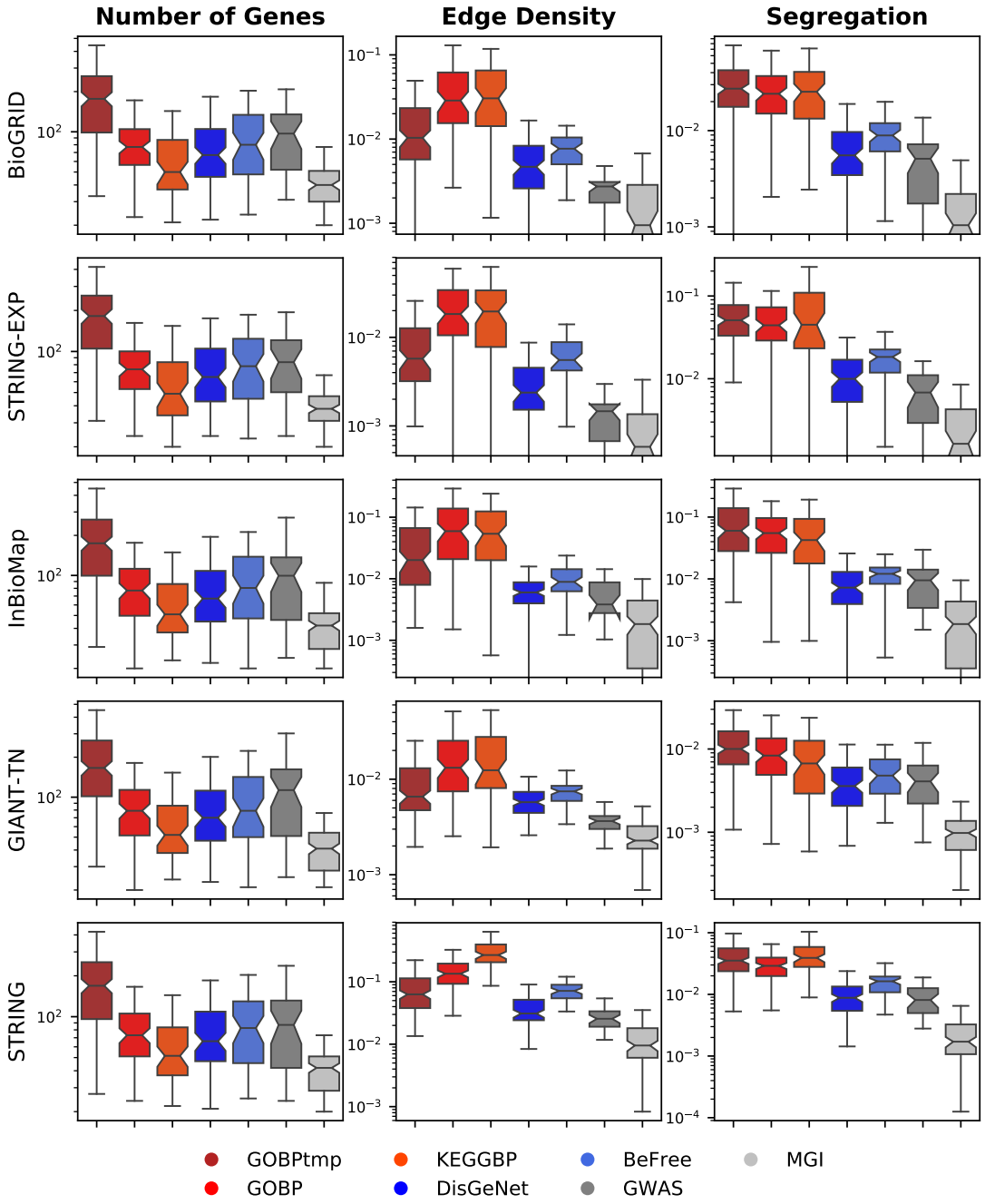


Figure A.3: **Network properties for the different gene set collections.** The gene set collections can be broken up into three prediction tasks; function (GOBPtmp, GOBP, KEGGBP; reds), disease (DiGeNet, BeFree; blues) and trait (GWAS, MGI; greys). In general, there is little difference in the number of genes across the different type prediction tasks (i.e. function, disease and trait), except for GOBPtmp which has the largest number of genes due to the fact the gene sets need to be larger to have enough with at least 10 testing genes. Edge density and segregation are highest for the function gene sets (GOBPtmp, GOBP, KEGGBP) and lowest for the disease and trait gene sets (DisGeNet, BeFree, GWAS, MGI).

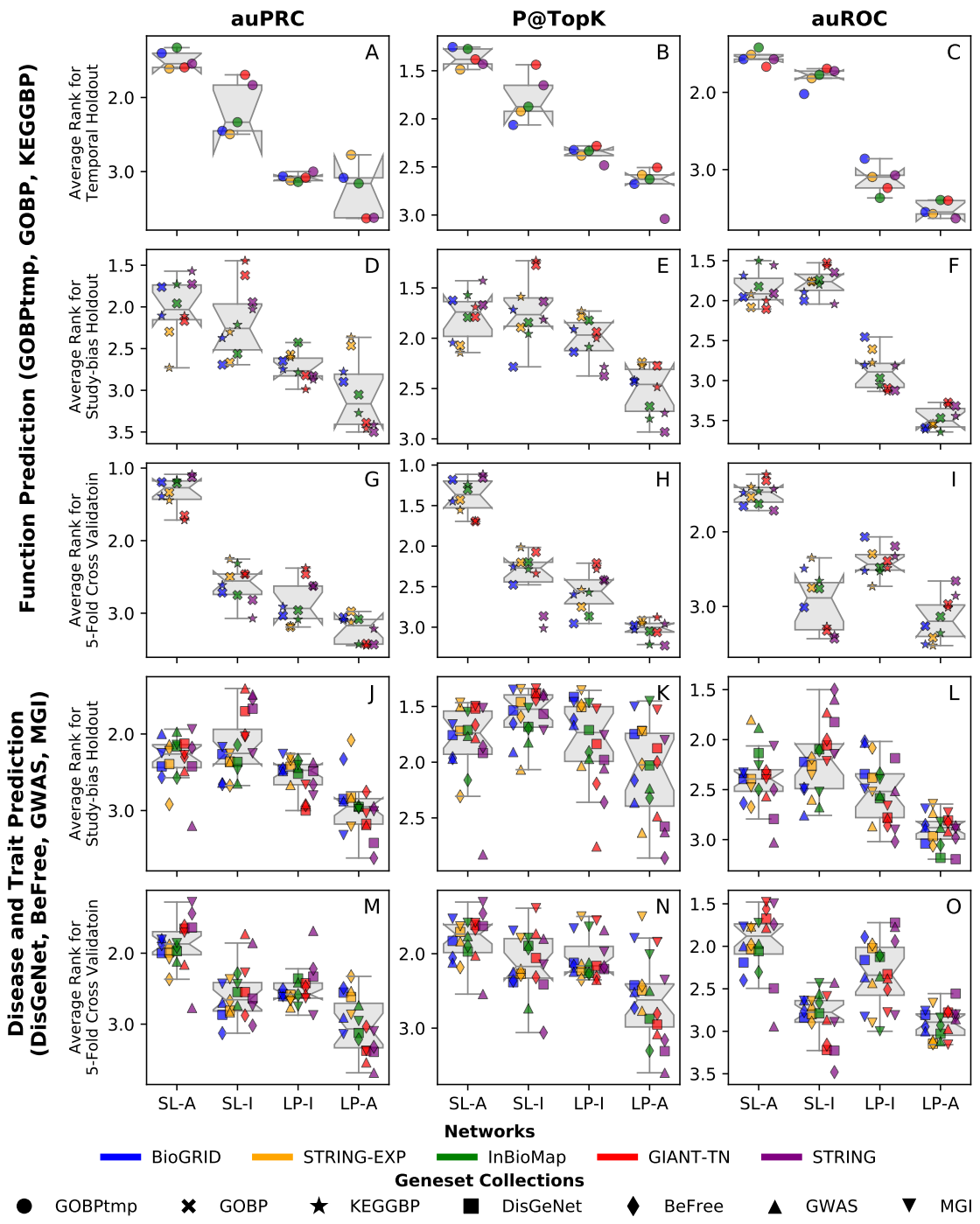


Figure A.4: **Average rank of the four methods for all evaluation metrics and validation schemes.** Each point in a boxplot represents the average rank for a gene set collection–network combination, where the four methods were ranked in terms of performance for each gene set in a gene set collection using the standard competition ranking. Different colors represent different networks and different marker shapes represent different gene set collections.

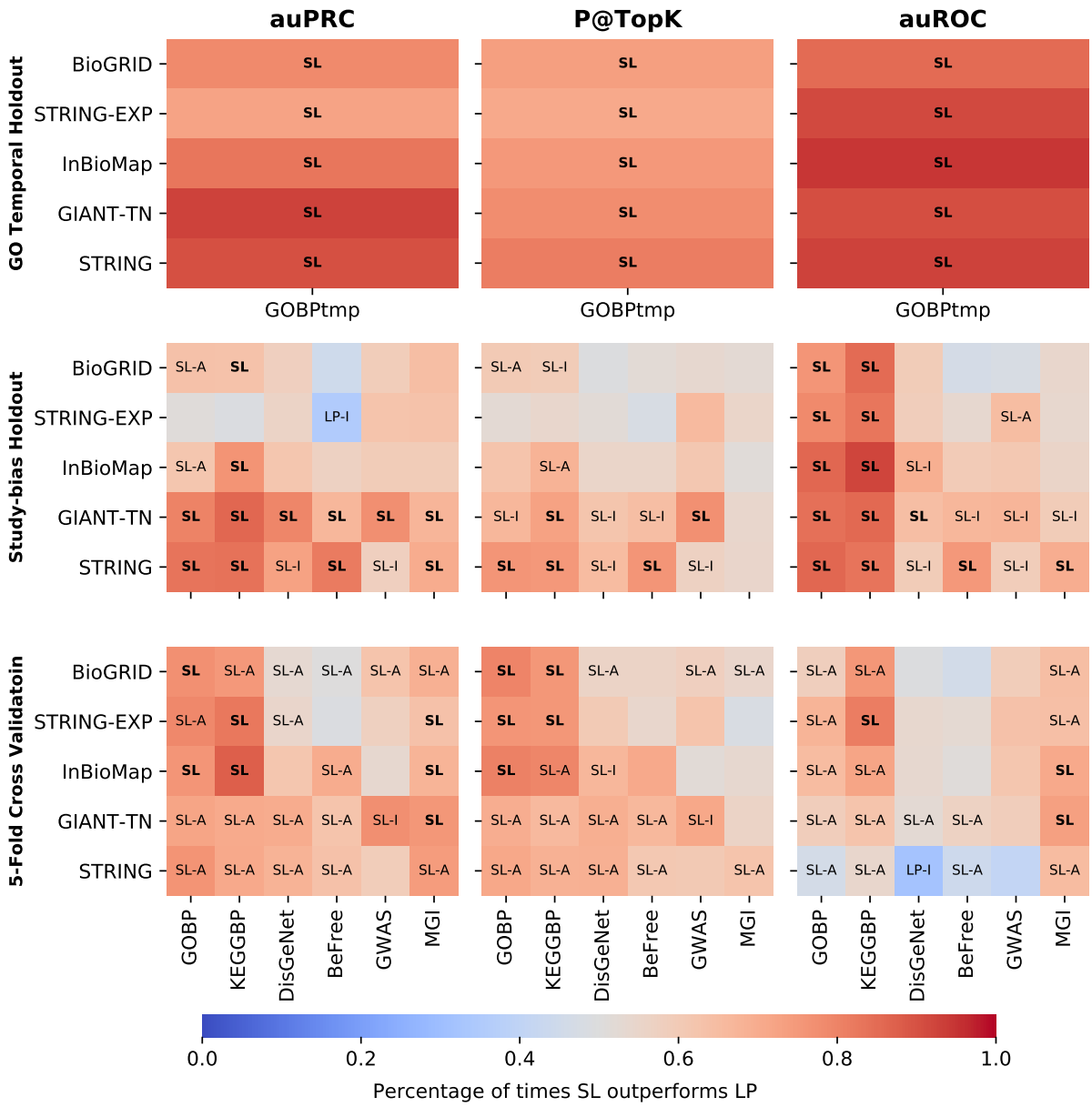


Figure A.5: **Testing for a statistically significant difference between SL and LP methods using all evaluation metrics and validation schemes.** For each network-gene set combination, each method is compared to the two methods from the other class (i.e. SL-A vs LP-I, SL-A vs LP-A, SL-I vs LP-I, SL-I vs LP-A). If a method was found to be significantly better than both methods from the other class (Wilcoxon ranked-sum test with an FDR threshold of 0.05), the cell is annotated with that method. If both models in that class were found to be significantly better than the two methods in the other class, the cell is annotated in bold with just the class. The color scale represents the fraction of gene sets that were higher for the SL methods across all four comparisons. The first column uses GOBP temporal holdout, whereas the remaining 6 columns use study-bias holdout. B) SL methods show a statistically significant improvement over LP methods, especially for function prediction/

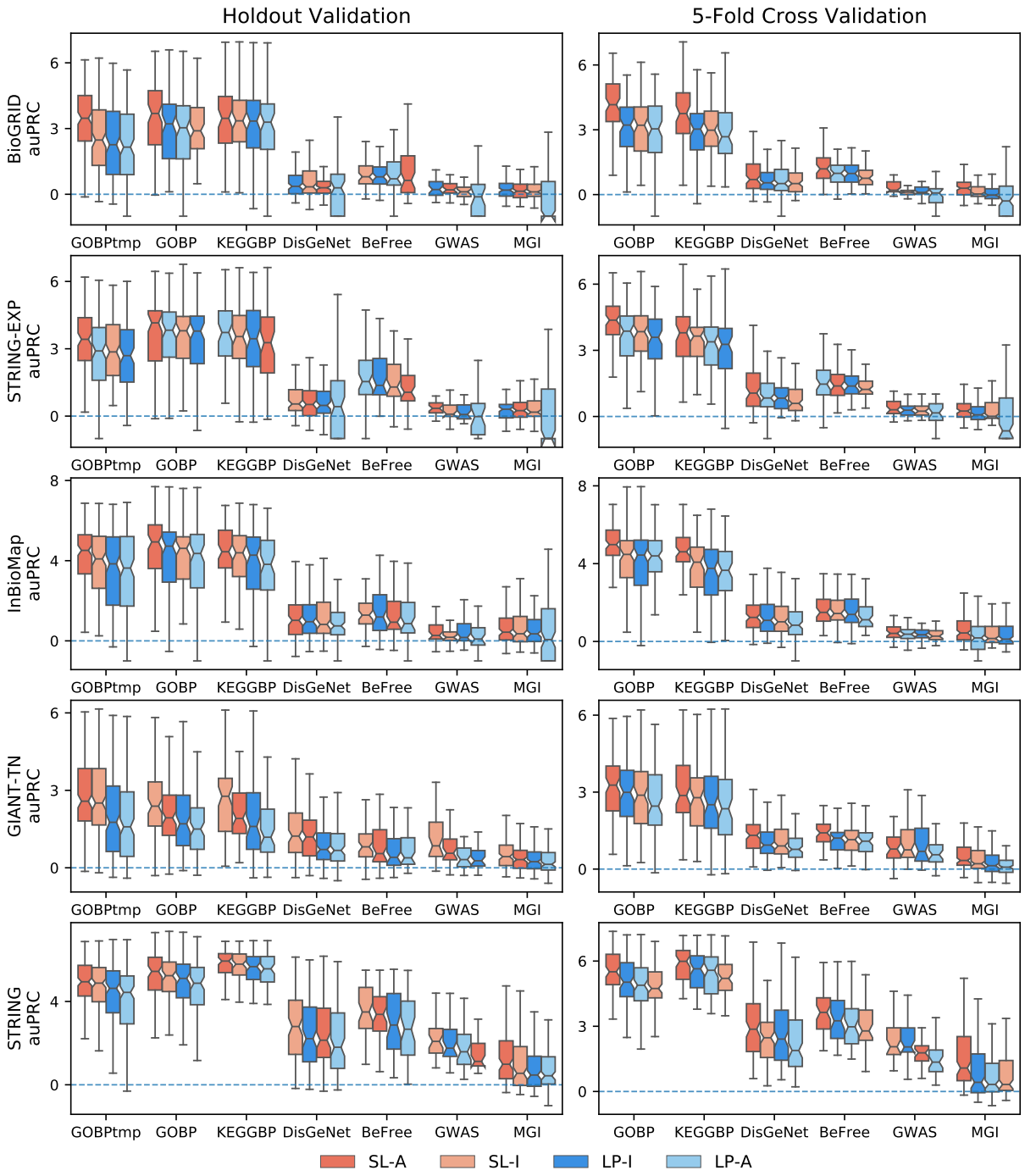


Figure A.6: **Boxplots for auPRC performance across all gene set collection–network combinations.** The performance for each individual gene set collection–network combination is compared across the four methods; SL-A (red), SL-I (light red), LP-I (blue), and LP-A (light blue). The methods are ranked by median value with the highest scoring method on the left. The first column contains temporal and study-bias holdout, and the second column is 5FCV. The scoring metric is auPRC.

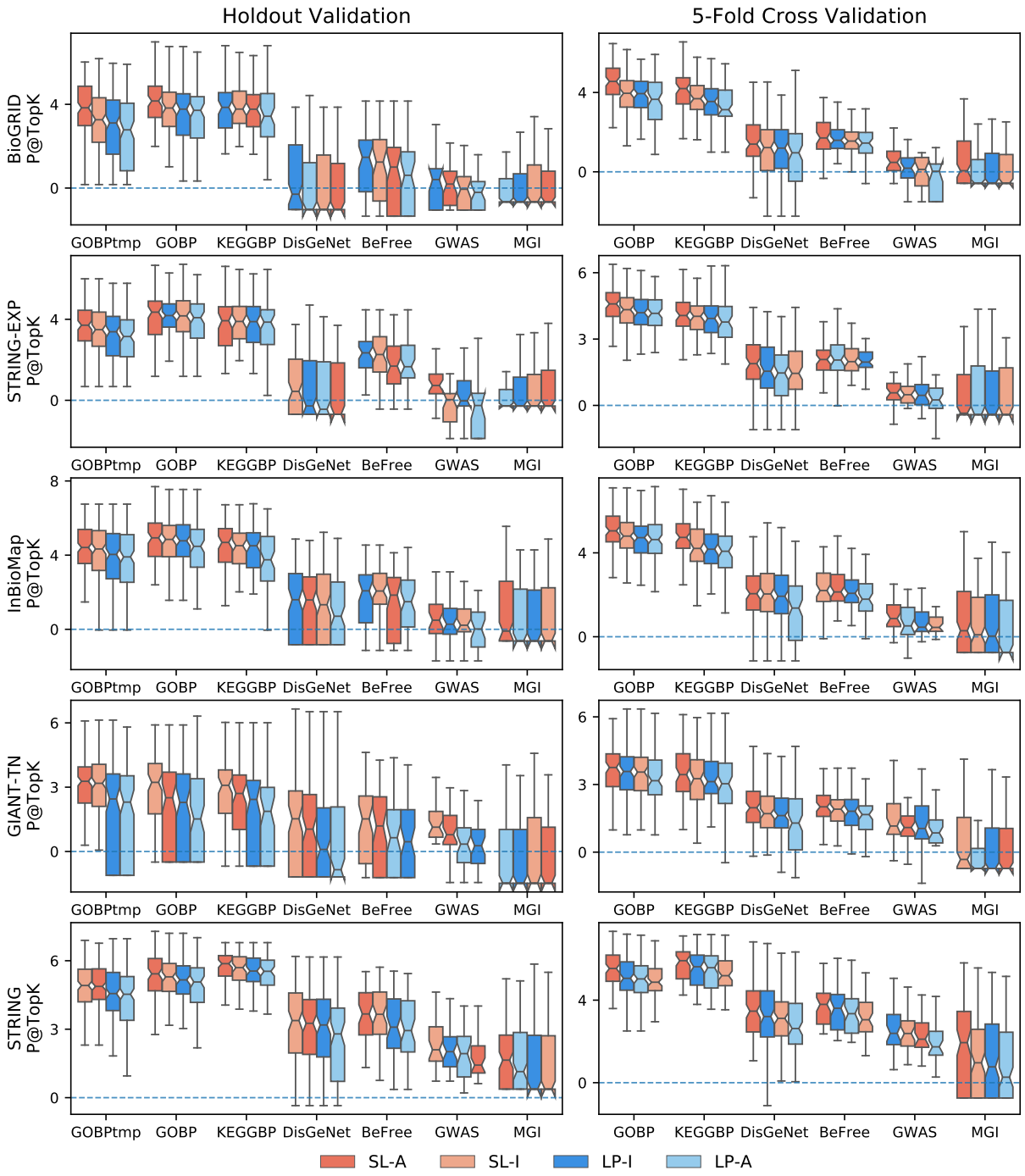


Figure A.7: **Boxplots for P@TopK performance across all gene set collection–network combinations.** The performance for each individual gene set collection–network combination is compared across the four methods; SL-A (red), SL-I (light red), LP-I (blue), and LP-A (light blue). The methods are ranked by median value with the highest scoring method on the left. The first column contains temporal and study-bias holdout, and the second column is 5FCV. The scoring metric is P@TopK.

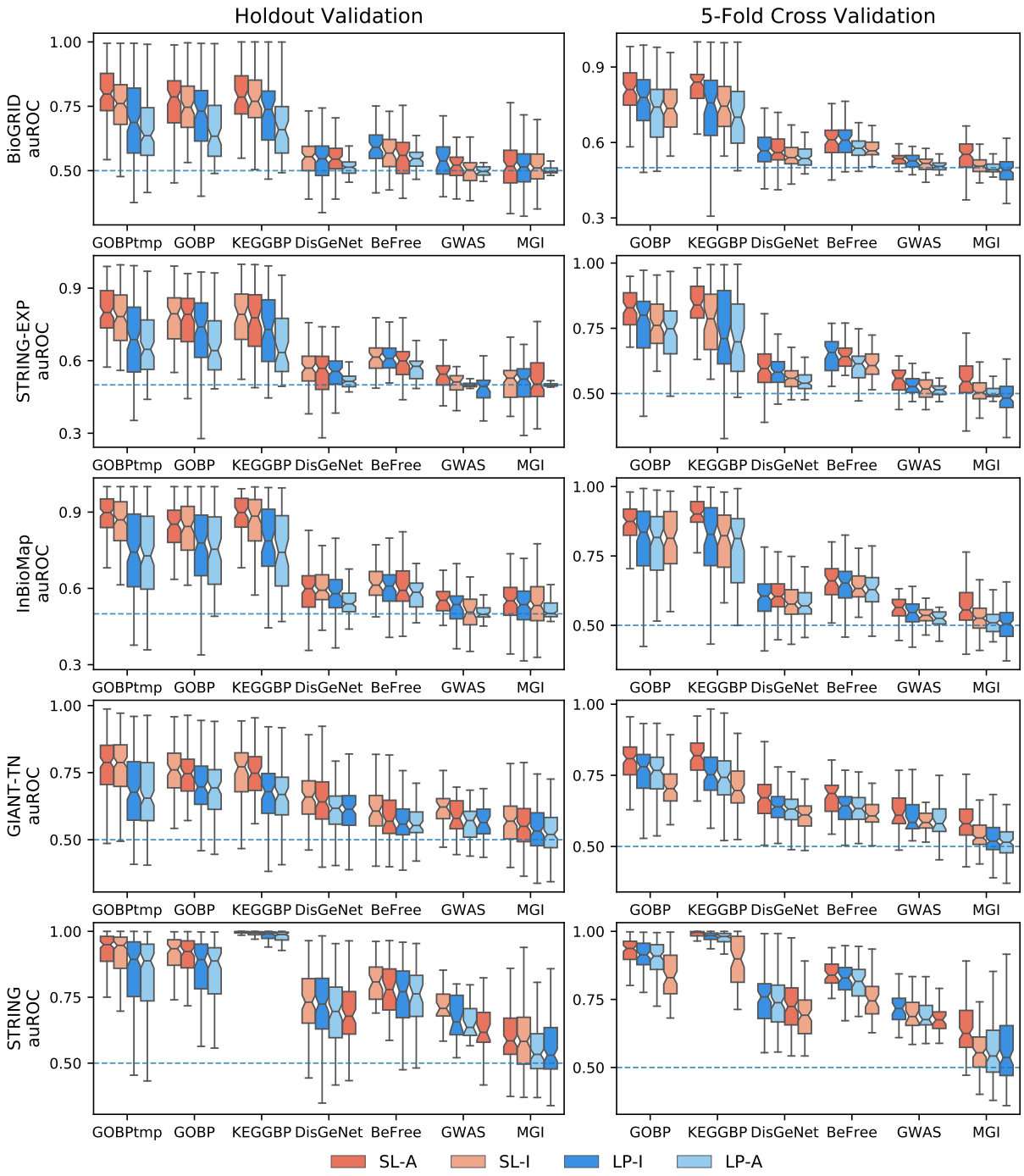


Figure A.8: **Boxplots for auROC performance across all gene set collection–network combinations.** The performance for each individual gene set collection–network combination is compared across the four methods; SL-A (red), SL-I (light red), LP-I (blue), and LP-A (light blue). The methods are ranked by median value with the highest scoring method on the left. The first column contains temporal and study-bias holdout, and the second column is 5FCV. The scoring metric is auROC.

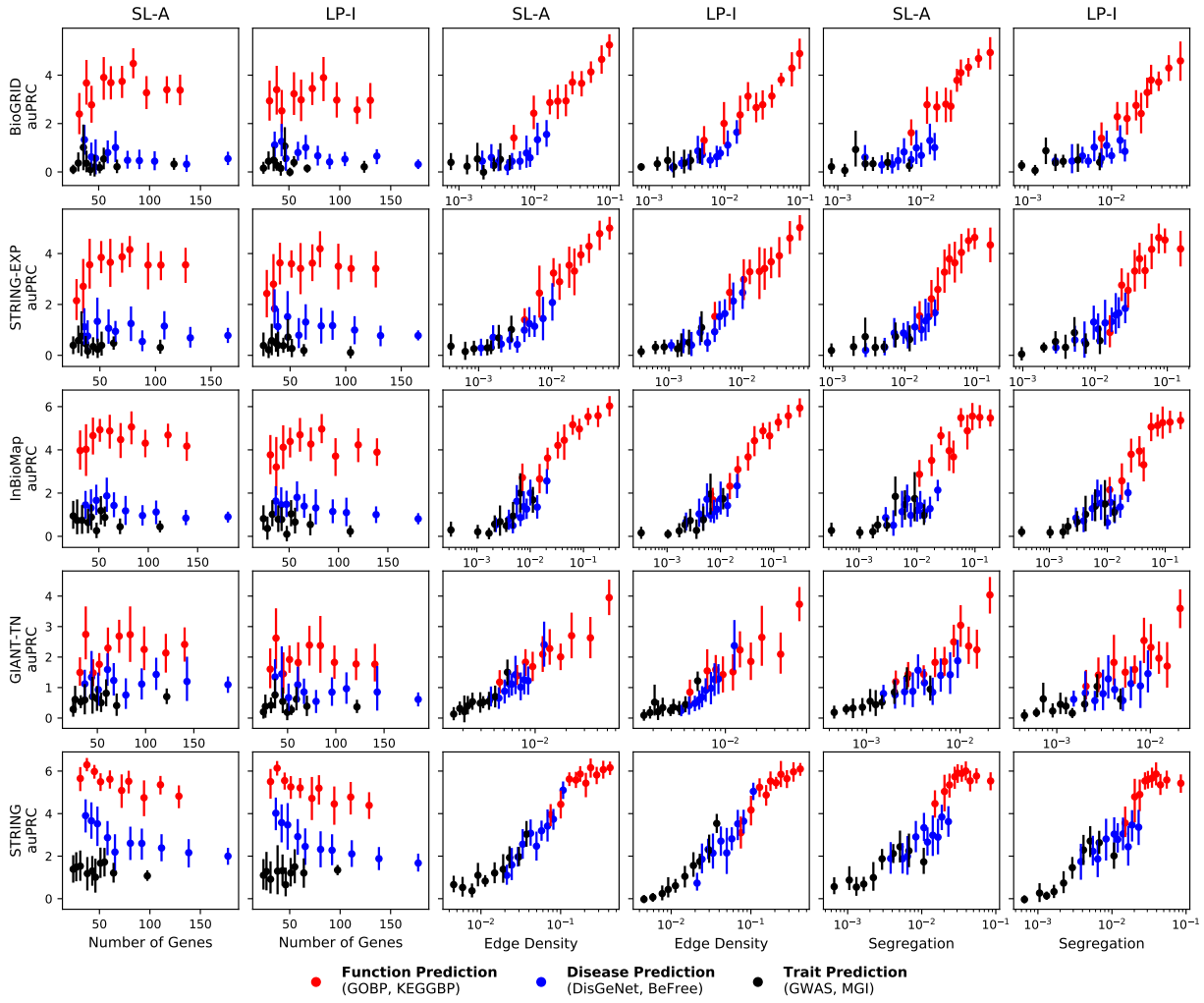


Figure A.9: **Performance vs Network/gene set properties for all networks.** SL-A is able to capture network information as efficiently as LP-I across all networks. There is no correlation between the number of genes in the gene set versus performance, but there is a strong correlation between the performance and the edge density as well as segregation. The different colored dots represent function gene sets (red, GOBP and KEGGBP), disease gene sets (blue, DIsGeNet and BeFree), and trait gene sets (black, GWAS and MGI). The vertical line is the 95% confidence interval and the performance metric is auPRC.

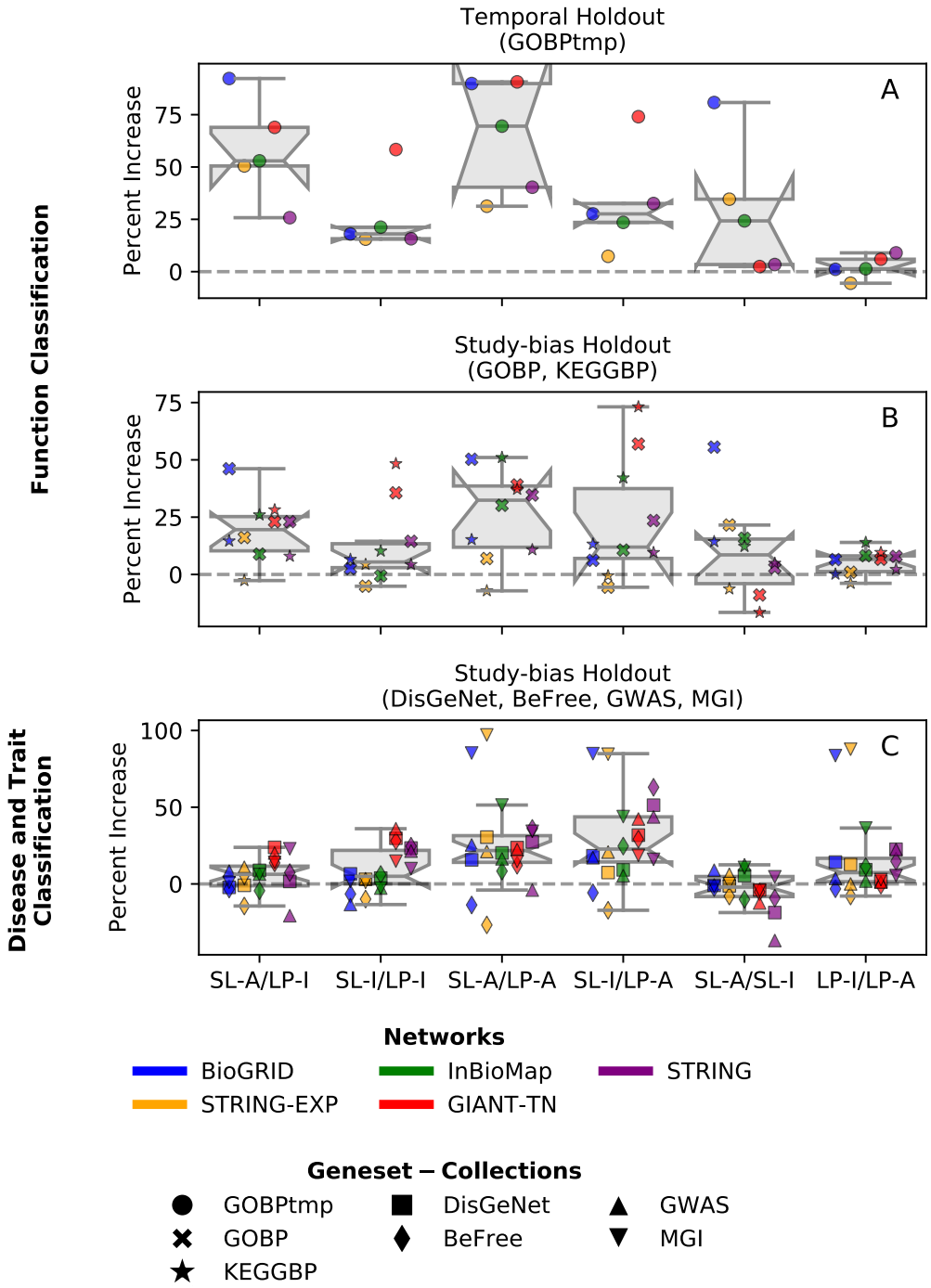


Figure A.10: **Effect size for every pair of methods.** Each point is the median percent increase for every gene set collection–network combination. (A) Functional prediction tasks using GOBP temporal holdout, (B) Functional prediction tasks using study-bias holdout for GOBP and KEGGBP, and (C) Disease and trait prediction tasks using study-bias holdout for DisGeNet, BeFree, GWAS, and MGI. The results are shown for auPRC where different colors represent different networks and different marker styles represent the different gene set collections.

A.2.1 Label propagation with negative examples

In this section, we present alternative LP results that consider both positive and negative examples (LPN). We perform the same hyperparameter tuning for the restart parameter as LP (Appendix A.1.2) and find the optimal restart parameter of 0.45 (Figure A.11). This optimal value for the restart parameter in LPN is relatively low compared to the optimum value for LP, except for the GIANT-TN network, where both LP and LPN prefer a higher restart value. It is worth noting that just like with LP, the dependence on the restart parameter is minimal (Figure A.11B). We also include boxplots comparing label propagation with and without negative examples (Figure A.12). Lastly, we show a side-by-side comparison of the ranking analysis (Figure A.13) and Wilxcon analysis (Figure A.14) using label propagation with and without negative examples. Our results show that even though using negative examples slightly increases performance in label propagation, the results, when compared against supervised learning, remain unchanged.

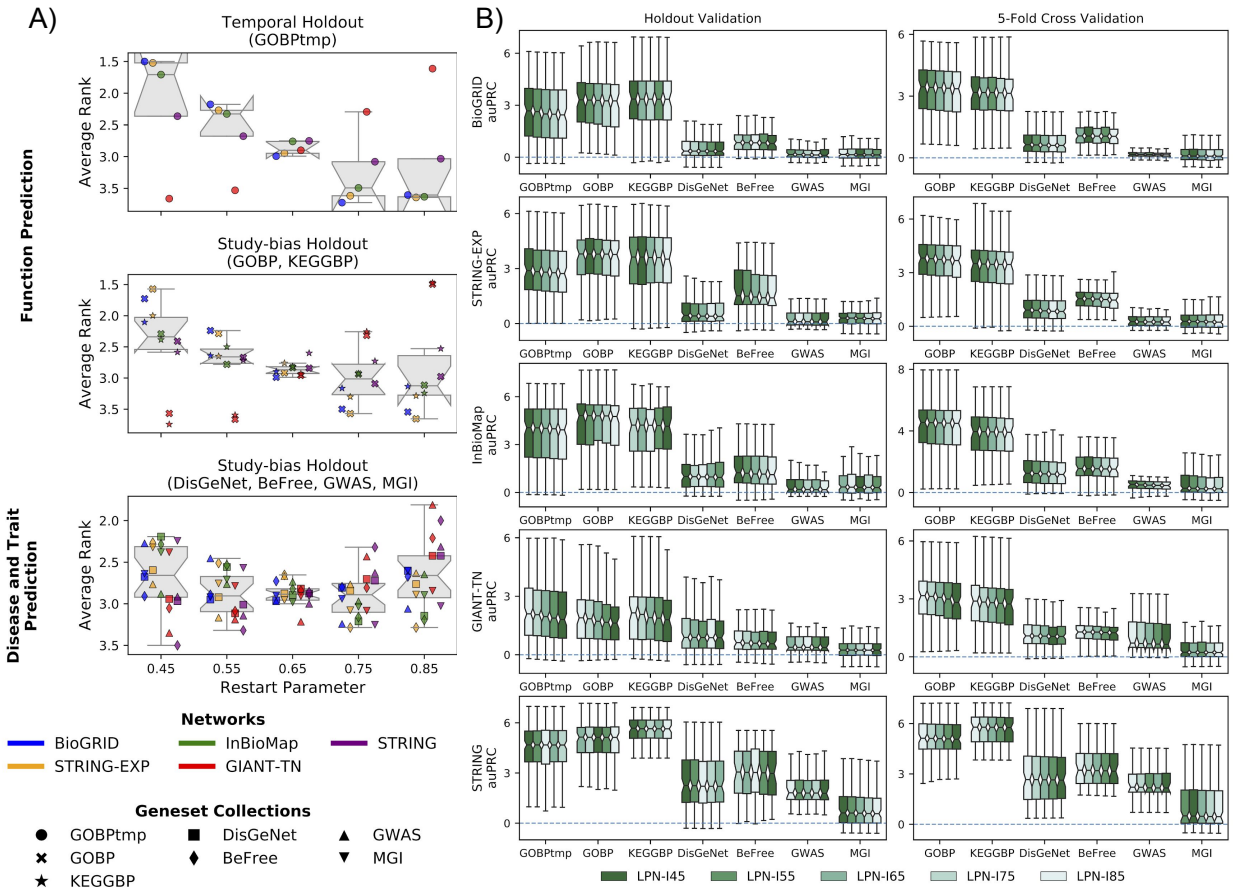


Figure A.11: **Tuning the restart probability hyperparameter when using negative examples in label propagation.** (A) Each point in each boxplot represents the average rank for a gene set collection–network combination, where the five restart probabilities (0.45, 0.55, 0.65, 0.75 and 0.85) were ranked in terms of performance (auPRC) for each gene set in a gene set collection using the standard competition ranking. A restart probability of 0.45 was chosen as optimal. (B) The performance for each individual gene set collection–network combination is compared across the five restart probabilities. The methods are ranked by median value of auRPC with the highest scoring method on the left. There is no strong dependence of auRPC on the restart probability.

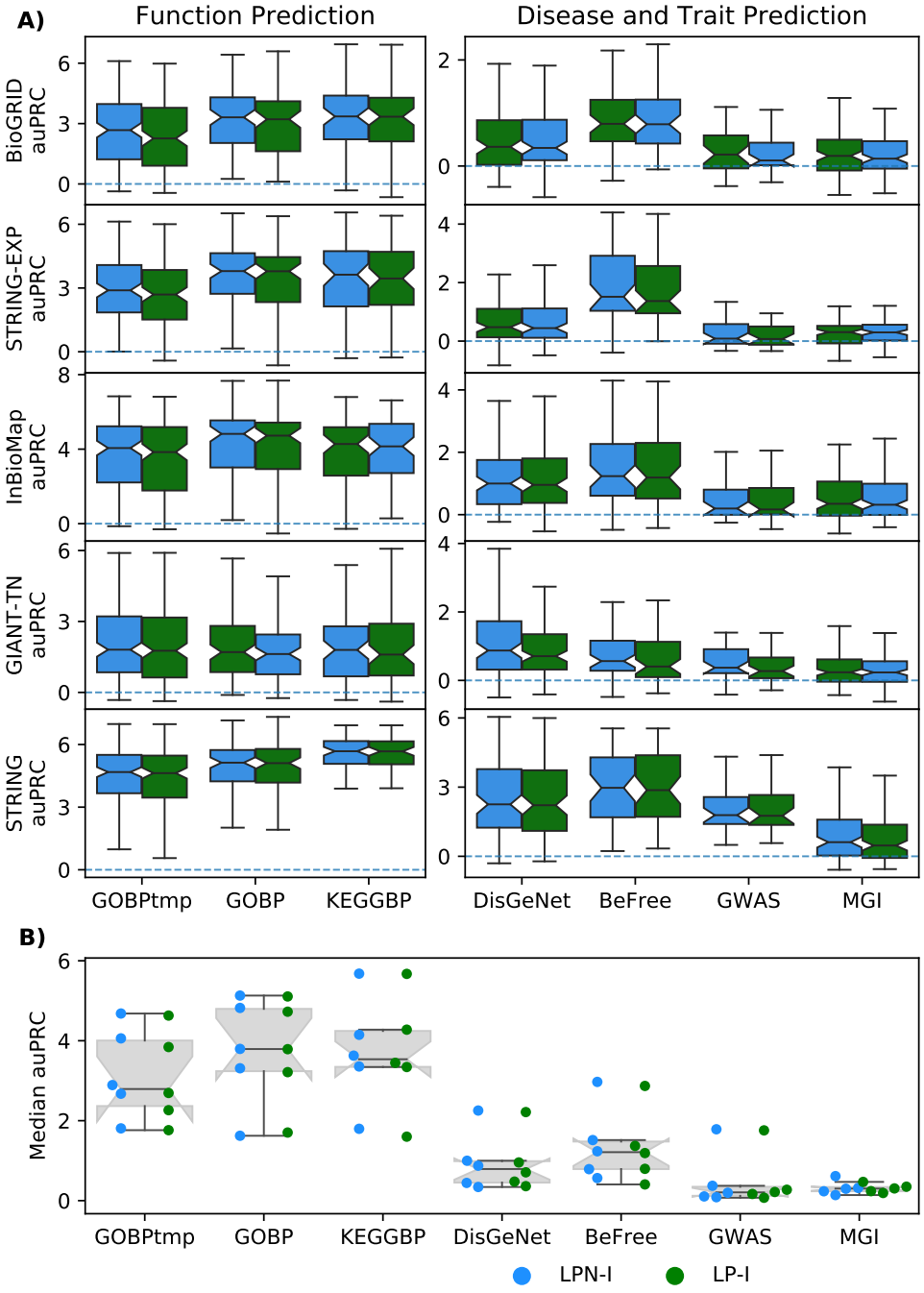


Figure A.12: **Boxplots for performance across all gene set collection–network combinations for label propagation on the influence matrix with and without using negative examples.** (A) The performance for each individual gene set collection–network combination is compared for label propagation with negative examples (LPN-I, blue) and label propagation without negative examples (LP-I, green). The methods are ranked by median value with the highest scoring method on the left. Results show LPN-I has a moderately increased performance when compared to LP-I. (B) Each point in the plot is the median value from one of the boxplots in A. This shows that both LPN and LP methods perform better for function prediction compared to disease/trait prediction.

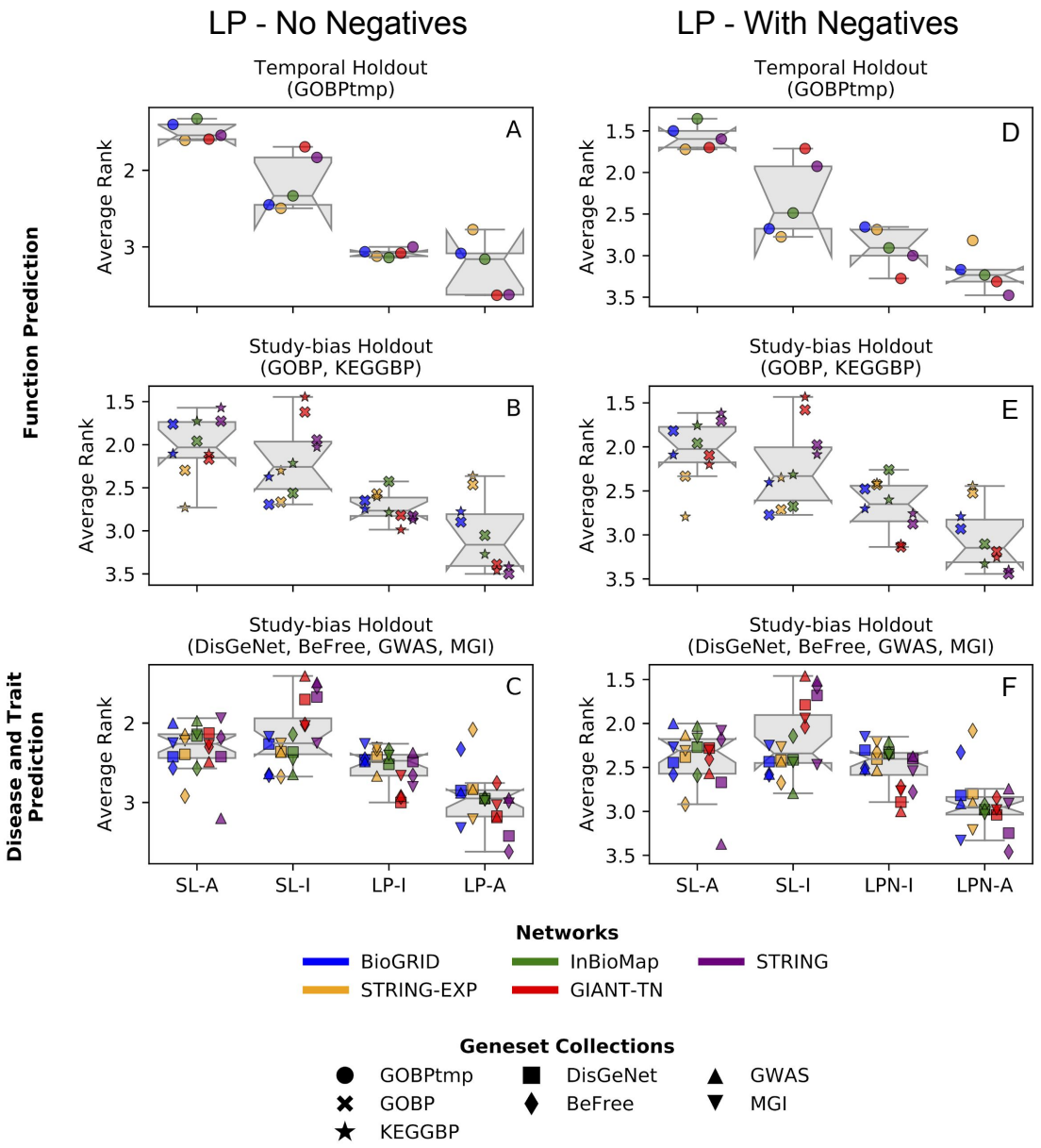


Figure A.13: **Comparing results from average rank analysis with and without using negative examples in label propagation.** The left column has label propagation without negative examples (LP) and the right column has label propagation with negative examples (LPN). Each point in each boxplot represents the average rank for a gene set collection–network combination, obtained based on ranking the four methods in terms of performance for each gene set in a gene set collection using the standard competition ranking. (A, D) Functional prediction tasks using GOBP temporal holdout, (B, E) Functional prediction tasks using study-bias holdout for GOBP and KEGGBP, and (C, F) Disease and trait prediction tasks using study-bias holdout for DisGeNet, BeFree, GWAS, and MGI. The results are shown for auPRC where different colors represent different networks and different marker styles represent the different gene set collections. The results show that no substantial difference can be seen between using or not using negative examples in label propagation.

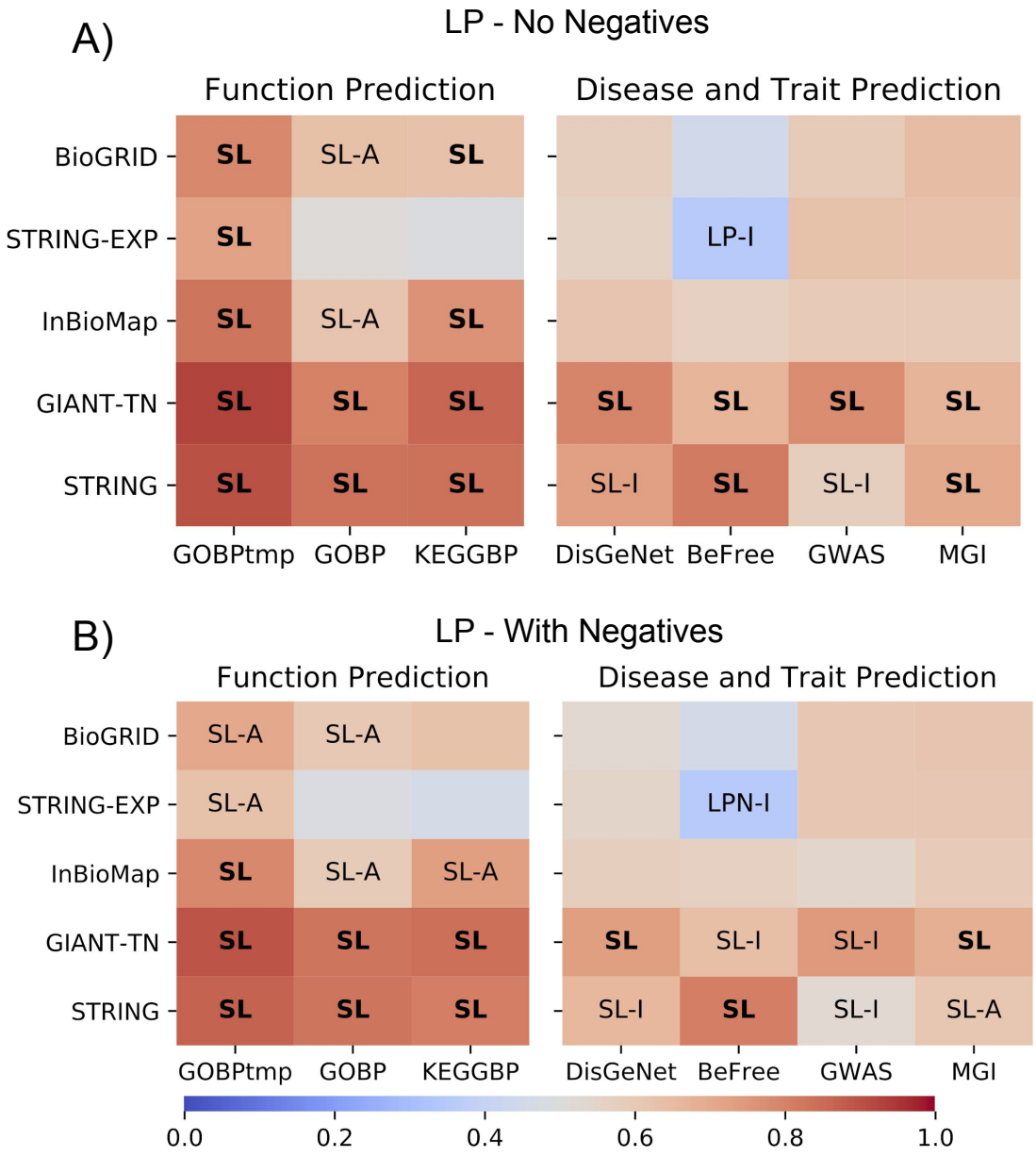


Figure A.14: **Comparing the Wilcoxon statistical test analysis with and without using negative examples in label propagation.** (A) Label propagation without negative examples (LP) and (B) label propagation with negative examples (LPN). For each network-gene set combination, each method is compared to the two methods from the other class. If a method was found to be significantly better than both methods from the other class (Wilcoxon ranked-sum test with an FDR threshold of 0.05), the cell is annotated with that method. If both models in that class were found to be significantly better than the two methods in the other class, the cell is annotated in bold with just the class. The color scale represents the fraction of gene sets that were higher for the SL methods across all four comparisons. The first column uses GOBP temporal holdout, whereas the remaining 6 columns use study-bias holdout. The results show that no substantial difference can be seen between using or not using negative examples in label propagation.

APPENDIX B

PECANPY

B.1 Existing *node2vec* implementations

We list existing open-source *node2vec* implementations in this section, all of which are gathered and accessed as of Feburary 2021. Besides the original implementations, we only tested `nodevectors` implementation as all the others do not contain any feature that could result in significant runtime or memory usage improvements, as we detailed below.

1. **Original Python implementation** (https://github.com/aditya-grover/node2vec)

2. **Original C++ implementation** (https://github.com/snap-stanford/snap/tree/master/examples/node2vec)

3. **`node2vectors`** (https://github.com/VHRanger/nodevectors) is similar to PecanPy-SparseOTF, which computes the transition probabilities on-the-fly using the cache optimized CSR graph object. However, it cannot handle dense networks effectively (Section 3.2.3).

4. **TensorFlow implementation** (https://github.com/apple2373/node2vec) replaces `gensim`'s word2vec with TensorFlow for training embedding using the precomputed random walks. However, we observe that the embedding step of the *node2vec* contributes minimally to the total runtime (Figure B.1, B.2). Thus, this implementation will provide significant runtime improvements.

5. **Python 3 implementation** (https://github.com/eliorc/node2vec) is the same as the original Python implementation, but has been reformatted to be compatible with Python 3. Thus, we do not expect any explicit performance improvements.

6. **C++ implementaion** (https://github.com/thibaudmartinez/node2vec) is essentially the original C++ implementation, but with a Python API. This implementation is expected to be at best as efficient as the original C++ implementation.

7. **Dependency-less C++ implementation** (https://github.com/xgfs/node2vec-c) is the same as the original C++ implementation, but as a standalone program without other SNAP functionalities.

8. **Implementation with different sampling method other than the original alias method** (https://github.com/NilsFrahm/Node2vec). This implementation is the same as the original Python implementation, with an alternative random distribution sampling approach, which does not have a significant impact on the runtime.

B.2 Additional results

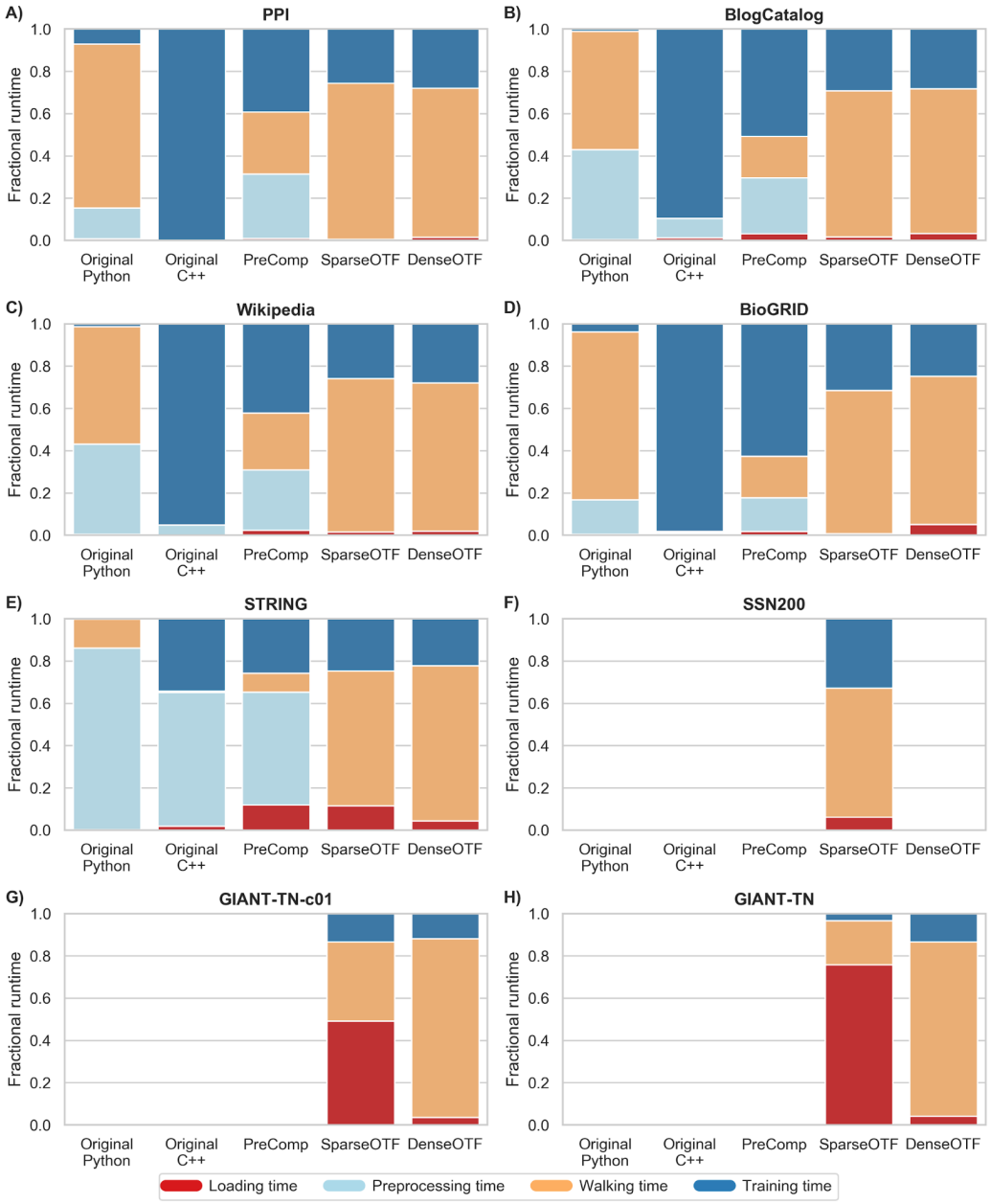


Figure B.1: **Fraction of runtime contributed by each stage of node2vec in different implementations using multiple cores.** Each panel corresponds to a single network and each stacked bar within a panel corresponds to an individual node2vec implementation. The height of each segment within a bar represents the fraction of runtime contributed by each of the different stages of node2vec, tested in a multi-core configuration.

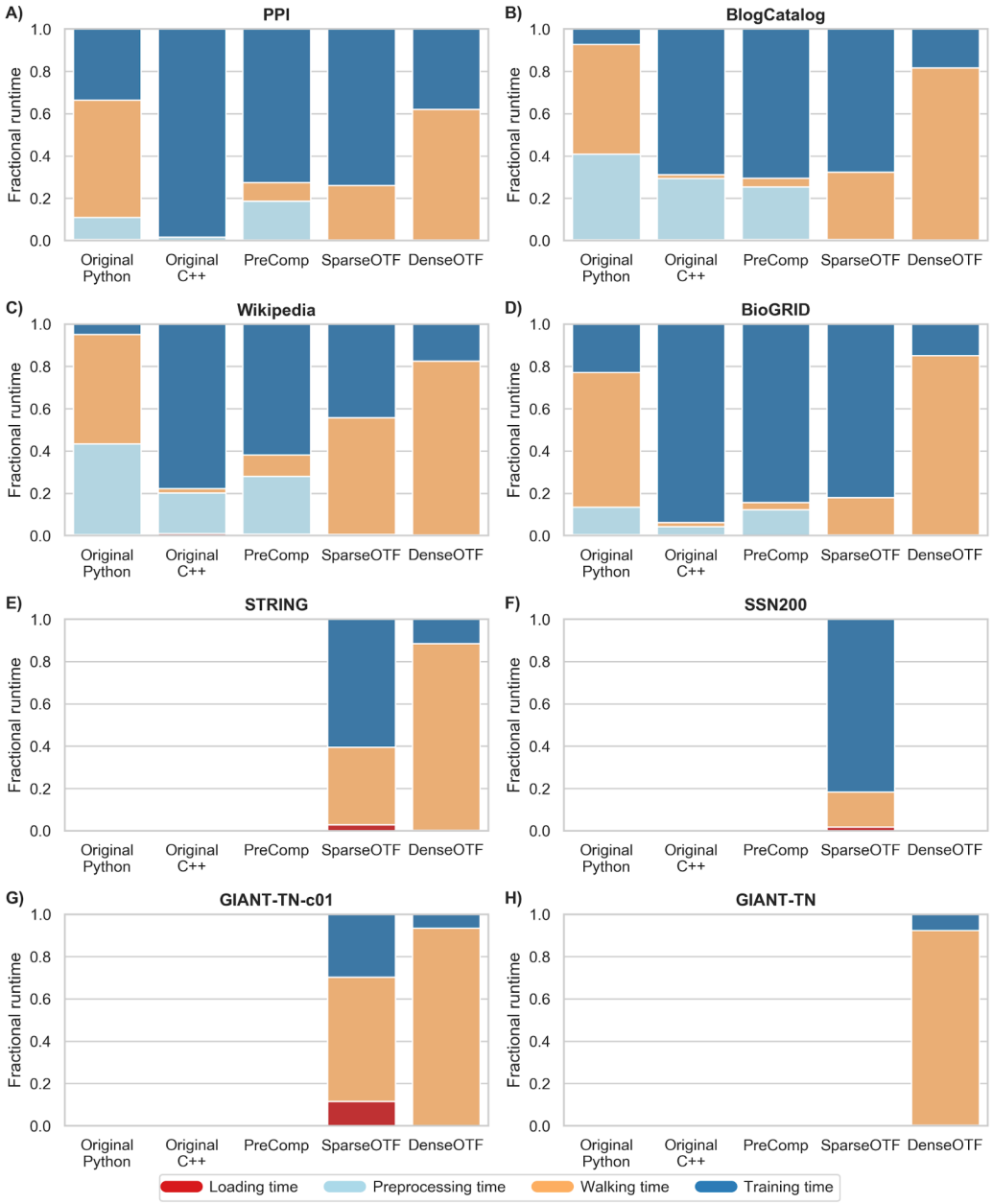


Figure B.2: **Fraction of runtime contributed by each stage of node2vec in different implementations using a single core.** Each panel corresponds to a single network and each stacked bar within a panel corresponds to an individual node2vec implementation. The height of each segment within a bar represents the fraction of runtime contributed by each of the different stages of node2vec, tested in a single-core configuration.

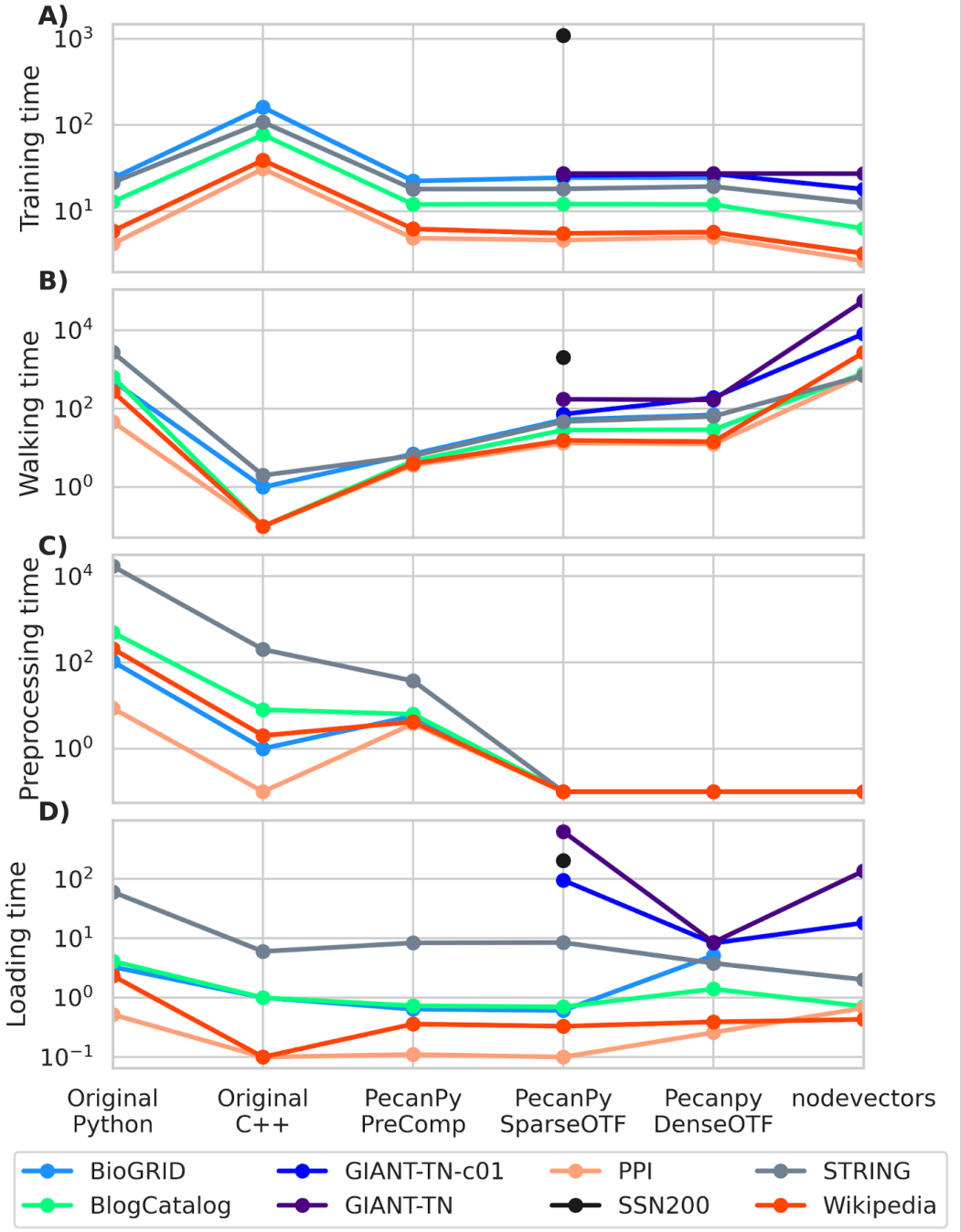


Figure B.3: **Raw runtimes of each stage of node2vec in different implementations using multiple cores.** Each parallel plot corresponds to one of four stages of node2vec. Each line traces the raw runtime (points on parallel y-axes) of a specific network (color) across the different implementations (x-axis). In all plots, absence of a point for a particular network for any implementation indicates that the network failed to load.

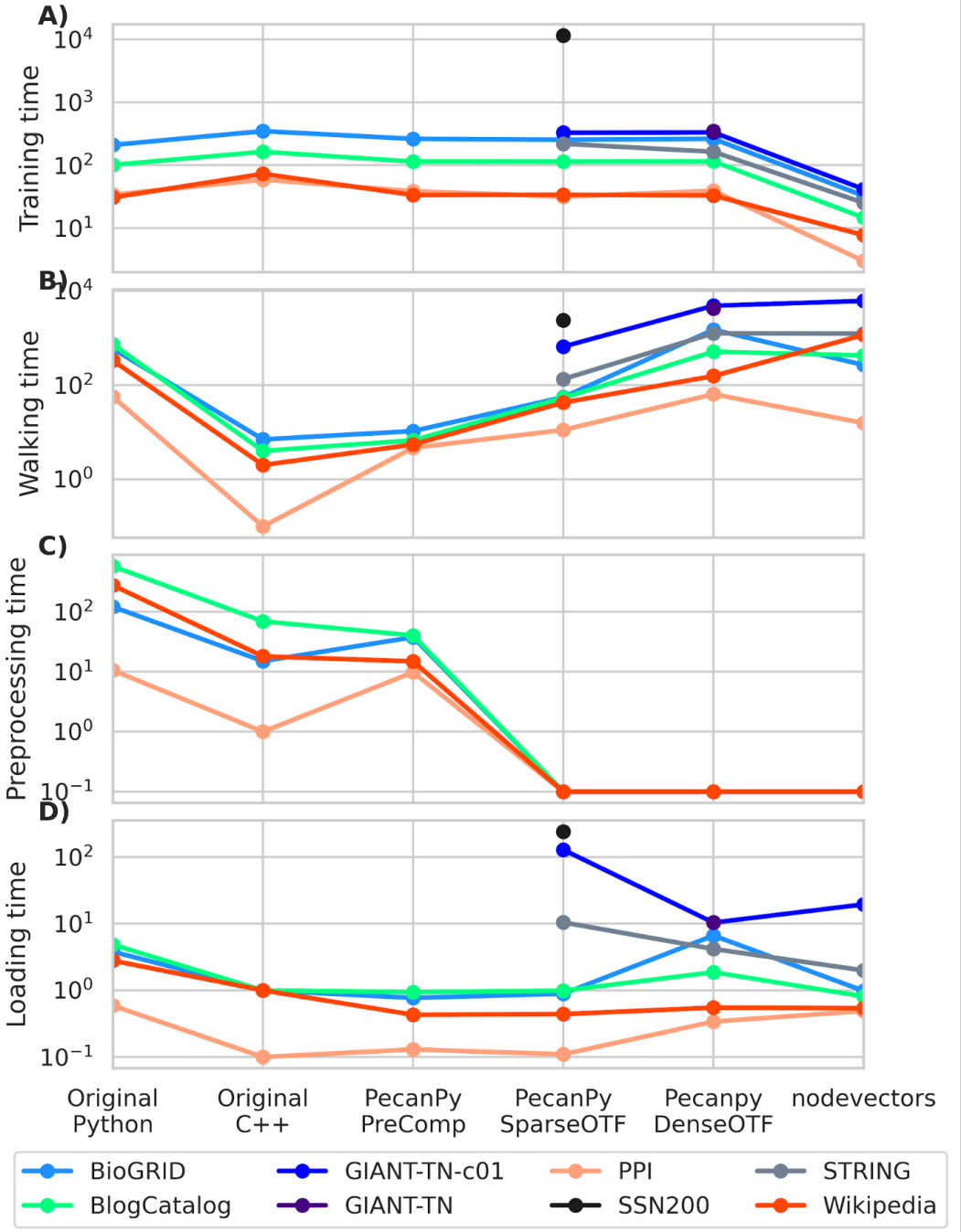


Figure B.4: **Raw runtimes of each stage of node2vec in different implementations using a single core.** Each parallel plot corresponds to one of four stages of node2vec. Each line traces the raw runtime (points on parallel y-axes) of a specific network (color) across the different implementations (x-axis). In all plots, absence of a point for a particular network for any implementation indicates that the network failed to load.

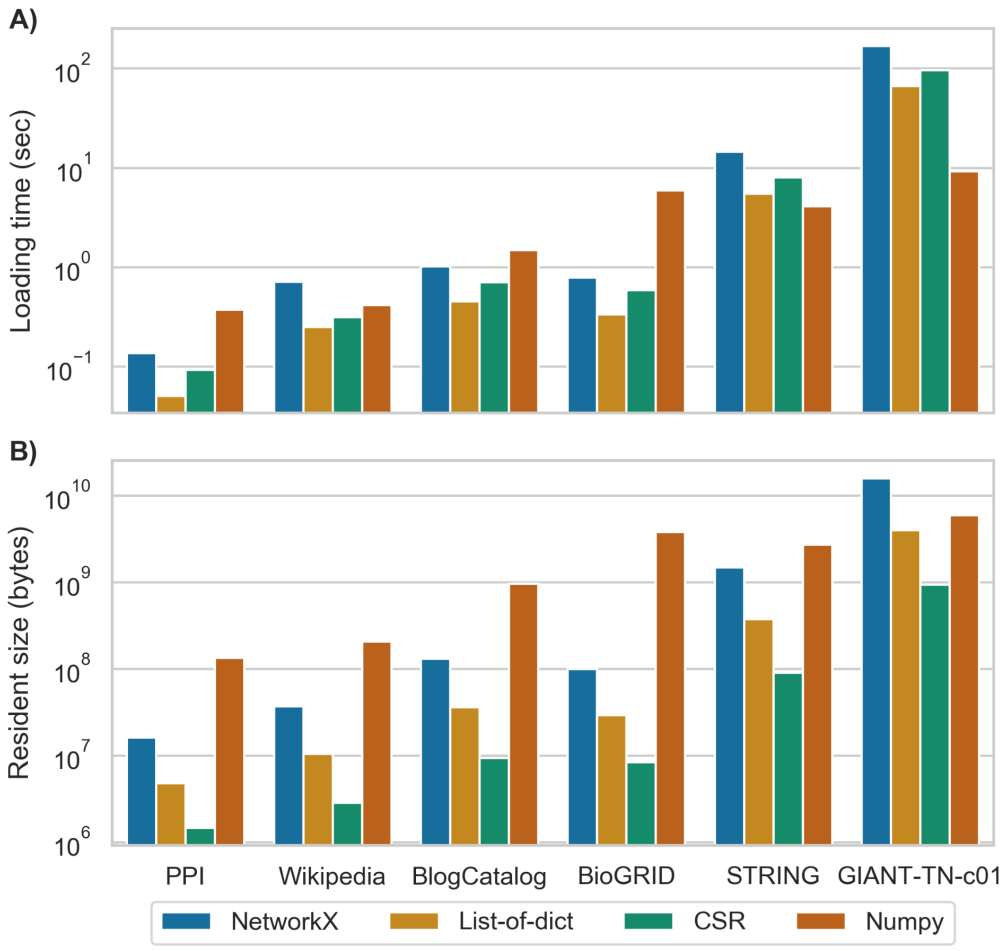


Figure B.5: **Effect of graph data structure on loading time and memory usage.** (A) the total time for loading networks. (B) the total memory usage. Each plot contains groups of bars, each group corresponding to a network (among seven select networks), and each bar in the group corresponding to a specific network data structure.

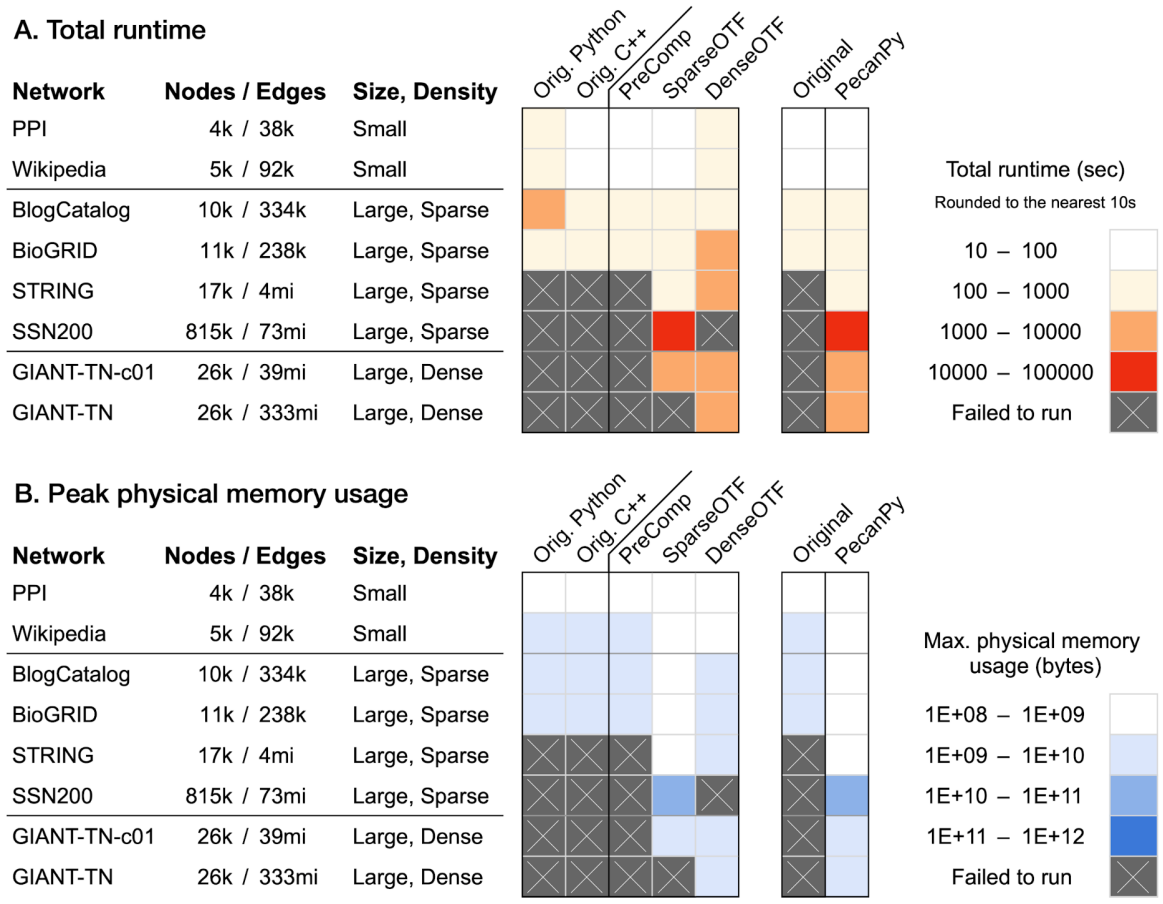


Figure B.6: **Summary of runtime and memory of PecanPy and the original implementations of node2vec using a single core.** The eight networks of varying sizes and densities are along the rows. The software implementations are along the columns. The first heatmap (on the left) shows the performance of the original Python and C++ software along with the three modes of PecanPy (PreComp, SparseOTF, and DenseOTF). The adjacent 2-column heatmap (on the right) summarizes the performance of the original (best of Python and C++ versions) and PecanPy (best of PreComp, SparseOTF, and DenseOTF) implementations. Lighter colors correspond to lower runtime in panel A and lower memory usage in panel B. Crossed grey indicates that the particular implementation (column) failed to run for a particular network (row).

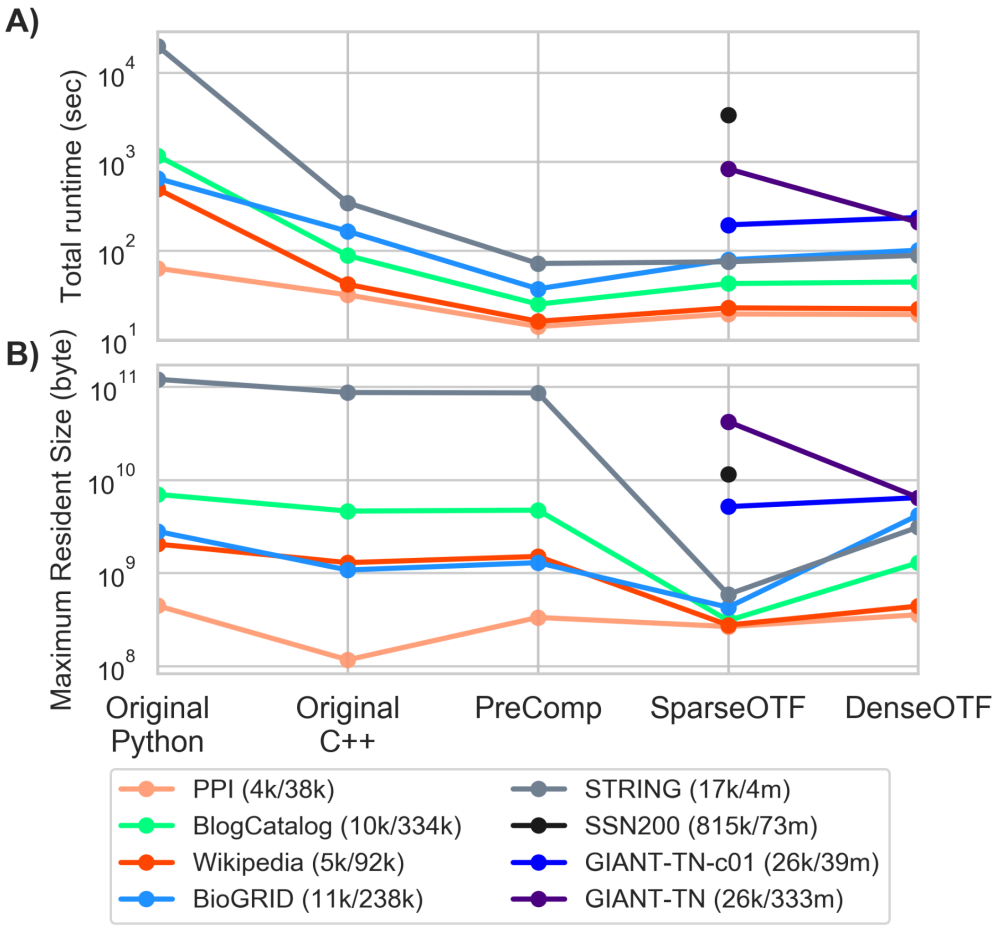


Figure B.7: **Performance of the original Python and C++ implementations and the three new implementations – PreComp, SparseOTF, and DenseOTF – on eight networks using multiple cores.** The parallel plots trace the performance of different node2vec implementations (x-axis) for 8 networks (colored dots/lines; number of nodes/edges are in legend below) in terms of (A) total runtime (seconds) and (B) peak memory usage (bytes). Absence of a dot indicates the failure of a particular implementation to run for a particular network.

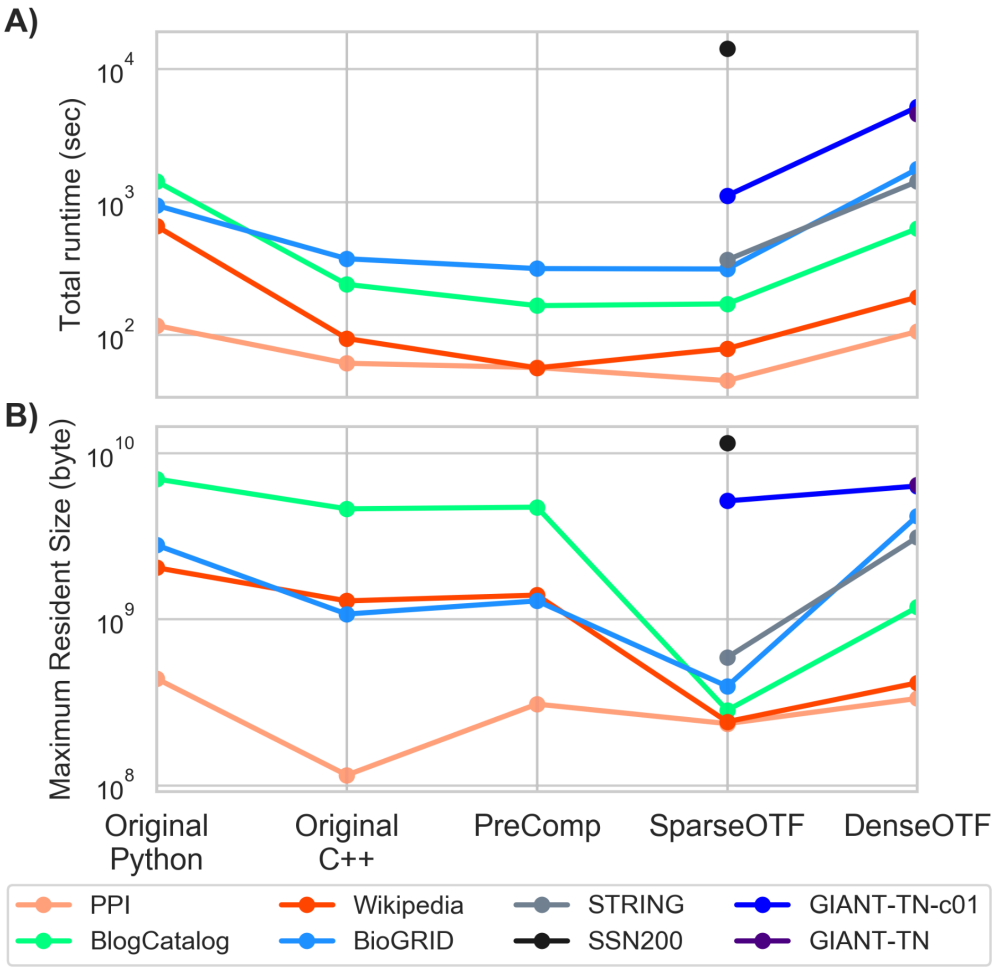


Figure B.8: **Performance of the original Python and C++ implementations and the three new implementations – PreComp, SparseOTF, and DenseOTF – on eight networks using a single core.** The parallel plots trace the performance of different node2vec implementations (x-axis) for 8 networks (colored dots/lines; number of nodes/edges are in legend below) in terms of (A) total runtime (seconds) and (B) peak memory usage (bytes). Absence of a dot indicates the failure of a particular implementation to run for a particular network.

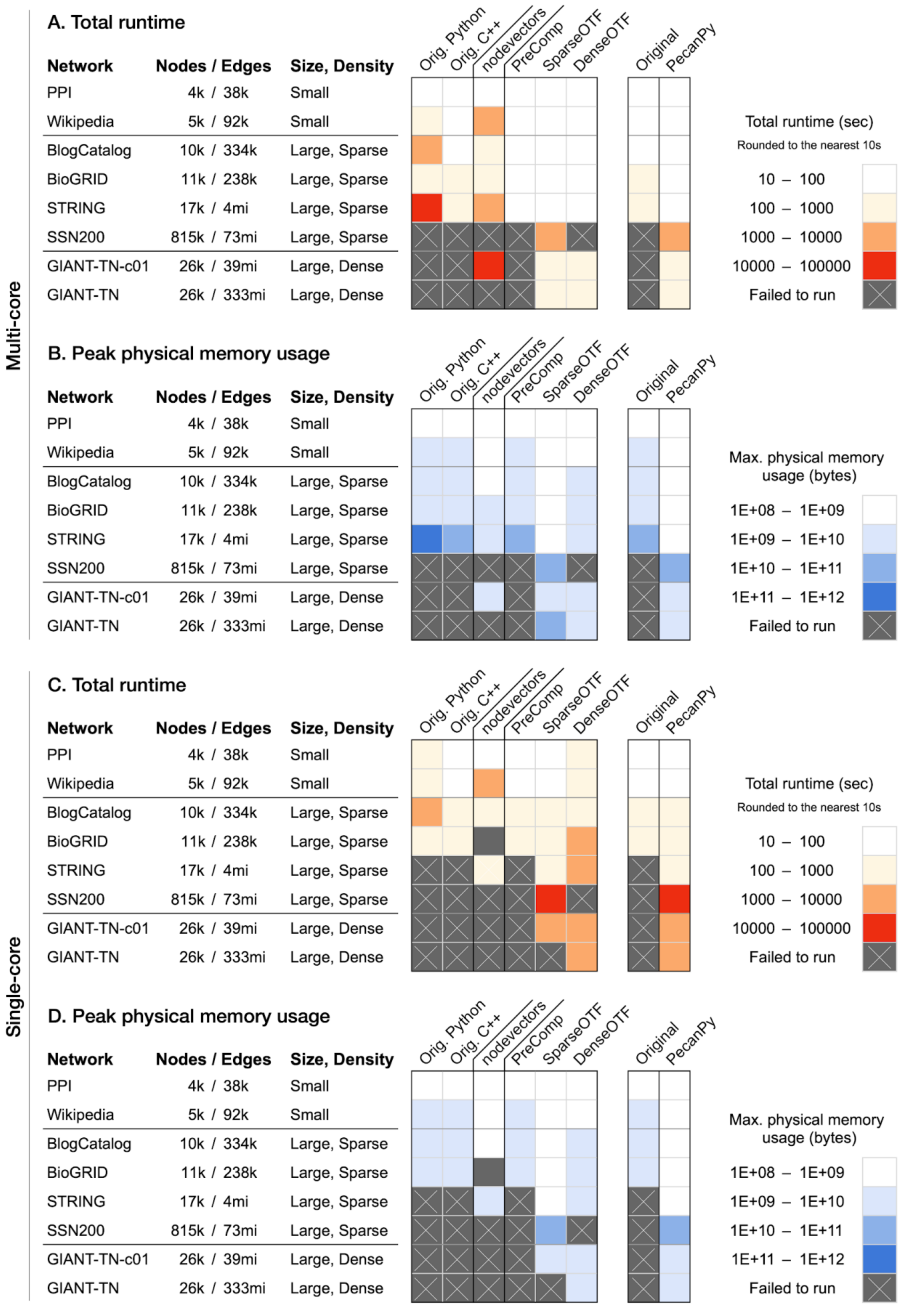


Figure B.9: **Summary of runtime and memory of PecanPy, nodevectors, and the original implementations of node2vec using multiple cores (A and B) and a single core (C and D).** The eight networks of varying sizes and densities are along the rows. The software implementations are along the columns. In each panel, the first heatmap (on the left) shows the performance of the original Python and C++ software and the nodevectors software along with the three modes of PecanPy (PreComp, SparseOTF, and DenseOTF). The adjacent 2-column heatmap (on the right) summarizes the performance of the original (best of Python and C++ versions) and PecanPy (best of PreComp, SparseOTF, and DenseOTF) implementations. Lighter colors correspond to lower runtime in panels A and C, and lower memory usage in panels B and D. Crossed grey indicates that the particular implementation (column) failed to run for a particular network (row).

APPENDIX C

OBNB

C.1 Data descriptions

C.1.1 Networks

BioGRID [240] (https://biogrid-downloads.nyc3.digitaloceanspaces.com/LICENSE.txt, MIT License) is a protein interaction network curated from primary experimental evidence from the biomedical literature, as well as evidence inferred from low- and high-throughput experiments.

BioPlex [106] (https://bioplex.hms.harvard.edu/explorer/help, Creative Commons Attribution-ShareAlike 4.0 International License) is a protein interaction network whose interactions are measured by affinity purification-mass spectrometry (AP-MS) analysis shared across two cell lines (HEK293T and HCT116). This shared interaction network encodes core complexes involving many essential proteins that are vital for the cell's survival.

ComPPIHumanInt [260] (https://academic.oup.com/nar/article/43/D1/D485/2435307, CC BY-NC 4.0 License) is a context-naive version of the cellular compartment-specific ComPPI [260] networks for humans, which are constructed by combining protein interactions from nine different PPI databases (including BioGRID).

ConsensusPathDB [117] (Free for academic use, see http://cpdb.molgen.mpg.de/ for more Licensing info) integrates protein interaction evidence (binary protein interaction, protein complex interaction, genetic interaction, metabolic, signaling, etc.) from 31 public databases, in addition to interactions curated from the literature.

FunCoup [204] (https://funcoup.org/help/, CC BY-SA 4.0 License) version 5 is a functional gene interaction network constructed by integrating a wide range of interaction evidence using a redundancy-weighted naive Bayes approach.

HIPPIE [5] (https://academic.oup.com/nar/article/45/D1/D408/2290937,CC BY-NC 4.0 License) integrates experimentally detected protein interactions from several public databases such as BioGRID.

HumanBaseTopGlobal [80] (https://humanbase.readthedocs.io/en/latest/, CC BY 4.0 License) is the tissue-naive version of the HumanBase tissue-specific gene interaction network collections, which are constructed by integrating hundreds of thousands of publicly available gene expression studies, protein–protein interactions and protein–DNA interactions via a Bayesian approach, calibrated against high-quality manually curated functional gene interactions.

HuMAP [62] (http://humap2.proteincomplexes.org/download, CC0 1.0 License) is a protein interaction network derived from over seven thousand protein complexes by integrating experimental evidence from public resources including AP-MS, large-scale biochemical fraction data, proximity labeling, and RNA hairpin pulldown data.

HumanNet [107] (https://staging2.inetbio.org/humannetv3/about.php, CC BY-SA 4.0 License) is a functional gene interaction network originally designed for disease studies. It contains interaction evidence from gene co-citation from the literature, gene co-expression, pathway, domain profile, genetic interaction, gene neighborhood, phylogenic profile, and other protein interaction data. All these interaction evidence are integrated using a Bayesian statistical framework, resulting in a single value for each pair of genes that indicate the odds ratio for their functional interaction.

HuRI [160] (http://www.interactome-atlas.org/download, CC BY-4.0 License) is a binary protein interaction network constructed via the yeast two-hybrid (Y2H), covering about 90% of the protein-coding genes in the human genome.

OmniPath [249] (See https://omnipathdb.org/info for Licenses collected for each database) integrates protein interactions, signaling and regulatory relationships from over 100 resources.

PCNet [100] (https://www.ndexbio.org/viewer/networks/f93f402c-86d4-11e7-a10d-0ac135e8bacf, CC BY-NC 4.0 License) is a protein interaction network constructed by requiring that an edge be supported by at least two out of the 21 selected protein interaction networks, such as BioGRID and ConsensusPathDB.

ProteomeHD [131] (https://www.ndexbio.org/viewer/networks/4cb4b0f3-83da-11e9-848d-0ac135e8bacf, CC BY-4.0 License) is the subnetwork of [131] containing top 0.5% strongest co-regulation signal between pairs of proteins. The co-regulation is measured by proteins' response to a total of 294

biological perturbations via isotope-labeling mass spectrometry.

SIGNOR [201] (https://signor.uniroma2.it/about/, CC BY-4.0 License) contains manually curated causal signaling relationships between proteins and other biochemical molecules, such as transcriptional activations and phosphorylation.

STRING [246] (https://string-db.org/cgi/access?footer_active_subpage=licensing, CC BY-4.0 License) is a functional interaction network constructed by integrating seven types of gene interaction evidence via a probabilistic approach that calibrates against the KEGG [118] pathway database. The seven evidence types include conserved neighborhood, gene co-occurrence across species, gene fusion events, gene co-expression, other databases, and text-mined interactions.

C.1.2 Annotations and Ontologies

Gene Ontology [14] (http://geneontology.org/docs/go-citation-policy/, CC BY-4.0 License) is a structured and standardized system that provides a comprehensive vocabulary to describe the molecular functions (GOMF), biological processes (GOBP), and cellular components (GOCC) associated with genes across different organisms. It aims to unify the representation of biological knowledge and enable effective analysis and interpretation of genomic data.

Mondo Disease Ontology [254] (https://github.com/monarch-initiative/mondo/blob/master/LICENSE, CC BY-4.0 License) is a unifying resource that integrates disease, genotype, and phenotype knowledge across diverse resources, providing a standardized knowledge graph with controlled vocabularies for diseases and phenotypes.

DisGeNET disease gene annotations [206] (https://academic.oup.com/nar/article/45/D1/D833/2290909, CC BY-NC 4.0 License) is a disease gene annotation database that contains a wide range of disease-gene association evidence, including curated annotations, high-throughput experiments and other inferred annotations, animal models, and literature text-mined annotation mostly from BEFREE. By default, we only use the curated and inferred annotations.

DISEASES disease gene annotations [81] (https://diseases.jensenlab.org/Downloads, CC BY License) is another disease gene annotation database that has a weekly update schedule for extracting disease gene annotations via text-mining from the fast-growing literature. The text-mined approach

uses full text instead of only using the titles and abstracts. Other disease gene annotations from experimental data and other databases are also available.

C.1.3 Archived data

In addition to downloading and processing the data directly from the original sources, we also provide archived versions of the data preprocessed by running the default OBNB processing pipelines. The archived data is versioned with DOI's and can be found on Zenodo under the record https://zenodo.org/record/8045270

C.2 Additional results

Table C.1: **Ablation study on different initial node feature constructions for GCN and GAT.** Reported values are average test APOP scores aggregated over five random seeds. The best score within a group (network x label x model) is **bolded**.

Network	Model	Feature	DISEASES	DisGeNET	GOBP
BioGRID	GCN	Constant	0.373 ± 0.018	0.451 ± 0.210	0.557 ± 0.144
		RandomNormal	1.329 ± 0.043	0.867 ± 0.017	2.121 ± 0.114
		OneHotLogDeg	1.077 ± 0.057	0.827 ± 0.050	2.011 ± 0.148
		RandomWalkDiag	0.844 ± 0.103	0.702 ± 0.064	1.358 ± 0.044
		Adj	**1.442 ± 0.120**	0.899 ± 0.108	2.148 ± 0.045
		RandProjGaussian	1.288 ± 0.041	0.828 ± 0.034	2.278 ± 0.084
		RandProjSparse	1.264 ± 0.047	0.749 ± 0.057	2.306 ± 0.088
		SVD	1.259 ± 0.076	0.808 ± 0.030	2.318 ± 0.080
		LapEigMap	1.415 ± 0.030	0.963 ± 0.045	2.378 ± 0.098
		LINE1	1.258 ± 0.041	0.798 ± 0.023	2.376 ± 0.081
		LINE2	1.312 ± 0.027	0.857 ± 0.083	2.416 ± 0.048
		Node2vec	1.392 ± 0.051	**0.965 ± 0.055**	**2.487 ± 0.112**
		Walklets	1.366 ± 0.044	0.886 ± 0.079	2.438 ± 0.052
		Embedding	1.348 ± 0.073	0.788 ± 0.060	2.147 ± 0.113
		AdjEmbBag	1.338 ± 0.071	0.798 ± 0.080	2.031 ± 0.089
	GAT	Constant	0.399 ± 0.039	0.362 ± 0.071	0.398 ± 0.071
		RandomNormal	1.290 ± 0.126	0.755 ± 0.140	2.268 ± 0.047
		OneHotLogDeg	0.946 ± 0.289	0.803 ± 0.066	2.181 ± 0.080
		RandomWalkDiag	0.873 ± 0.115	0.554 ± 0.160	1.702 ± 0.068
		Adj	1.290 ± 0.320	0.594 ± 0.154	1.972 ± 0.230
		RandProjGaussian	1.357 ± 0.073	0.897 ± 0.070	2.402 ± 0.081
		RandProjSparse	1.351 ± 0.091	0.842 ± 0.063	2.461 ± 0.062
		SVD	1.019 ± 0.199	0.700 ± 0.081	2.384 ± 0.058
		LapEigMap	1.423 ± 0.072	**0.914 ± 0.103**	2.541 ± 0.042
		LINE1	**1.480 ± 0.073**	0.874 ± 0.071	2.486 ± 0.040
		LINE2	1.454 ± 0.043	0.888 ± 0.057	2.463 ± 0.138
		Node2vec	1.416 ± 0.126	0.877 ± 0.032	**2.585 ± 0.052**

Table C.1 (cont'd).

Network	Model	Feature	DISEASES	DisGeNET	GOBP
		Walklets	1.464 ± 0.072	0.852 ± 0.073	2.392 ± 0.170
		Embedding	1.375 ± 0.092	0.834 ± 0.056	2.279 ± 0.197
		AdjEmbBag	0.645 ± 0.049	0.525 ± 0.041	1.242 ± 0.062
HumanNet	GCN	Constant	0.385 ± 0.018	0.898 ± 0.873	0.610 ± 0.037
		RandomNormal	3.006 ± 0.112	2.511 ± 0.031	3.302 ± 0.104
		OneHotLogDeg	2.873 ± 0.134	2.552 ± 0.059	3.574 ± 0.116
		RandomWalkDiag	2.056 ± 0.102	1.672 ± 0.171	3.001 ± 0.149
		Adj	**3.562 ± 0.049**	2.788 ± 0.096	3.715 ± 0.077
		RandProjGaussian	3.346 ± 0.096	2.768 ± 0.036	3.654 ± 0.054
		RandProjSparse	3.341 ± 0.085	2.784 ± 0.021	3.759 ± 0.039
		SVD	3.211 ± 0.062	2.723 ± 0.064	3.780 ± 0.075
		LapEigMap	3.219 ± 0.077	2.676 ± 0.072	3.735 ± 0.068
		LINE1	2.883 ± 0.097	2.560 ± 0.049	3.700 ± 0.078
		LINE2	2.918 ± 0.070	2.563 ± 0.082	3.656 ± 0.038
		Node2vec	3.359 ± 0.070	**2.798 ± 0.024**	3.816 ± 0.039
		Walklets	3.464 ± 0.066	2.762 ± 0.053	**3.842 ± 0.052**
		Embedding	3.389 ± 0.051	2.637 ± 0.032	3.538 ± 0.040
		AdjEmbBag	3.499 ± 0.032	2.700 ± 0.066	3.482 ± 0.048
	GAT	Constant	0.280 ± 0.043	0.449 ± 0.047	0.333 ± 0.035
		RandomNormal	3.442 ± 0.331	2.758 ± 0.204	3.535 ± 0.126
		OneHotLogDeg	3.052 ± 0.312	2.605 ± 0.074	3.661 ± 0.088
		RandomWalkDiag	2.623 ± 0.862	2.340 ± 0.285	3.409 ± 0.059
		Adj	**3.791 ± 0.071**	2.943 ± 0.053	3.770 ± 0.029
		RandProjGaussian	3.390 ± 0.105	2.749 ± 0.134	3.803 ± 0.103
		RandProjSparse	3.477 ± 0.135	2.811 ± 0.126	3.773 ± 0.085
		SVD	3.691 ± 0.080	2.653 ± 0.101	3.792 ± 0.076
		LapEigMap	3.438 ± 0.269	2.837 ± 0.180	3.907 ± 0.034
		LINE1	3.630 ± 0.058	2.794 ± 0.163	3.857 ± 0.085
		LINE2	3.430 ± 0.133	2.867 ± 0.051	3.779 ± 0.123

Table C.1 (cont'd).

Network	Model	Feature	DISEASES	DisGeNET	GOBP
		Node2vec	3.662 ± 0.036	**2.975 ± 0.027**	**3.947 ± 0.090**
		Walklets	3.483 ± 0.103	2.887 ± 0.092	3.885 ± 0.112
		Embedding	3.539 ± 0.227	2.797 ± 0.087	3.738 ± 0.090
		AdjEmbBag	3.592 ± 0.086	2.802 ± 0.065	3.692 ± 0.075
ComPPIHumanInt	GCN	Constant	0.263 ± 0.026	0.307 ± 0.014	0.440 ± 0.039
		RandomNormal	1.544 ± 0.024	1.075 ± 0.058	2.411 ± 0.064
		OneHotLogDeg	1.394 ± 0.077	0.964 ± 0.043	2.244 ± 0.072
		RandomWalkDiag	0.946 ± 0.109	0.759 ± 0.053	1.652 ± 0.081
		Adj	1.471 ± 0.024	1.037 ± 0.074	2.320 ± 0.049
		RandProjGaussian	1.527 ± 0.067	1.069 ± 0.071	2.649 ± 0.030
		RandProjSparse	1.558 ± 0.069	1.030 ± 0.046	2.593 ± 0.058
		SVD	1.459 ± 0.026	**1.112 ± 0.056**	2.622 ± 0.091
		LapEigMap	**1.626 ± 0.037**	1.110 ± 0.061	2.484 ± 0.058
		LINE1	1.536 ± 0.103	1.057 ± 0.049	2.540 ± 0.049
		LINE2	1.606 ± 0.062	1.060 ± 0.041	2.588 ± 0.016
		Node2vec	1.546 ± 0.080	1.072 ± 0.047	**2.744 ± 0.067**
		Walklets	1.564 ± 0.053	1.086 ± 0.054	2.593 ± 0.034
		Embedding	1.356 ± 0.040	0.863 ± 0.068	2.190 ± 0.065
		AdjEmbBag	1.390 ± 0.042	0.929 ± 0.071	2.366 ± 0.080
	GAT	Constant	0.323 ± 0.052	0.327 ± 0.056	0.366 ± 0.047
		RandomNormal	1.386 ± 0.072	1.171 ± 0.107	2.584 ± 0.079
		OneHotLogDeg	1.287 ± 0.148	0.756 ± 0.272	2.197 ± 0.222
		RandomWalkDiag	1.197 ± 0.097	0.811 ± 0.044	1.933 ± 0.211
		Adj	1.348 ± 0.198	0.791 ± 0.095	2.087 ± 0.035
		RandProjGaussian	1.565 ± 0.027	1.074 ± 0.065	2.706 ± 0.107
		RandProjSparse	1.562 ± 0.067	1.107 ± 0.029	2.728 ± 0.068
		SVD	1.382 ± 0.076	1.050 ± 0.054	2.730 ± 0.058
		LapEigMap	1.622 ± 0.044	1.192 ± 0.043	2.742 ± 0.072
		LINE1	**1.633 ± 0.055**	1.134 ± 0.071	2.625 ± 0.085

Table C.1 (cont'd).

Network	Model	Feature	DISEASES	DisGeNET	GOBP
		LINE2	1.569 ± 0.034	1.112 ± 0.067	2.656 ± 0.071
		Node2vec	1.572 ± 0.064	1.169 ± 0.079	2.766 ± 0.064
		Walklets	1.623 ± 0.028	**1.188 ± 0.070**	**2.789 ± 0.043**
		Embedding	1.544 ± 0.036	1.011 ± 0.082	2.487 ± 0.072
		AdjEmbBag	0.887 ± 0.078	0.611 ± 0.061	1.724 ± 0.066
BioPlex	GCN	Constant	0.342 ± 0.032	0.356 ± 0.052	0.488 ± 0.083
		RandomNormal	**1.304 ± 0.016**	0.868 ± 0.030	2.445 ± 0.064
		OneHotLogDeg	1.225 ± 0.065	**0.909 ± 0.050**	2.500 ± 0.044
		RandomWalkDiag	1.242 ± 0.043	0.835 ± 0.038	2.461 ± 0.138
		Adj	1.182 ± 0.067	0.817 ± 0.023	**2.613 ± 0.078**
		RandProjGaussian	1.218 ± 0.030	0.855 ± 0.066	2.583 ± 0.061
		RandProjSparse	1.256 ± 0.085	0.869 ± 0.017	2.563 ± 0.053
		SVD	1.273 ± 0.055	0.707 ± 0.046	2.513 ± 0.036
		LapEigMap	1.206 ± 0.038	0.785 ± 0.057	2.382 ± 0.026
		LINE1	1.234 ± 0.028	0.790 ± 0.033	2.604 ± 0.073
		LINE2	1.242 ± 0.059	0.789 ± 0.053	2.544 ± 0.060
		Node2vec	1.206 ± 0.042	0.784 ± 0.060	2.549 ± 0.074
		Walklets	1.215 ± 0.060	0.817 ± 0.045	2.558 ± 0.065
		Embedding	1.157 ± 0.024	0.772 ± 0.066	2.582 ± 0.034
		AdjEmbBag	1.240 ± 0.038	0.770 ± 0.021	2.582 ± 0.100
	GAT	Constant	0.275 ± 0.063	0.275 ± 0.146	0.569 ± 0.143
		RandomNormal	1.256 ± 0.070	0.884 ± 0.047	2.489 ± 0.063
		OneHotLogDeg	1.089 ± 0.100	0.793 ± 0.044	2.479 ± 0.098
		RandomWalkDiag	1.041 ± 0.082	0.788 ± 0.090	2.430 ± 0.085
		Adj	1.157 ± 0.029	0.811 ± 0.042	2.582 ± 0.069
		RandProjGaussian	1.222 ± 0.064	**0.924 ± 0.041**	2.588 ± 0.094
		RandProjSparse	1.213 ± 0.041	0.882 ± 0.063	**2.632 ± 0.009**
		SVD	1.139 ± 0.089	0.721 ± 0.073	2.539 ± 0.045
		LapEigMap	1.204 ± 0.062	0.832 ± 0.058	2.371 ± 0.064

Table C.1 (cont'd).

Network	Model	Feature	DISEASES	DisGeNET	GOBP
		LINE1	1.201 ± 0.060	0.802 ± 0.035	2.453 ± 0.074
		LINE2	**1.242 ± 0.034**	0.850 ± 0.035	2.478 ± 0.057
		Node2vec	1.229 ± 0.039	0.768 ± 0.042	2.369 ± 0.061
		Walklets	1.196 ± 0.027	0.855 ± 0.073	2.531 ± 0.059
		Embedding	1.159 ± 0.044	0.746 ± 0.088	2.493 ± 0.071
		AdjEmbBag	1.183 ± 0.043	0.784 ± 0.050	2.522 ± 0.056
HuRI	GCN	Constant	0.346 ± 0.015	0.529 ± 0.033	0.384 ± 0.055
		RandomNormal	0.504 ± 0.108	0.676 ± 0.080	0.956 ± 0.125
		OneHotLogDeg	0.552 ± 0.080	0.579 ± 0.076	1.047 ± 0.102
		RandomWalkDiag	0.549 ± 0.107	0.484 ± 0.045	1.027 ± 0.129
		Adj	0.587 ± 0.023	0.631 ± 0.032	0.942 ± 0.075
		RandProjGaussian	0.676 ± 0.070	0.673 ± 0.046	1.016 ± 0.186
		RandProjSparse	**0.686 ± 0.087**	0.689 ± 0.055	1.076 ± 0.101
		SVD	0.628 ± 0.056	0.667 ± 0.010	1.005 ± 0.049
		LapEigMap	0.581 ± 0.014	0.526 ± 0.045	0.997 ± 0.085
		LINE1	0.658 ± 0.069	0.690 ± 0.054	1.053 ± 0.089
		LINE2	0.632 ± 0.125	**0.741 ± 0.074**	1.106 ± 0.084
		Node2vec	0.566 ± 0.061	0.738 ± 0.053	1.126 ± 0.114
		Walklets	0.596 ± 0.078	0.681 ± 0.032	**1.179 ± 0.056**
		Embedding	0.572 ± 0.070	0.606 ± 0.034	0.888 ± 0.109
		AdjEmbBag	0.652 ± 0.039	0.660 ± 0.030	1.059 ± 0.076
	GAT	Constant	0.305 ± 0.078	0.393 ± 0.077	0.345 ± 0.060
		RandomNormal	0.524 ± 0.159	0.541 ± 0.131	0.968 ± 0.114
		OneHotLogDeg	0.465 ± 0.060	0.510 ± 0.053	0.872 ± 0.189
		RandomWalkDiag	0.473 ± 0.057	0.445 ± 0.129	0.972 ± 0.057
		Adj	**0.683 ± 0.109**	0.528 ± 0.053	0.956 ± 0.157
		RandProjGaussian	0.592 ± 0.086	0.530 ± 0.071	1.028 ± 0.139
		RandProjSparse	0.603 ± 0.115	0.456 ± 0.085	1.017 ± 0.064
		SVD	0.522 ± 0.099	0.412 ± 0.029	1.021 ± 0.099

Table C.1 (cont'd).

Network	Model	Feature	DISEASES	DisGeNET	GOBP
		LapEigMap	0.599 ± 0.082	0.557 ± 0.048	1.071 ± 0.080
		LINE1	0.585 ± 0.068	0.652 ± 0.043	1.101 ± 0.033
		LINE2	0.634 ± 0.067	**0.743 ± 0.053**	1.095 ± 0.054
		Node2vec	0.577 ± 0.017	0.657 ± 0.067	**1.116 ± 0.129**
		Walklets	0.630 ± 0.065	0.602 ± 0.032	1.085 ± 0.085
		Embedding	0.647 ± 0.087	0.601 ± 0.020	0.941 ± 0.081
		AdjEmbBag	0.603 ± 0.056	0.581 ± 0.040	0.902 ± 0.059
OmniPath	GCN	Constant	0.415 ± 0.083	0.416 ± 0.044	0.451 ± 0.046
		RandomNormal	1.417 ± 0.042	0.910 ± 0.126	1.798 ± 0.073
		OneHotLogDeg	1.195 ± 0.041	0.930 ± 0.046	1.753 ± 0.090
		RandomWalkDiag	0.847 ± 0.038	0.762 ± 0.045	1.427 ± 0.056
		Adj	**1.444 ± 0.019**	**1.118 ± 0.062**	2.050 ± 0.049
		RandProjGaussian	1.381 ± 0.089	1.014 ± 0.053	1.935 ± 0.075
		RandProjSparse	1.338 ± 0.073	1.066 ± 0.060	1.935 ± 0.020
		SVD	1.281 ± 0.052	0.849 ± 0.024	1.884 ± 0.048
		LapEigMap	1.408 ± 0.030	1.108 ± 0.092	**2.120 ± 0.040**
		LINE1	1.366 ± 0.075	0.946 ± 0.049	2.027 ± 0.042
		LINE2	1.338 ± 0.065	0.963 ± 0.054	1.938 ± 0.096
		Node2vec	1.384 ± 0.046	1.024 ± 0.020	1.982 ± 0.054
		Walklets	1.433 ± 0.050	1.036 ± 0.041	2.101 ± 0.033
		Embedding	1.329 ± 0.062	0.831 ± 0.069	1.504 ± 0.048
		AdjEmbBag	1.373 ± 0.042	1.085 ± 0.024	1.917 ± 0.054
	GAT	Constant	0.353 ± 0.044	0.343 ± 0.077	0.381 ± 0.053
		RandomNormal	1.374 ± 0.041	1.034 ± 0.091	1.945 ± 0.104
		OneHotLogDeg	0.775 ± 0.142	0.773 ± 0.135	1.685 ± 0.110
		RandomWalkDiag	0.754 ± 0.171	0.657 ± 0.159	1.580 ± 0.181
		Adj	**1.257 ± 0.179**	**1.184 ± 0.068**	1.865 ± 0.095
		RandProjGaussian	1.146 ± 0.117	1.032 ± 0.041	2.079 ± 0.126
		RandProjSparse	1.216 ± 0.075	0.916 ± 0.121	2.023 ± 0.272

Table C.1 (cont'd).

Network	Model	Feature	DISEASES	DisGeNET	GOBP
		SVD	1.032 ± 0.125	0.889 ± 0.061	1.916 ± 0.159
		LapEigMap	1.344 ± 0.052	1.103 ± 0.087	2.105 ± 0.061
		LINE1	1.263 ± 0.082	0.936 ± 0.035	2.077 ± 0.108
		LINE2	1.376 ± 0.041	1.031 ± 0.104	2.026 ± 0.081
		Node2vec	1.317 ± 0.040	1.014 ± 0.090	2.096 ± 0.063
		Walklets	1.248 ± 0.099	1.022 ± 0.059	**2.190 ± 0.052**
		Embedding	1.371 ± 0.032	0.917 ± 0.103	1.738 ± 0.023
		AdjEmbBag	0.752 ± 0.076	0.724 ± 0.140	1.399 ± 0.171
ProteomeHD	GCN	Constant	0.289 ± 0.112	0.489 ± 0.021	0.729 ± 0.481
		RandomNormal	0.695 ± 0.128	0.583 ± 0.060	1.267 ± 0.074
		OneHotLogDeg	0.698 ± 0.060	0.692 ± 0.018	1.413 ± 0.078
		RandomWalkDiag	0.645 ± 0.072	0.665 ± 0.053	1.247 ± 0.049
		Adj	0.656 ± 0.060	0.588 ± 0.067	1.386 ± 0.054
		RandProjGaussian	0.617 ± 0.096	**0.714 ± 0.070**	1.412 ± 0.089
		RandProjSparse	0.685 ± 0.074	0.669 ± 0.084	1.461 ± 0.112
		SVD	**0.809 ± 0.073**	0.575 ± 0.081	1.474 ± 0.073
		LapEigMap	0.715 ± 0.017	0.535 ± 0.090	1.272 ± 0.035
		LINE1	0.588 ± 0.061	0.594 ± 0.042	1.316 ± 0.082
		LINE2	0.646 ± 0.071	0.630 ± 0.034	1.323 ± 0.073
		Node2vec	0.682 ± 0.049	0.621 ± 0.071	1.379 ± 0.102
		Walklets	0.635 ± 0.093	0.649 ± 0.082	1.400 ± 0.141
		Embedding	0.687 ± 0.074	0.539 ± 0.024	1.293 ± 0.158
		AdjEmbBag	0.600 ± 0.096	0.621 ± 0.123	**1.480 ± 0.083**
	GAT	Constant	0.375 ± 0.158	0.473 ± 0.082	0.675 ± 0.233
		RandomNormal	0.655 ± 0.115	0.657 ± 0.129	1.458 ± 0.049
		OneHotLogDeg	0.705 ± 0.115	0.687 ± 0.063	1.339 ± 0.199
		RandomWalkDiag	0.700 ± 0.065	0.691 ± 0.077	1.196 ± 0.277
		Adj	0.731 ± 0.149	0.625 ± 0.068	1.487 ± 0.080
		RandProjGaussian	0.808 ± 0.118	0.701 ± 0.036	1.491 ± 0.102

Table C.1 (cont'd).

Network	Model	Feature	DISEASES	DisGeNET	GOBP
		RandProjSparse	0.725 ± 0.130	0.710 ± 0.061	1.502 ± 0.080
		SVD	0.792 ± 0.137	**0.828 ± 0.039**	**1.556 ± 0.057**
		LapEigMap	0.678 ± 0.041	0.534 ± 0.135	1.394 ± 0.079
		LINE1	0.638 ± 0.159	0.627 ± 0.094	1.338 ± 0.120
		LINE2	0.579 ± 0.079	0.599 ± 0.115	1.433 ± 0.076
		Node2vec	**0.839 ± 0.100**	0.702 ± 0.075	1.446 ± 0.089
		Walklets	0.644 ± 0.106	0.653 ± 0.098	1.544 ± 0.064
		Embedding	0.650 ± 0.141	0.659 ± 0.100	1.460 ± 0.041
		AdjEmbBag	0.685 ± 0.064	0.666 ± 0.079	1.408 ± 0.052
SIGNOR	GCN	Constant	0.388 ± 0.066	0.425 ± 0.157	0.500 ± 0.148
		RandomNormal	1.478 ± 0.050	1.349 ± 0.080	1.571 ± 0.056
		OneHotLogDeg	1.427 ± 0.032	1.183 ± 0.087	1.754 ± 0.066
		RandomWalkDiag	1.268 ± 0.011	1.022 ± 0.043	1.601 ± 0.127
		Adj	1.461 ± 0.042	1.306 ± 0.058	1.575 ± 0.087
		RandProjGaussian	1.488 ± 0.046	1.253 ± 0.083	1.764 ± 0.062
		RandProjSparse	1.535 ± 0.067	1.369 ± 0.107	1.780 ± 0.071
		SVD	1.396 ± 0.093	1.279 ± 0.071	1.788 ± 0.066
		LapEigMap	**1.597 ± 0.035**	1.345 ± 0.042	1.746 ± 0.057
		LINE1	1.546 ± 0.068	1.354 ± 0.049	1.813 ± 0.101
		LINE2	1.506 ± 0.084	**1.390 ± 0.103**	1.708 ± 0.076
		Node2vec	1.566 ± 0.071	1.345 ± 0.068	**1.887 ± 0.115**
		Walklets	1.562 ± 0.076	1.333 ± 0.088	1.732 ± 0.049
		Embedding	1.396 ± 0.035	1.200 ± 0.076	1.498 ± 0.064
		AdjEmbBag	1.496 ± 0.071	1.229 ± 0.130	1.665 ± 0.100
	GAT	Constant	0.279 ± 0.038	0.242 ± 0.051	0.488 ± 0.176
		RandomNormal	1.603 ± 0.049	1.281 ± 0.060	1.707 ± 0.050
		OneHotLogDeg	1.180 ± 0.069	1.009 ± 0.087	1.576 ± 0.047
		RandomWalkDiag	1.185 ± 0.115	0.935 ± 0.151	1.759 ± 0.115
		Adj	1.538 ± 0.104	1.251 ± 0.023	1.529 ± 0.053

Table C.1 (cont'd).

Network	Model	Feature	DISEASES	DisGeNET	GOBP
		RandProjGaussian	1.531 ± 0.085	1.382 ± 0.107	1.751 ± 0.037
		RandProjSparse	1.508 ± 0.033	1.290 ± 0.073	1.715 ± 0.090
		SVD	1.529 ± 0.048	1.178 ± 0.102	1.718 ± 0.017
		LapEigMap	1.578 ± 0.053	1.326 ± 0.059	1.749 ± 0.081
		LINE1	1.583 ± 0.036	1.282 ± 0.052	1.844 ± 0.071
		LINE2	**1.630 ± 0.038**	1.325 ± 0.083	**1.863 ± 0.048**
		Node2vec	1.519 ± 0.066	1.287 ± 0.047	1.807 ± 0.033
		Walklets	1.579 ± 0.045	**1.411 ± 0.025**	1.762 ± 0.052
		Embedding	1.474 ± 0.095	1.254 ± 0.051	1.695 ± 0.053
		AdjEmbBag	1.457 ± 0.198	1.154 ± 0.096	1.405 ± 0.156

Table C.2: **Baseline performance reference**. Reported values are average test APOP scores aggregated over five random seeds. The best performance achieved by (1) GNNs or (2) logistic regression and label propagation are **bolded** for each dataset (network × label). The best performance across the two methods groups is additionally colored by green.

Network	Model	Feature	DISEASES	DisGeNET	GOBP
BioGRID	GCN	OneHotLogDeg	**1.111 ± 0.043**	**0.773 ± 0.035**	**2.022 ± 0.100**
	SAGE	OneHotLogDeg	0.840 ± 0.105	0.665 ± 0.071	1.510 ± 0.114
	GIN	OneHotLogDeg	0.720 ± 0.061	0.700 ± 0.050	1.443 ± 0.120
	GAT	OneHotLogDeg	0.946 ± 0.289	0.552 ± 0.111	1.592 ± 0.408
	GatedGCN	OneHotLogDeg	1.014 ± 0.065	0.753 ± 0.047	1.924 ± 0.055
	LabelProp	–	1.210 ± 0.000	**0.931 ± 0.000**	1.885 ± 0.000
	LogReg	Adj	**1.328 ± 0.001**	0.743 ± 0.000	**2.528 ± 0.000**
	LogReg	LapEigMap	1.288 ± 0.001	0.864 ± 0.002	2.149 ± 0.000
	LogReg	SVD	0.881 ± 0.010	0.724 ± 0.003	2.088 ± 0.003
	LogReg	LINE1	1.117 ± 0.024	0.722 ± 0.004	2.264 ± 0.019
	LogReg	LINE2	1.132 ± 0.021	0.825 ± 0.007	2.351 ± 0.017
	LogReg	Node2vec	1.116 ± 0.054	0.836 ± 0.029	2.571 ± 0.015
	LogReg	Walklets	1.023 ± 0.052	0.786 ± 0.046	2.189 ± 0.020
HumanNet	GCN	OneHotLogDeg	2.902 ± 0.050	2.452 ± 0.107	3.524 ± 0.061
	SAGE	OneHotLogDeg	2.850 ± 0.107	2.356 ± 0.093	3.326 ± 0.067
	GIN	OneHotLogDeg	2.378 ± 0.126	2.019 ± 0.113	3.151 ± 0.017
	GAT	OneHotLogDeg	**3.052 ± 0.312**	**2.547 ± 0.207**	**3.571 ± 0.159**
	GatedGCN	OneHotLogDeg	3.004 ± 0.132	2.327 ± 0.026	3.486 ± 0.047
	LabelProp	–	3.728 ± 0.000	**3.059 ± 0.000**	3.806 ± 0.000
	LogReg	Adj	**3.812 ± 0.000**	3.053 ± 0.000	3.964 ± 0.006
	LogReg	LapEigMap	2.737 ± 0.003	2.301 ± 0.000	3.778 ± 0.001
	LogReg	SVD	2.785 ± 0.002	2.412 ± 0.004	3.618 ± 0.001
	LogReg	LINE1	2.178 ± 0.010	1.632 ± 0.005	3.348 ± 0.011
	LogReg	LINE2	2.270 ± 0.016	1.679 ± 0.005	3.485 ± 0.004
	LogReg	Node2vec	3.316 ± 0.020	2.433 ± 0.029	**4.036 ± 0.019**
	LogReg	Walklets	2.670 ± 0.069	2.050 ± 0.115	3.081 ± 0.054
ComPPIHumanInt	GCN	OneHotLogDeg	**1.359 ± 0.094**	**0.993 ± 0.042**	**2.251 ± 0.064**

Table C.2 (cont'd).

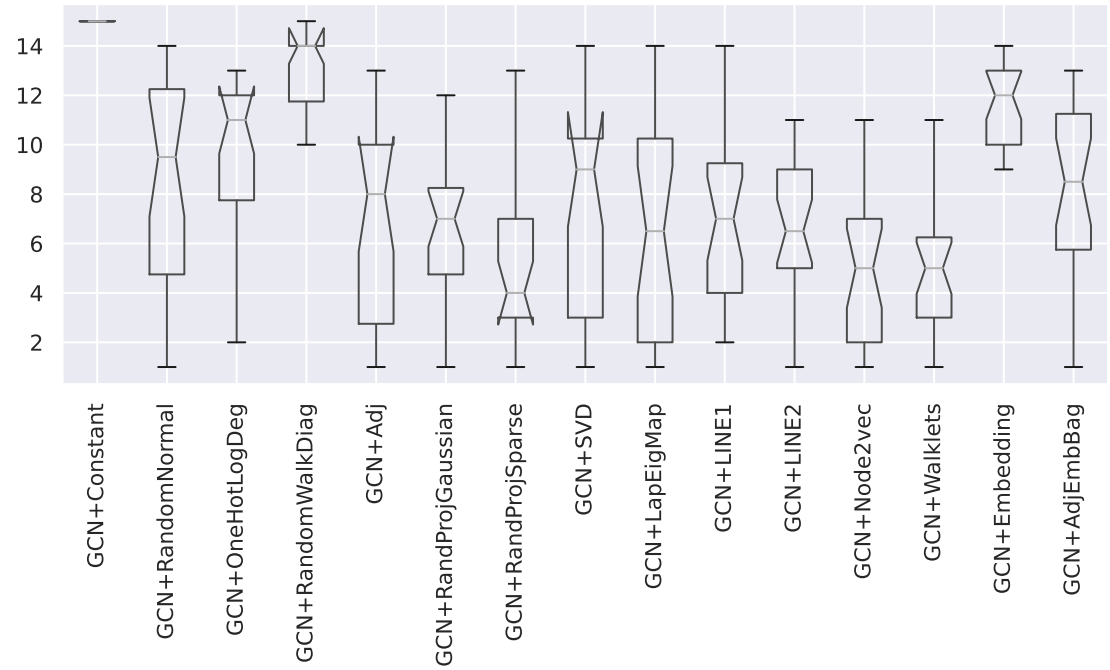
Network	Model	Feature	DISEASES	DisGeNET	GOBP
	SAGE	OneHotLogDeg	1.101 ± 0.063	0.724 ± 0.076	1.865 ± 0.091
	GIN	OneHotLogDeg	0.963 ± 0.057	0.777 ± 0.057	1.621 ± 0.077
	GAT	OneHotLogDeg	1.287 ± 0.148	0.756 ± 0.272	2.197 ± 0.222
	GatedGCN	OneHotLogDeg	1.166 ± 0.045	0.861 ± 0.050	2.044 ± 0.094
	LabelProp	–	1.352 ± 0.000	**1.106 ± 0.000**	2.076 ± 0.000
	LogReg	Adj	**1.431 ± 0.000**	1.016 ± 0.000	2.707 ± 0.002
	LogReg	LapEigMap	1.257 ± 0.001	1.045 ± 0.005	2.177 ± 0.002
	LogReg	SVD	0.888 ± 0.003	0.702 ± 0.001	1.999 ± 0.002
	LogReg	LINE1	1.135 ± 0.023	0.905 ± 0.014	2.438 ± 0.019
	LogReg	LINE2	1.185 ± 0.021	0.895 ± 0.012	2.495 ± 0.024
	LogReg	Node2vec	1.341 ± 0.034	1.074 ± 0.005	**2.806 ± 0.049**
	LogReg	Walklets	1.073 ± 0.053	0.890 ± 0.032	2.109 ± 0.104
BioPlex	GCN	OneHotLogDeg	**1.277 ± 0.034**	**0.895 ± 0.046**	**2.535 ± 0.065**
	SAGE	OneHotLogDeg	1.118 ± 0.043	0.787 ± 0.049	2.215 ± 0.084
	GIN	OneHotLogDeg	1.182 ± 0.083	0.822 ± 0.086	2.360 ± 0.054
	GAT	OneHotLogDeg	1.089 ± 0.100	0.793 ± 0.044	2.479 ± 0.098
	GatedGCN	OneHotLogDeg	0.970 ± 0.049	0.723 ± 0.052	2.303 ± 0.114
	LabelProp	–	0.964 ± 0.000	0.556 ± 0.000	2.174 ± 0.000
	LogReg	Adj	1.087 ± 0.003	0.838 ± 0.011	2.467 ± 0.001
	LogReg	LapEigMap	**1.147 ± 0.005**	**0.903 ± 0.019**	2.298 ± 0.017
	LogReg	SVD	0.824 ± 0.006	0.588 ± 0.001	2.170 ± 0.022
	LogReg	LINE1	0.914 ± 0.062	0.653 ± 0.011	**2.475 ± 0.042**
	LogReg	LINE2	0.843 ± 0.050	0.627 ± 0.015	2.321 ± 0.015
	LogReg	Node2vec	0.816 ± 0.073	0.639 ± 0.053	2.194 ± 0.039
	LogReg	Walklets	0.742 ± 0.067	0.688 ± 0.053	1.822 ± 0.046
HuRI	GCN	OneHotLogDeg	0.529 ± 0.075	**0.625 ± 0.089**	**1.040 ± 0.049**
	SAGE	OneHotLogDeg	0.491 ± 0.083	0.541 ± 0.029	0.937 ± 0.048
	GIN	OneHotLogDeg	0.591 ± 0.089	0.477 ± 0.052	1.008 ± 0.113
	GAT	OneHotLogDeg	0.465 ± 0.060	0.510 ± 0.053	0.872 ± 0.189

Table C.2 (cont'd).

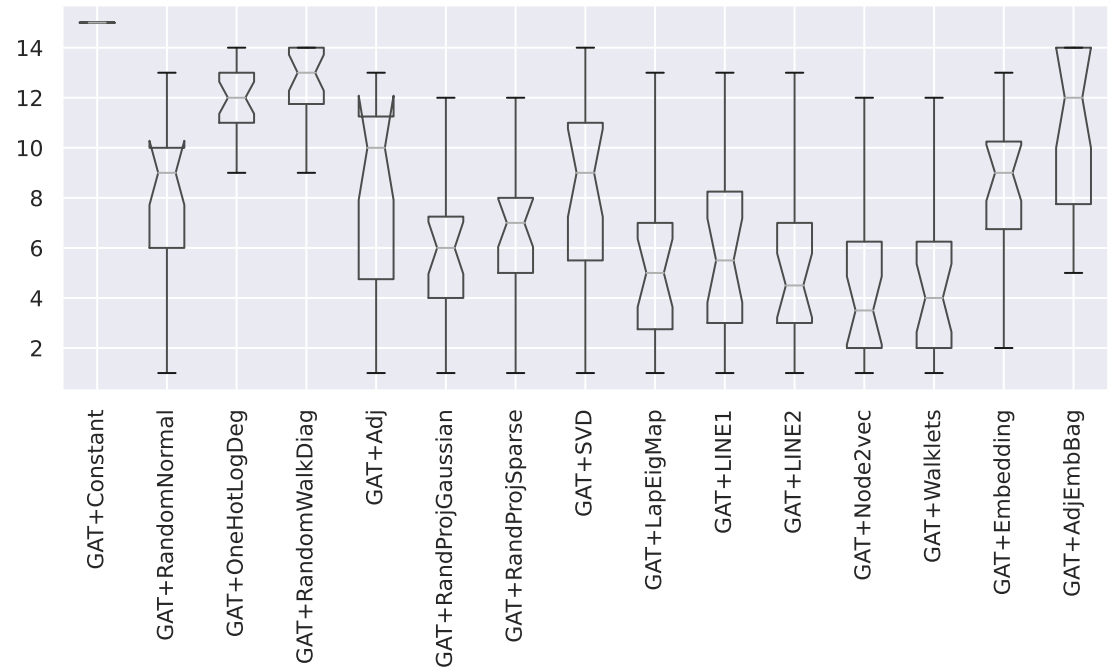
Network	Model	Feature	DISEASES	DisGeNET	GOBP
	GatedGCN	OneHotLogDeg	**0.602 ± 0.090**	0.591 ± 0.098	0.957 ± 0.074
	LabelProp	–	0.545 ± 0.000	**0.598 ± 0.000**	0.962 ± 0.000
	LogReg	Adj	0.494 ± 0.003	0.455 ± 0.000	1.002 ± 0.003
	LogReg	LapEigMap	0.538 ± 0.007	0.596 ± 0.001	0.985 ± 0.023
	LogReg	SVD	0.545 ± 0.027	0.433 ± 0.005	0.721 ± 0.063
	LogReg	LINE1	0.608 ± 0.030	0.596 ± 0.030	**1.025 ± 0.040**
	LogReg	LINE2	0.575 ± 0.062	0.557 ± 0.018	1.017 ± 0.093
	LogReg	Node2vec	**0.633 ± 0.065**	0.500 ± 0.043	0.904 ± 0.078
	LogReg	Walklets	0.485 ± 0.071	0.458 ± 0.088	0.707 ± 0.069
OmniPath	GCN	OneHotLogDeg	**1.196 ± 0.054**	**0.983 ± 0.067**	**1.742 ± 0.107**
	SAGE	OneHotLogDeg	0.913 ± 0.074	0.779 ± 0.051	1.539 ± 0.074
	GIN	OneHotLogDeg	0.795 ± 0.044	0.731 ± 0.030	1.521 ± 0.086
	GAT	OneHotLogDeg	0.775 ± 0.142	0.773 ± 0.135	1.685 ± 0.110
	GatedGCN	OneHotLogDeg	1.112 ± 0.039	0.898 ± 0.031	1.698 ± 0.059
	LabelProp	–	**1.358 ± 0.000**	0.897 ± 0.000	1.593 ± 0.000
	LogReg	Adj	1.051 ± 0.001	0.709 ± 0.000	1.862 ± 0.000
	LogReg	LapEigMap	1.319 ± 0.000	**1.060 ± 0.004**	1.943 ± 0.004
	LogReg	SVD	0.866 ± 0.001	0.635 ± 0.006	1.512 ± 0.012
	LogReg	LINE1	0.834 ± 0.026	0.787 ± 0.007	1.893 ± 0.032
	LogReg	LINE2	0.913 ± 0.033	0.692 ± 0.012	1.844 ± 0.032
	LogReg	Node2vec	1.178 ± 0.035	0.924 ± 0.035	**2.125 ± 0.059**
	LogReg	Walklets	0.915 ± 0.041	0.795 ± 0.045	1.704 ± 0.064
ProteomeHD	GCN	OneHotLogDeg	0.690 ± 0.064	0.637 ± 0.066	**1.459 ± 0.138**
	SAGE	OneHotLogDeg	0.556 ± 0.107	0.644 ± 0.030	1.304 ± 0.041
	GIN	OneHotLogDeg	0.608 ± 0.083	0.606 ± 0.107	1.350 ± 0.221
	GAT	OneHotLogDeg	**0.705 ± 0.115**	0.687 ± 0.063	1.339 ± 0.199
	GatedGCN	OneHotLogDeg	0.521 ± 0.101	**0.712 ± 0.044**	1.169 ± 0.052
	LabelProp	–	0.709 ± 0.000	0.669 ± 0.000	1.036 ± 0.000
	LogReg	Adj	0.849 ± 0.047	0.619 ± 0.005	1.329 ± 0.141

Table C.2 (cont'd).

Network	Model	Feature	DISEASES	DisGeNET	GOBP
	LogReg	LapEigMap	**0.955 ± 0.001**	0.721 ± 0.016	1.219 ± 0.039
	LogReg	SVD	0.878 ± 0.002	0.736 ± 0.008	**1.508 ± 0.079**
	LogReg	LINE1	0.566 ± 0.089	0.667 ± 0.023	1.307 ± 0.060
	LogReg	LINE2	0.576 ± 0.099	0.663 ± 0.032	1.226 ± 0.085
	LogReg	Node2vec	0.643 ± 0.076	**0.772 ± 0.068**	1.357 ± 0.040
	LogReg	Walklets	0.432 ± 0.050	0.525 ± 0.111	1.048 ± 0.061
SIGNOR	GCN	OneHotLogDeg	**1.387 ± 0.067**	**1.167 ± 0.079**	**1.732 ± 0.031**
	SAGE	OneHotLogDeg	0.924 ± 0.106	0.838 ± 0.086	1.441 ± 0.094
	GIN	OneHotLogDeg	1.106 ± 0.058	0.870 ± 0.055	1.479 ± 0.090
	GAT	OneHotLogDeg	1.180 ± 0.069	1.009 ± 0.087	1.576 ± 0.047
	GatedGCN	OneHotLogDeg	1.067 ± 0.028	0.852 ± 0.060	1.430 ± 0.085
	LabelProp	–	1.288 ± 0.000	1.096 ± 0.000	1.695 ± 0.000
	LogReg	Adj	1.303 ± 0.003	1.052 ± 0.001	1.417 ± 0.004
	LogReg	LapEigMap	1.306 ± 0.004	1.056 ± 0.005	1.746 ± 0.010
	LogReg	SVD	0.864 ± 0.004	0.801 ± 0.002	1.183 ± 0.002
	LogReg	LINE1	**1.412 ± 0.033**	0.955 ± 0.022	**1.781 ± 0.056**
	LogReg	LINE2	1.257 ± 0.034	0.917 ± 0.016	1.572 ± 0.047
	LogReg	Node2vec	1.341 ± 0.021	**1.172 ± 0.027**	1.684 ± 0.099
	LogReg	Walklets	1.017 ± 0.101	0.985 ± 0.049	1.259 ± 0.065



(a) GCN



(b) GAT

Figure C.1: **Boxplots representing the rankings of different feature construction when used by GNNs (lower the better).** Each point in a box is a ranking of a particular node feature construction on a specific dataset. A lower rank indicates that the particular node feature achieved higher test performance than others. For both GCN and GAT, *node2vec* appears to be the top-ranked node feature overall.

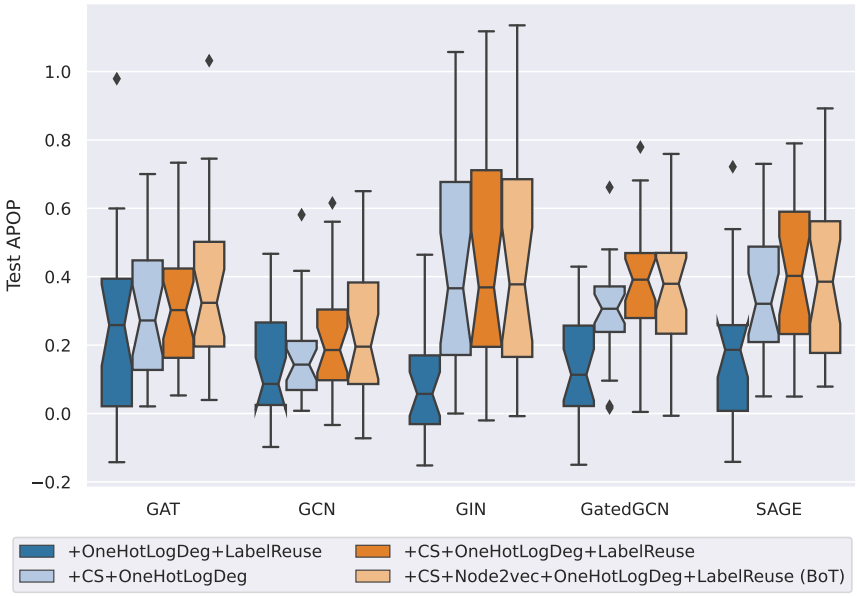


Figure C.2: **Boxplots representing the performance improvement of using different tricks with GNNs across datasets.** Each point in a box is the test APOP difference of GNN with added tricks vs. plain GNN using OneHotLogDeg feature on a specific dataset. A positive value implies the added trick improves GNN performance.

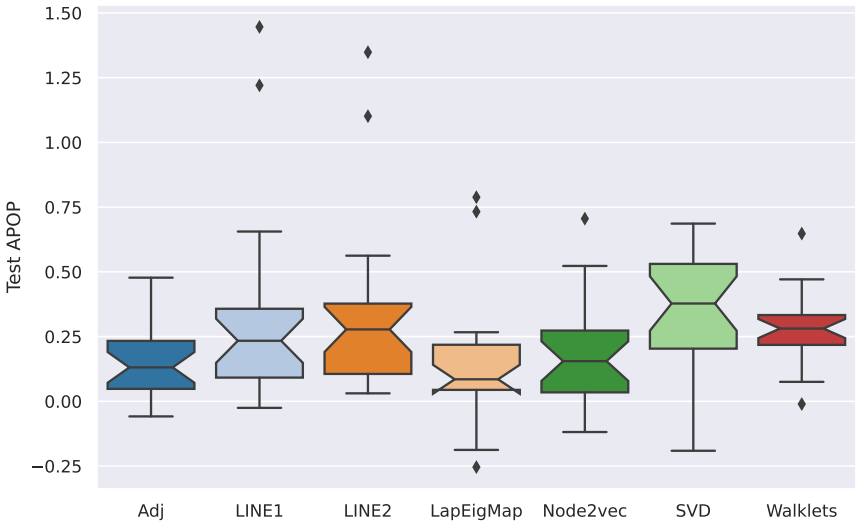


Figure C.3: **Boxplots representing the performance improvement of using C&S with logistic regression models across datasets.** Each point in a box is the test APOP difference of logistic regression augmented with C&S vs. plain logistic regression on a specific dataset. A positive value implies C&S improves logistic regression performance.

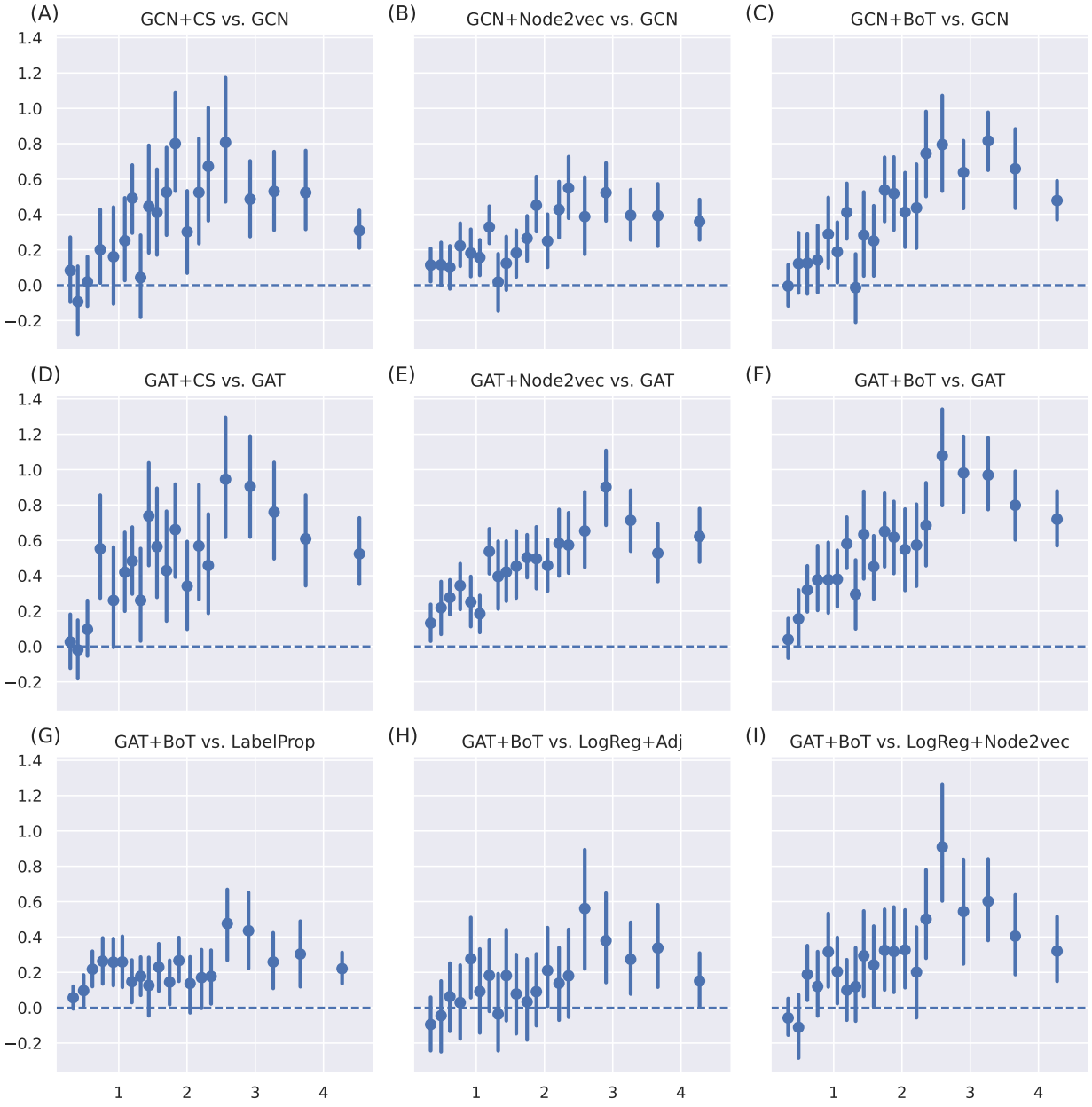


Figure C.4: **Relationships between the corrected homophily ratio and performance difference between methods.** Each panel summarizes tasks over three networks (BioGRID, HumanNet, ComPPIHumanInt) and three gene set collections (DISEASES, DisGeNET, GOBP), with a total of 1,672 tasks. In each panel, the x-axis represents the corrected homophily ratio and the y-axis represents the test APOP performance difference between the two methods. The first (A, B, C) and second rows (D, E, F) show the performance improvement to GCN and GAT when augmented with individual tricks or combined BoT. The third row (G, H, I) shows the performance difference between the best GNN method and the baseline methods, including label propagation and logistic regression.

Table C.3: **Combined SOTA performance reference.** LogReg* are the best test performances achieved from any logistic regression models we have tested for each dataset. Colored texts indicate the first (green), second (orange), and third (purple) best performance achieved for a particular dataset (network × label).

Network	Model	DISEASES	DisGeNET	GOBP
BioGRID	LabelProp	1.210 ± 0.000	0.931 ± 0.000	1.885 ± 0.000
	LogReg*	**1.556 ± 0.002**	1.026 ± 0.022	**2.571 ± 0.015**
	GCN+BoT	1.511 ± 0.053	1.014 ± 0.020	2.411 ± 0.044
	SAGE+BoT	1.486 ± 0.058	**1.031 ± 0.049**	2.402 ± 0.061
	GIN+BoT	1.410 ± 0.024	1.007 ± 0.028	2.386 ± 0.022
	GAT+BoT	**1.609 ± 0.048**	**1.037 ± 0.036**	**2.624 ± 0.070**
	GatedGCN+BoT	**1.547 ± 0.027**	**1.038 ± 0.006**	**2.517 ± 0.047**
HumanNet	LabelProp	**3.728 ± 0.000**	3.059 ± 0.000	3.806 ± 0.000
	LogReg*	**3.812 ± 0.000**	**3.158 ± 0.000**	**4.053 ± 0.021**
	GCN+BoT	3.552 ± 0.050	3.053 ± 0.078	**3.921 ± 0.045**
	SAGE+BoT	3.401 ± 0.066	3.052 ± 0.041	3.816 ± 0.083
	GIN+BoT	3.513 ± 0.029	3.054 ± 0.051	3.861 ± 0.063
	GAT+BoT	**3.761 ± 0.060**	**3.100 ± 0.031**	**3.908 ± 0.086**
	GatedGCN+BoT	3.677 ± 0.066	**3.086 ± 0.020**	3.889 ± 0.048
ComPPIHumanInt	LabelProp	1.352 ± 0.000	1.106 ± 0.000	2.076 ± 0.000
	LogReg*	1.644 ± 0.006	**1.240 ± 0.009**	**2.806 ± 0.049**
	GCN+BoT	1.648 ± 0.012	1.211 ± 0.013	2.685 ± 0.047
	SAGE+BoT	**1.694 ± 0.055**	1.210 ± 0.033	2.629 ± 0.082
	GIN+BoT	1.608 ± 0.020	**1.219 ± 0.006**	2.611 ± 0.021
	GAT+BoT	**1.665 ± 0.035**	**1.230 ± 0.025**	**2.785 ± 0.041**
	GatedGCN+BoT	**1.672 ± 0.053**	1.218 ± 0.009	**2.735 ± 0.048**
BioPlex	LabelProp	0.964 ± 0.000	0.556 ± 0.000	2.174 ± 0.000
	LogReg*	**1.358 ± 0.006**	**0.939 ± 0.002**	**2.587 ± 0.022**
	GCN+BoT	1.324 ± 0.027	**0.911 ± 0.010**	**2.553 ± 0.069**
	SAGE+BoT	1.246 ± 0.022	0.865 ± 0.035	2.513 ± 0.038
	GIN+BoT	**1.349 ± 0.010**	0.868 ± 0.009	2.504 ± 0.024
	GAT+BoT	**1.355 ± 0.040**	**0.873 ± 0.025**	2.548 ± 0.075

Table C.3 (cont'd).

Network	Model	DISEASES	DisGeNET	GOBP
	GatedGCN+BoT	1.301 ± 0.035	0.859 ± 0.011	**2.590 ± 0.029**
HuRI	LabelProp	0.545 ± 0.000	0.598 ± 0.000	0.962 ± 0.000
	LogReg*	**0.650 ± 0.000**	0.656 ± 0.000	1.084 ± 0.020
	GCN+BoT	**0.634 ± 0.065**	**0.693 ± 0.019**	**1.229 ± 0.119**
	SAGE+BoT	0.593 ± 0.040	0.679 ± 0.031	**1.190 ± 0.127**
	GIN+BoT	0.583 ± 0.042	**0.702 ± 0.012**	1.143 ± 0.047
	GAT+BoT	**0.667 ± 0.048**	0.687 ± 0.045	1.174 ± 0.047
	GatedGCN+BoT	0.596 ± 0.017	**0.695 ± 0.018**	**1.195 ± 0.054**
OmniPath	LabelProp	1.358 ± 0.000	0.897 ± 0.000	1.593 ± 0.000
	LogReg*	**1.542 ± 0.017**	**1.093 ± 0.006**	**2.125 ± 0.059**
	GCN+BoT	**1.577 ± 0.038**	1.073 ± 0.024	2.052 ± 0.032
	SAGE+BoT	1.465 ± 0.048	1.041 ± 0.065	1.974 ± 0.040
	GIN+BoT	1.478 ± 0.032	**1.104 ± 0.017**	1.995 ± 0.046
	GAT+BoT	1.520 ± 0.028	**1.083 ± 0.025**	**2.067 ± 0.050**
	GatedGCN+BoT	**1.544 ± 0.036**	1.079 ± 0.020	**2.122 ± 0.080**
ProteomeHD	LabelProp	0.709 ± 0.000	0.669 ± 0.000	1.036 ± 0.000
	LogReg*	**0.955 ± 0.001**	**0.776 ± 0.000**	**1.519 ± 0.071**
	GCN+BoT	0.764 ± 0.017	**0.745 ± 0.026**	1.387 ± 0.088
	SAGE+BoT	0.747 ± 0.051	**0.734 ± 0.023**	1.425 ± 0.100
	GIN+BoT	0.771 ± 0.034	0.718 ± 0.029	**1.529 ± 0.161**
	GAT+BoT	**0.830 ± 0.039**	0.727 ± 0.042	**1.463 ± 0.052**
	GatedGCN+BoT	**0.829 ± 0.029**	0.716 ± 0.022	1.389 ± 0.097
SIGNOR	LabelProp	1.288 ± 0.000	1.096 ± 0.000	1.695 ± 0.000
	LogReg*	1.582 ± 0.006	**1.298 ± 0.007**	**1.894 ± 0.001**
	GCN+BoT	**1.590 ± 0.037**	**1.273 ± 0.013**	1.844 ± 0.031
	SAGE+BoT	1.540 ± 0.017	1.243 ± 0.005	1.784 ± 0.041
	GIN+BoT	**1.593 ± 0.026**	1.253 ± 0.010	**1.850 ± 0.028**
	GAT+BoT	**1.627 ± 0.042**	**1.260 ± 0.009**	1.799 ± 0.034

Table C.3 (cont'd).

Network	Model	DISEASES	DisGeNET	GOBP
	GatedGCN+BoT	1.525 ± 0.026	1.254 ± 0.006	**1.849 ± 0.029**

Table C.4: **Random splitting evaluation performance of different methods on the four primary OBNB datasets evaluated in APOP ↑ aggregated over five seeds. Bold** indicates the best-performing method within the method class (traditional ML or GNN). Blue indicates the evaluated performance is higher than the study-biased holdout evaluation counterpart.

			BioGRID		HumanNet	
Model	Features	C&S?	DisGeNET	GOBP	DisGeNET	GOBP
LabelProp	–	✗	1.415 ± 0.000	2.654 ± 0.000	**3.370 ± 0.000**	4.138 ± 0.000
LogReg	Adj	✗	**1.546 ± 0.001**	**3.479 ± 0.000**	3.280 ± 0.000	**4.300 ± 0.001**
LogReg	N2V	✗	1.485 ± 0.037	3.269 ± 0.022	2.987 ± 0.014	4.107 ± 0.033
LogReg	LapEigMap	✗	1.421 ± 0.000	2.812 ± 0.004	2.516 ± 0.002	3.876 ± 0.004
GCN	LogDeg	✗	1.055 ± 0.116	2.076 ± 0.204	2.770 ± 0.045	3.898 ± 0.063
GCN	LogDeg	✓	1.664 ± 0.066	2.832 ± 0.082	3.213 ± 0.010	4.134 ± 0.044
GCN†	N2V+LogDeg+Label	✓	1.780 ± 0.043	3.330 ± 0.049	**3.278 ± 0.023**	4.205 ± 0.030
GCN‡	N2V+LogDeg+Label	✓	1.769 ± 0.051	3.380 ± 0.047	3.274 ± 0.013	4.202 ± 0.023
GAT	LogDeg	✗	0.921 ± 0.290	2.722 ± 0.498	2.837 ± 0.197	3.838 ± 0.092
GAT	LogDeg	✓	1.791 ± 0.015	3.076 ± 0.037	3.252 ± 0.021	4.141 ± 0.016
GAT†	N2V+LogDeg+Label	✓	1.728 ± 0.063	3.357 ± 0.185	3.265 ± 0.018	4.226 ± 0.030
GAT‡	N2V+LogDeg+Label	✓	**1.818 ± 0.013**	**3.448 ± 0.010**	3.266 ± 0.003	**4.235 ± 0.020**

† recommended BoT, ‡ recommended BoT with fully tuned GNNs (see Appendix 6.3.4 for tuning details)

As shown in Table C.4, random-split performance is higher than study-biased holdout in all cases. This indicates that, besides being more realistic, the study-biased holdout split is a much harder evaluation scheme and thus provides a more stringent evaluation of the tested gene classification method.

Table C.5: **Statistics for all networks in OBNB (`obnbdata-0.1.0`)**

Network	Weighted	Num. nodes	Num. edges	Density	Category
HumanBaseTopGlobal [80]	✓	25,689	77,807,094	0.117908	Large & Dense
HuMAP [62]	✓	15,433	35,052,604	0.147180	Large & Dense
STRING [246]	✓	18,480	11,019,492	0.032269	Large
ConsensusPathDB [117]	✓	17,735	10,611,416	0.033739	Large
FunCoup [204]	✓	17,892	10,037,478	0.031357	Large
PCNet [100]	✗	18,544	5,365,116	0.015603	Large
BioGRID [240]	✗	19,765	1,554,790	0.003980	Medium
HumanNet [107]	✓	18,591	2,250,780	0.006513	Medium
HIPPIE [5]	✓	19,338	1,542,044	0.004124	Medium
ComPPIHumanInt [260]	✓	17,015	699,620	0.002417	Medium
OmniPath [249]	✗	16,325	289,134	0.001085	Small
ProteomeHD [131]	✗	2,471	125,172	0.020509	Small
HuRI [160]	✗	8,100	103,188	0.001573	Small
BioPlex [106]	✗	8,108	71,004	0.001080	Small
SIGNOR [201]	✗	5,291	28,676	0.001025	Small

Table C.6: **Statistics for all datasets in OBNB (`obnbdata-0.1.0`).**

Label	Network	Num. tasks	Num. pos. avg.	Num. pos. std.	Num. pos. med.
DISEASES	BioGRID	145	178.1	137.4	127.0
	BioPlex	72	123.8	64.4	101.5
	ComPPIHumanInt	145	174.6	134.5	125.0
	ConsensusPathDB	144	177.4	137.5	126.0
	FunCoup	145	177.1	135.1	127.0
	HIPPIE	143	178.1	137.6	127.0
	HuMAP	123	168.0	119.2	120.0
	HuRI	50	130.3	56.7	112.5
	HumanBaseTopGlobal	149	178.5	137.7	129.0
	HumanNet	142	179.0	136.9	127.0
	OmniPath	135	180.2	131.1	131.0
	PCNet	143	171.8	130.6	122.0
	ProteomeHD	15	76.9	22.4	70.0
	SIGNOR	89	144.6	89.4	117.0
	STRING	146	175.4	135.6	126.0
DisGeNET	BioGRID	305	208.3	143.1	159.0
	BioPlex	189	138.6	71.4	111.0
	ComPPIHumanInt	301	204.1	138.7	159.0
	ConsensusPathDB	298	207.4	140.8	161.5
	FunCoup	299	204.7	139.4	158.0
	HIPPIE	306	208.1	142.9	159.5
	HuMAP	279	194.3	126.7	155.0
	HuRI	152	122.9	54.7	108.0
	HumanBaseTopGlobal	287	219.7	145.7	173.0
	HumanNet	302	204.2	140.3	158.5
	OmniPath	298	199.6	136.0	153.5
	PCNet	292	202.1	135.5	159.0
	ProteomeHD	56	78.0	24.8	71.0
	SIGNOR	219	147.3	81.9	124.0
	STRING	296	208.0	140.6	162.0
GOBP	BioGRID	114	89.5	37.1	76.0
	BioPlex	38	77.6	22.6	76.0
	ComPPIHumanInt	104	91.8	37.0	77.5
	ConsensusPathDB	112	90.1	37.0	76.5
	FunCoup	114	87.8	36.7	74.0

Table C.6 (cont'd).

Label	Network	Num. tasks	Num. pos. avg.	Num. pos. std.	Num. pos. med.
	HIPPIE	111	89.2	37.1	76.0
	HuMAP	96	84.6	32.3	74.0
	HuRI	27	69.9	16.0	65.0
	HumanBaseTopGlobal	115	89.2	37.3	76.0
	HumanNet	117	88.6	36.9	75.0
	OmniPath	106	88.7	36.2	74.0
	PCNet	105	89.0	36.0	77.0
	ProteomeHD	5	80.4	22.6	70.0
	SIGNOR	41	81.3	22.7	78.0
	STRING	116	88.9	36.6	75.0
GOCC	BioGRID	71	96.0	36.3	87.0
	BioPlex	35	77.0	20.9	71.0
	ComPPIHumanInt	68	96.1	35.5	88.0
	ConsensusPathDB	71	95.5	35.8	87.0
	FunCoup	71	95.9	36.4	87.0
	HIPPIE	69	97.3	35.8	89.0
	HuMAP	62	91.8	33.4	82.0
	HuRI	21	69.0	12.4	67.0
	HumanBaseTopGlobal	71	96.9	36.3	88.0
	HumanNet	70	95.8	35.6	88.0
	OmniPath	67	94.0	34.6	84.0
	PCNet	70	93.5	35.0	85.0
	ProteomeHD	2	59.5	4.5	59.5
	SIGNOR	26	66.4	11.3	67.5
	STRING	69	96.1	35.6	88.0
GOMF	BioGRID	63	97.7	39.8	85.0
	BioPlex	25	79.6	18.0	79.0
	ComPPIHumanInt	62	98.1	39.7	86.0
	ConsensusPathDB	63	98.2	39.5	85.0
	FunCoup	64	97.5	39.1	83.5
	HIPPIE	63	97.7	39.9	84.0
	HuMAP	58	92.1	36.8	74.5
	HuRI	22	74.8	12.8	73.5
	HumanBaseTopGlobal	62	99.2	39.9	86.0
	HumanNet	61	98.7	39.5	84.0

Table C.6 (cont'd).

Label	Network	Num. tasks	Num. pos. avg.	Num. pos. std.	Num. pos. med.
	OmniPath	64	95.2	38.5	83.5
	PCNet	58	97.4	38.6	86.0
	ProteomeHD	2	59.5	4.5	59.5
	SIGNOR	25	91.4	23.7	94.0
	STRING	62	98.0	39.3	82.5