Bulletin of the Technical Community on

Data Engineering

June 2023 Vol. 47 No. 2

IEEE Computer Society

Letters

Opinions

Special Issue on Graph Neural Networks

Conference and Journal Notices

The TC on Data Engineering

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TCDE web page is http://tab.computer.org/tcde/index.html.

The Data Engineering Bulletin

The Bulletin of the Technical Community on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

The Data Engineering Bulletin web site is at http://tab.computer.org/tcde/bull_about.html.

Letter from the Editor-in-Chief

Graph is an essential representation for many real-world data exhibiting intricate relationships. We first surveyed work in this field in one of our 2017 issues, where the focus was on developing efficient algorithms for very large graphs. Then, in one of our 2022 issues, we highlighted applications on graph data, including knowledge graphs, causality, and reasoning over knowledge graphs.

This issue is devoted to Graph Neural Networks (GNNs), a topic that has garnered a great deal of attention and is the driving force behind many important applications such as personalization and recommendations systems. GNNs are effective at relational reasoning, which expands the frontiers of traditional machine learning methodologies. Significant effort was also devoted to the scalability challenge, which is crucial in today's data-driven landscape characterized by vast and complex data structures. This issue, curated by Karthik Subbian, contains eight papers from leading researchers in this field, which include comprehensive surveys as well as deep dives into key technical challenges.

Wang-Chiew Tan has penned an opinion piece on the topic of data management and LLMs. It is a call to action for the database community in light of the rise of LLMs and generative AI. While LLMs provide an adept natural language interface to unstructured data, the absence of structure or a definitive data schema frequently results in less reliable outcomes. DBMSs, on the other hand, cannot handle unstructured data, a category that greatly eclipses structured data in terms of sheer volume. Tan suggests that a more robust, next-generation data management paradigm might emerge from the fusion of LLMs' strengths with those of DBMSs.

We would also like to congratulate Professor Kyu-Young Whang on receiving the prestigious TCDE Service Award, as well as celebrate his retirement and enormous contributions to the field of data engineering. We are privileged to publish a letter from Whang in this issue, in which he reminisced about the many research projects he had spearheaded, the changes he had witnessed in the field over the decades, and the community he had mentored.

Haixun Wang
Instacart

Letter from the TCDE Service Award Winner

A Life-long Saga with Data Engineering

It is my great honor to receive this prestigious TCDE Service Award in recognition of my life-long contribution to the data engineering community over 3 to 4 decades. I am retired now but, looking back, I have really been privileged to serve our community for the advancement of the data engineering discipline through various opportunities.

ICDE, TCDE, and VLDB Endowment

I had opportunities to serve VLDB and ICDE in various capacities including the general chair (VLDB2006), honorary general chair (ICDE 2015), a PC co-chair (VLDB2000, ICDE 2006), ICDE steering committee member (2007-2015), VLDB Endowment trustee twice (1998-2003, 2010-2015), and TCDE executive including chair and advisor (2011-2022). An early contributor of ICDE from the 2^{nd} conference in 1986 as a PC member, I also served as a program co-chair or vice chair many times in initial years from 1989 helping to settle the newly established conference. I am glad that ICDE has continuously been a top conference in the data engineering field. During my tenure as the TCDE Chair, we significantly broadened TCDE activities raising the level of vitality and prestige of TCDE. We initiated TCDE Archives restoring many years of institutional memory, newly instituted the IEEE TCDE Awards, and initiated membership promotion tripling the membership. In the VLDB Endowment, we have done a lot to promote global database research, but I would like to note on two efforts in particular. The first one is "broadening"; the scope of database research, in which I participated as an endowment trustee and PC co-chair (VLDB2000). This effort started from VLDB2000, resulting in the creation of the "Infrastructure for Information System (IIS)" track in 2002. The IIS track had lasted until 2013 when it was merged back with the Core DB track to a single one as it fulfilled its original mission. This broadening initiative significantly enlarged the scope of database research as it is today. The second one is that the endowment eagerly supported the Asia-Pacific region, then lagging in database research, to help bring it up to a level equivalent to those of the Americas and European regions—by various programs including the "VLDB database school." Nowadays, the Asia-Pacific region stands very strong and competitive with others.

The VLDB Journal

I also had the honor to serve the VLDB Journal for 19 years continuously as a founding editorial board member, an Editor-in-Chief (EIC), and the coordinating EIC (1990-2009). We emphasized on the strong editorial board, identifying timely impactful topics for thematic special issues, guaranteeing timely reviews, and increasing availability. During my tenure as the coordinating EIC, the VLDB Journal ranked the top in the Information Systems field with the highest impact factor (7.067 in 2008) according to Thomson's Science Citation Index. I am glad that nowadays the VLDB Journal stands itself as a top journal in the data engineering field.

Awards Committees

It was an honor to serve many prestigious awards committees including the SIGMOD Jim Gray dissertation award committee (2007-2012), the VLDB 10-year best paper award committee ('03,'05,'06,'10,'12), ICDE Influential paper award committee (2004-2008), TCDE awards committee (2014-2017 as advisor and member), DASFAA awards committee (2011-2019 as chair and member), and many best paper award committees including ICDE2006 (as chair) helping to ensure high academic standards.

Asia-Pacific

I also was privileged to serve the Asia-Pacific community through steering committee activities of DASFAA including chair, advisor, and awards chair for 15 years (1999-2014). An early contributor from the 2 nd conference in 1991 as a PC co-chair, I helped establish the current stature of DASFAA and globalize the DASFAA conferences. I am glad that DASFAA stands itself now as a prestigious data engineering conference serving the world-wide community as well as the Asia-Pacific one. I also had an opportunity to contribute to PAKDD as a life member of the steering committee and to the Korea-Japan Database (KJDB) Working Group as a co-founder, chair, and advisor promoting active academic exchanges through annual KJDB workshops.

Korea

A no less important goal of my effort was to help bring up the level of data engineering research in Korea to a global one, which was barely sprouting when I first came back to Korea in 1990. I served as the chair of the Special Interest Group on Databases of the Korea Information Science Society (SIGDB of KISS—later renamed to be the Database Society of KIISE) in early 90's and the president of the Korean Institute of information Scientists and Engineers (KIISE) in '20's, through which I promoted globalization of computer science and data engineering research in Korea—including hosting VLDB2006, PAKDD2003, and DASFAA2004 in Seoul and creating the KIISE JCSE journal and IEEE BigComp conference with KIISE scholars. Today I am glad to see that the Korean data engineering research community stands strong by global standards.

Leadership and Goals

In all my effort in the leadership positions, my primary goals have been to ensure the highest standards for publications and to vitalize the research activities, which I hope made whatever little contribution to the advancement of our field. I wish to share this honor with so many colleagues who selflessly took initiatives, helped, and cooperated in various roles and responsibilities in the course of this decades-long saga. They are true heroes who are behind this flourishing field of data engineering that we are enjoying today. Thank you very much.

Kyu-Young Whang
Distinguished Professor Emeritus, KAIST
ACM Fellow, IEEE Life Fellow

Letter from the Special Issue Editor

Graph Neural Networks (GNNs) have propelled the field of graph-based machine learning, unlocking new and innovative applications in various domains, such as natural language processing, drug discovery, recommendation systems, and social network analysis. By leveraging the graph structure and node features, GNNs capture intricate relationships and dependencies in complex networks, resulting in more accurate predictions and a deeper understanding of the data. These advancements have led to significant breakthroughs in drug design, personalized recommendations, and community detection in social networks. Moreover, GNNs' ability to model and analyze structured data has opened up new avenues for advancing artificial intelligence. Particularly, integrating GNNs with large language models (LLMs) will elevate their capabilities by incorporating world knowledge from LLMs and enabling natural language querying of graph-structured data. These advancements are poised to reshape the landscape of GNN research, paving the way for exciting possibilities and future advancements.

Despite their promise, GNNs have limitations. These include challenges in generalizing to unseen graph structures, scalability issues with large-scale graphs, difficulties in handling heterophily, limited interpretability resulting in black-box behavior, complexities in data requirements and feature engineering, as well as concerns regarding over-smoothing and potential loss of discriminative information. In this special edition, seven carefully selected papers address these limitations and offer insights into improving GNNs.

Graph based machine learning has been centered on the premise of similar nodes have a stronger relationship. Heterophily breaks this assumption. The first article by **Zhu et al.**, titled "Heterophily and Graph Neural Networks: Past, Present, and Future," investigates the performance of GNNs on graphs exhibiting heterophily. The authors review various GNN designs proposed for handling heterophilous graphs and explore their connections to research objectives like robustness, fairness, and over-smoothing avoidance. They emphasize the need for tailored GNN designs specific to heterophily.

Explainable GNN models are often necessary for legal, regulatory, and compliance purposes. Two articles in this edition focus on the explainability of GNN models. **Kakkad et al.**'s "A Survey on Explainability of Graph Neural Networks" offers a comprehensive overview of explainability techniques for GNNs, categorizing them based on objectives, methodologies, and application scenarios. **Rex Ying**'s paper, "Generative Explanation for Graph Neural Networks: Methods and Evaluation," proposes a unified optimization framework for generative explanation methods, highlighting the shared characteristics and distinctions among these approaches.

Representation learning is an important topic in GNN research, and this edition features two articles that delve into this topic. First, **Han et al.** presents a graph contrastive learning (GCL) framework aimed at learning graph representations of homogeneous, heterogeneous, and hypergraphs. They discuss improvements in principled view generation, which contribute to generalizability, fairness, and interpretability. Next, **Seshadri**'s paper highlights the limitations of low-dimensional embeddings in learning representations. The work presents theoretical underpinnings showing how low-dimensional embeddings cannot capture the fine-grained community structure of real-world data.

Finally, **Wang et al.** presents the concept of "Customized Graph Neural Networks." They propose a novel framework, Customized-GNN, which generates sample-specific GNN models for individual graphs based on their structures. The authors show the effectiveness of this framework through comprehensive experiments on various graph classification benchmarks.

We believe these seven articles offer a sample of the ongoing work, recent advances, and existing limitations in the field of GNN research. By exploring various aspects of GNNs, such as performance on heterophilous graphs, explainability techniques, generative explanations, graph contrastive learning, limitations of low-dimensional embeddings, and customized GNN frameworks, these articles contribute to a deeper understanding of GNNs and their potential applications. Our special thanks to **Yoachen Xie** for their feedback on selected submissions and to **Nurendra Choudhary** for their role as the web publication chair for this edition.

Karthik Subbian
Amazon

Unstructured and structured data: Can we have the best of both worlds with large language models?

Wang-Chiew Tan
Reality Labs Research, Meta

1 Introduction

We are witnessing rapid advancements in the area of large language models (LLMs). A search on Google Scholar shows there are about 3,910 papers with "large language models" in the titles of the papers in 2022. As of April 12, 2023, there are already 1700 articles with LLMs in their titles. In addition to Google Scholar, we are also witnessing huge volumes of blog posts, news articles, twitter feeds, and open-source repositories around LLMs that have sprung up in recent months.

Perhaps ChatGPT (released on November 30, 2022) is the epitome of this LLM revolution that truly unleashed and showcased, to the masses, the power of what has been brewing in the natural language and machine learning communities in recent years. There are many things ChatGPT[1] can do and does so impressively by generating intelligent human-like responses to your questions. Through natural language as input and output[2], it can solve non-trival mathematical problems, to a certain extent [2, 19], translate your specification into code in a programming language of your choice (e.g., [9, 19]), help you write proses in different styles and the list of accolades goes on [16].

In addition to its ability to answer questions, one can also prompt it with tables, such as CSV files, and ask questions over the tables in natural language. More interestingly, it is also possible to prompt ChatGPT with both text and tables and ask questions over the two types of information seamlessly. A little more perseverance in this exercise quickly reveals that ChatGPT has a limit on how much one can input with each prompt and how much information it will retain, at least based on my experience when I tried this at the end of March 2023.

It is natural to wonder whether we can use ChatGPT, with some extensions, as a system for storing and querying data, with natural language as the primary medium of input and output, which it excels at. What are the challenges of doing so, and how can we, as database practitioners and theoreticians, make progress in this context?

2 Unstructured and structured data

A lot of data, including text, images, audio, and videos, sits "outside the box" today. Often, such unstructured data contain multiple modalities simultaneously. For example, we often find text in images [10], and we may also find text associated with videos and/or images on the web. Unstructured data is prevalent to a large extent because it is easily authored and shared by users [6] through a variety of apps and authoring tools that are widely available. Such data is often queried with keyword search and today, they can also be queried in natural language with LLMs. Typically the cost of devising a schema and setting up a database for querying the data inhibits the use of a database management system (DBMS) upfront. However, as data scales, the need for structure and semantics becomes more critical, so as to enable faster and more accurate retrieval of content. For example, organizing photos by year, trips, or entity types (such as people or pets) adds some structure, which makes answering certain types of queries much faster and more accurate. However, answering complex queries such as "*when was the last time I went to the coffee shop beside restaurant Italio?*" or "*how many times did I celebrate Anna's birthday with*

[1] I use the term "ChatGPT" as a representative for general chat systems based on LLMs for the rest of this article.
[2] GPT-4 can accept image and text inputs.

a mango cake?" requires non-trivial reasoning and computation over the data that goes beyond the capabilities of LLMs today [20, 22].

Enterprise data sits on the other end of the spectrum. It is, for the most part, not authored by everyday users and is highly structured, often sitting "in a box" in some DBMS. Enterprise data comes with a well-devised schema and is typically highly optimized to serve a sizable query workload with great efficiency. Such data is often queried directly with SQL which is adequately expressive for specifying complex queries such as those with aggregates and recursion. Significant research has also been carried out to enable querying a database with natural language (e.g., [15, 18]), where questions are posed in natural language and translated into SQL, which can then be executed over the DBMS. A DBMS is highly optimized to handle large amounts of data and can also perform transactions with ACID guarantees [4]. However, the core dbms does not understand natural language and it tends to fall short in its ability to query unstructured data, which are often stored as blobs and adding semantics to the blobs require additional effort. For example, "*show me the sales numbers over Black Friday and Cyber Monday last year*" requires commonsense knowledge on what "sales numbers" mean and how that maps to relevant table attribute(s), which may be stored under different names in different databases, and when Black Friday and Cyber Monday occurred. Another example is "*find all items with good reviews that are similar to these images*," which requires matching images (semantically) and interpreting what "good reviews" mean based on the data.

Despite the divide in how unstructured and structured data are managed today, the desire to query in natural language is common to both. At the same time, it is unreasonable and unnatural to expect all unstructured data to fit in some structure, or vice versa to leverage one system for querying. So, how can we effectively query both types of data with the help of LLMs, which possess tremendous knowledge and language understanding?

3 The best of both worlds with LLMs?

LLMs contain tremendous parametric knowledge in their model parameters but lack the ability to incorporate external data (i.e., data outside their model). Hence, if a model is trained based on data up to, say, 2021, it will not provide correct answers about events or facts that require knowledge after 2021. For example, if someone asks the question "*How were the midterm election results of 2022?*" on the webpage with `text-davinci-003` [13], the answer returned is "*The midterm election results of 2022 are not yet available, as the election has not yet taken place*." This is because `text-davinci-003` is trained with data up to June 2021.

Retrieval-augmented language models (e.g., [3, 8, 25]) overcome this limitation by adopting a semi-parametric approach to answering queries. They use external data, by first retrieving relevant data from an external data store, and then attempts to answer a question conditioned on the retrieved data and with their parametric knowledge. However, as demonstrated by (Table 5, [20]), such systems can still perform poorly on complex queries involving aggregates and certain types of temporal queries. This is because, for such queries, oftentimes, it is impossible to fit all necessary data for answering the question into the finite-length token input imposed by language models. However, as LLMs get even larger or as more advances are made to increase the token limits imposed by language models, one can anticipate that LLMs will take larger and larger inputs in future and the finite-length token limit may no longer be an issue soon. At the same time, it is likely that there will be even larger datasets to manage and an even larger set of data is required for computing the right answers. So this problem will persist, at least for a while.

A proposal to overcome the above limitation for some types of queries is described in [21]. The paper presents a vision of using views (e.g., as tables) to structure portions of the underlying data sources. The data sources may be of different modalities, such as text, images, videos, or even tables. Views are used to surface important properties about data or associations between data of different modalities and LLMs are used to translate natural language queries into queries (e.g., SQL in this case) that can be executed over the views whenever possible. With this proposal, a key question is to understand when a natural language query can be answered with views. If a query cannot be answered using views, the system falls back to retrieval-augmented language models to answer

the query to its best effort. Alternatively, the two components (view-based and retrieval-based query answering components) can also collaborate to produce a final answer. There are several other questions raised in the paper, such as what views should be materialized? How can one automatically select the "right" views to materialize given an anticipated query workload? And how can one decide when a question is better answered with views, the retrieval system, or even both?

In addition to the above, I will highlight below what I believe are some of the more pertinent questions that may be of immediate interest to the database community:

Query answering with different resources and budget constraints The topic of finding a good query plan to answer an given query was already discussed in [21]. The core of that system relies on two components — views and a retrieval-augmented language model for answering queries. It is conceivable to augment the system with additional components, such as one that generates images given a natural language prompt, which may sometimes be useful for answering certain queries[3]. A key question then becomes how do we understand when to leverage which component for answering a query or have the components collaborate to derive an answer? Furthermore, LLMs are compute-intensive and can be slow in generating an answer. Some LLMs are also not free. In addition to deciding how best to answer a query with all the available resources, how can one account for the strengths and limitations of each component to enumerate and compare plans for computing an answer to a given query and/or under a given budget? Can we also use a language model to generate a query plan or some parts of it, similar to how language models have been used to self-reason a sequence of steps to derive answers from questions (e.g., [23, 24])?

Provenance Provenance is well-studied for certain classes of SQL queries, and is roughly defined as the source tuples that explains why a tuple is in the result of the query. As mentioned in [21], one should attempt to answer queries using table views whenever possible by translating the natural language query into a SQL query that can be executed over the views. This way, it is possible that provenance can be obtained "for free". However, SQL queries that are generated by LLMs can be complex, for example, with nested SQL queries and/or aggregates in the FROM or WHERE clauses. For such cases, can we decompose the generated SQL query into a sequence of one or more "simpler queries" instead, where the provenance for simpler queries is well-understood and can be derived easily? If this is not possible, can we strategize a plan for answering the query in a different way so that provenance can be derived? The problem of finding an alternative query plan is related to the discussion in the earlier paragraph, but here the focus is on deriving a plan with sufficiently simple steps to enable provenance.

Retrieval-augmented language models also provide more guarantees for providing evidence for their answers. We are also beginning to witness implementations where sources of answers presented by retrieval-augmented language models are returned as part of the answers [11]. However, more research needs to be carried out to attribute provenance to training data (e.g., [1, 7, 14]) to form a more comprehensive picture of provenance for the provided answer.

Prompt Engineering Analysis LLMs have limits on the number of tokens they can take as input. Even if one takes advantage of all the tokens one can use, prompting them with more information does not always translate to better answers as sometimes, presenting the language model with more information confuses the language model. This means one needs to be judicious in what we send to a language model for it to derive a correct answer of high quality[4].

The answers returned by language models are also sensitive to how they are prompted. Sometimes the same question phrased slightly differently will result in completely different answers. In prompt engineering, the goal is to find the best prompt for the task at hand.

Given the maximum token limit of language models, can we optimize the answers returned by a language model by strategically summarizing relevant data and/or removing irrelevant data? For example, the entity matching system of [12], which uses a language model, immediately performs entity matching more accurately

[3] As they say, a picture is worth a thousand words.

[4] There are varied ways to answer a question correctly. Some are better than others.

when text descriptions of data are strategically shortened using a simple trick; by keeping only tokens of value, with high TD/IDF. On a more theoretical side, given a suitable definition of what is a prompt and assumptions about language models, can we characterize what types of queries that require access to external data can be answered with one prompt, a finite number of prompts, or an asymptotic number of prompts under a budget of tokens?

4 Conclusion

The field of LLMs is moving fast, both in research and industry. In addition to [21], [5] have also described the challenges of answering queries, in the context of augmented language models and data integration, with a single source or by chaining multiple sources. In [17], the authors described how a DBMS can be extended to leverage LLMs to improve query answering and also pointed to the direction of a hybrid query answering system involving both a DBMS and LLMs. I believe this is only the beginning and we will see many more visions, research, and implementations soon in this area of query answering systems that embrace both unstructured and structured data through the use of large language models.

Acknowledgements Many of the ideas above are inspired from discussions with Alon Halevy and Yuliang Li. I also thank many of my colleagues at Meta — Lambert Mathias, Richard Newcombe, and Luna Dong — for active discussions around large language models from which I have learnt lots and also to Lucian Popa for his feedback on this article.

References

[1] Ekin Akyurek, Tolga Bolukbasi, Frederick Liu, Binbin Xiong, Ian Tenney, Jacob Andreas, and Kelvin Guu. Towards tracing knowledge in language models back to the training data. In EMNLP, pages 2429–2446, December 2022. URL https://aclanthology.org/2022.findings-emnlp.180.

[2] Simon Frieder, Luca Pinchetti, Ryan-Rhys Griffiths, Tommaso Salvatori, Thomas Lukasiewicz, Philipp Christian Petersen, Alexis Chevalier, and Julius Berner. Mathematical Capabilities of ChatGPT, 2023.

[3] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. Realm: Retrieval-augmented language model pre-training. In ICML, 2020.

[4] Theo Haerder and Andreas Reuter. Principles of transaction-oriented database recovery. ACM Comput. Surv., 15(4): 287–317, dec 1983. ISSN 0360-0300. doi: 10.1145/289.291. URL https://doi.org/10.1145/289.291.

[5] Alon Y. Halevy and Jane Dwivedi-Yu. Learnings from data integration for augmented language models. CoRR, abs/2304.04576, 2023. URL https://doi.org/10.48550/arXiv.2304.04576.

[6] Alon Y. Halevy, Oren Etzioni, AnHai Doan, Zachary G. Ives, Jayant Madhavan, Luke K. McDowell, and Igor Tatarinov. Crossing the Structure Chasm. In CIDR, 2003. URL http://www-db.cs.wisc.edu/cidr/cidr2003/program/p11.pdf.

[7] Xiaochuang Han, Byron C. Wallace, and Yulia Tsvetkov. Explaining black box predictions and unveiling data artifacts through influence functions. In ACL, pages 5553–5563, July 2020. doi: 10.18653/v1/2020.acl-main.492. URL https://aclanthology.org/2020.acl-main.492.

[8] Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Few-shot Learning with Retrieval Augmented Language Models. 2022. URL http://arxiv.org/abs/2208.03299.

[9] Ali Kashefi and Tapan Mukerji. ChatGPT for Programming Numerical Methods, 2023.

[10] Douwe Kiela, Hamed Firooz, Aravind Mohan, Vedanuj Goswami, Amanpreet Singh, Pratik Ringshia, and Davide Testuggine. The hateful memes challenge: Detecting hate speech in multimodal memes. In NeurIPS, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1b84c4cee2b8b3d823b30e2d604b1878-Abstract.html.

[11] LangChain. Retrieval Question Answering with Sources. https://python.langchain.com/en/latest/modules/chains/index_examples/vector_db_qa_with_sources.html.

[12] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. Deep entity matching with pre-trained language models. Proc. VLDB Endow., 14(1):50–60, 2020. doi: 10.14778/3421424.3421431. URL http://www.vldb.org/pvldb/vol14/p50-li.pdf.

[13] OpenAI. text-davinci-003. cited on April 12, 2023, Playground at https://platform.openai.com/playground, info on training data at https://help.openai.com/en/articles/6643408-how-do-davinci-and-text-davinci-003-differ.

[14] Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. In NeurIPS, volume 33, pages 19920–19930, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/e6385d39ec9394f2f3a354d9d2b88eec-Paper.pdf.

[15] Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Yu Cheng, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. RASAT: Integrating Relational Structures into Pretrained Seq2Seq Model for Text-to-SQL. In EMNLP, 2022. doi: 10.48550/ARXIV.2205.06983. URL https://arxiv.org/abs/2205.06983.

[16] Kevin Rose. How Should I Use A.I. Chatbots Like ChatGPT? New York Times (Mar 30, 2023) https://www.nytimes.com/2023/03/30/technology/ai-chatbot-chatgpt-uses-work-life.html.

[17] Mohammed Saeed, Nicola De Cao, and Paolo Papotti. Querying Large Language Models with SQL, 2023.

[18] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In EMNLP, pages 9895–9901, November 2021. doi: 10.18653/v1/2021.emnlp-main.779. URL https://aclanthology.org/2021.emnlp-main.779.

[19] Paulo Shakarian, Abhinav Koyyalamudi, Noel Ngu, and Lakshmivihari Mareedu. An Independent Evaluation of ChatGPT on Mathematical Word Problems (MWP), 2023.

[20] Wang-Chiew Tan, Jane Dwivedi-Yu, Yuliang Li, Lambert Mathias, Marzieh Saeidi, Jing Nathan Yan, and Alon Y. Halevy. Timelineqa: A benchmark for question answering over timelines. CoRR, abs/2306.01069, 2023.

[21] Wang-Chiew Tan, Yuliang Li, Pedro Rodriguez, Richard James, Xi Victoria Lin, Alon Y. Halevy, and Scott Yih. Reimagining retrieval augmented language models for answering queries. CoRR, abs/2306.01061, 2023.

[22] James Thorne, Majid Yazdani, Marzieh Saeidi, Fabrizio Silvestri, Sebastian Riedel, and Alon Y. Levy. From Natural Language Processing to Neural Databases. VLDB Endow., 14(6):1033–1039, 2021. doi: 10.14778/3447689.3447706. URL http://www.vldb.org/pvldb/vol14/p1033-thorne.pdf.

[23] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In NeurIPS, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

[24] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In ICLR, 2023.

[25] Michihiro Yasunaga, Armen Aghajanyan, Weijia Shi, Rich James, Jure Leskovec, Percy Liang, Mike Lewis, Luke Zettlemoyer, and Wen tau Yih. Retrieval-augmented multimodal language modeling, 2022.

Heterophily and Graph Neural Networks: Past, Present and Future

Jiong Zhu[†] Yujun Yan[‡] Mark Heimann[§] Lingxiao Zhao[‖] Leman Akoglu[‖]
Danai Koutra[†]
[†] University of Michigan {jiongzhu, dkoutra}@umich.edu
[‡] Dartmouth College yujun.yan@dartmouth.edu
[§] Lawrence Livermore National Laboratory heimann2@llnl.gov
[‖] Carnegie Mellon University {lingxia1, lakoglu}@andrew.cmu.edu

Abstract

Recently, there has been interest in understanding the performance of Graph Neural Networks (GNNs) on input graphs exhibiting heterophily, or the tendency for nodes of different classes to connect. Initial findings showed that many standard GNN models struggled on certain benchmark datasets exhibiting high heterophily, prompting research into existing and novel GNN designs that improved learning in these contexts. However, further analyses revealed that certain highly heterophilous settings did not challenge GNNs without these specialized designs, raising questions about the true factors causing performance degradation. In this work, we first review various GNN designs proposed for handling graphs with heterophily, and examine their connections to other GNN research objectives such as robustness, fairness, and oversmoothing avoidance. Next, we conduct an empirical study to investigate the specific heterophilous graph conditions under which GNNs can and cannot perform effectively. Our analysis reveals that although high heterophily does not universally impede conventional GNNs, unique challenges in heterophilous graphs, particularly the intertwined effects with low-degree nodes and complex compatibility patterns, warrant GNN designs specifically tailored to heterophily. In conclusion, we discuss future research directions aimed at advancing the understanding of the impact of heterophily on GNNs across a broader range of contexts.

1 Introduction

Graph Neural Networks (GNNs) [51, 43] have gained prominence in recent years due to their remarkable theoretical and empirical potential for learning powerful representations of graph-structured data. Many real-world graphs or networks exhibit homophily, where nodes predominantly connect with others belonging to the same class [27, 58]. While early GNNs demonstrated promise on graphs with this property, they faced challenges on graphs exhibiting heterophily, where the majority of nodes connect to those of different classes [1, 29, 58]. This prompted investigations into GNN design choices conducive to learning on graphs with heterophily and sparked interest in developing new GNN models tailored for this property [58, 46, 6, 49, 24, 46, 52, 33].

Beyond improving the effectiveness of GNNs on heterophilous datasets, recent research has shown that the challenges posed by graphs with heterophily are closely connected to other GNN challenges, including oversmoothing [19, 5], algorithmic bias [18, 40], and sensitivity to adversarial attacks [60, 8, 44, 42, 17, 25]. Designs addressing heterophily often improve the ability of GNNs to handle these challenges as well, leading to significant advances in overall GNN capabilities [6, 46, 23, 55, 4].

Another line of work, however, has revisited whether early GNN designs were as ill-suited for learning from heterophilous graphs as initially thought. On some heterophilous networks, basic Graph Convolutional Networks (GCNs) [16] have proven competitive with, or even outperformed, models specifically designed for

heterophily [26, 24]. This has led to the proposition that the challenges posed by some graph datasets are not best captured by the traditional homophily ratio. Consequently, other works have focused on analyzing the properties of heterophilous graphs that challenge early GNNs and designing generalized homophily metrics that offer more insight into the difficulties a graph dataset may present [26, 24]. Thus, a valid debate exists over whether "heterophily" is a real problem that GNNs face.

Our work revisits this debate with additional analysis. First, we provide a concise review of recent designs for graphs with heterophily, their connections to other GNN research objectives, as well as generalized heterophily metrics. We then examine the heterophilous conditions under which conventional GNNs have been shown to be competitive with those tailored for heterophily. Our analysis reveals that while conventional GNNs can sometimes succeed in learning on heterophilous graphs without specialized designs, such condition is often broken when the underlying data has low-degree nodes and complex heterophilous patterns ("compatibility matrices"). Thus, we believe that continuing to develop GNNs that can learn across the spectrum of low-to-high homophily remains an important theoretical and empirical problem. We summarize our contributions as follows:

- We review and summarize recent designs proposed for graphs with heterophily (§3.1), providing a unifying intuition. Moreover, we discuss their use in subsequent GNN works and their implications for other objectives of GNN research (§3.2), such as fairness, robustness, and reducing oversmoothing.
- We conduct an empirical analysis on the conditions under which conventional GNNs can succeed on heterophilous datasets (§4). Our analysis demonstrates the unique challenges in achieving high separability of Neighborhood Label Distribution (NLD) when low-degree nodes (§4.2.2) or complex heterophilous patterns (§4.2.3) are present. These challenges hinder the effectiveness of conventional GNNs and are best addressed by GNN designs specifically tailored for heterophily (§4.2.4).
- We discuss future research directions aimed at enhancing our understanding of how heterophily impacts GNNs across a broader range of contexts (§5). These include moving beyond node classification and global homophily, introducing more diverse graph datasets and applications, and exploring the connections between heterophily and heterogeneity.

2 Notation and Preliminaries

In this section, we give the key notations and definitions that we use throughout our paper. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected, unweighted graph with node set $\mathcal{V}$ and edge set $\mathcal{E}$. We denote a general neighborhood centered around v as $N(v)$ ($\mathcal{G}$ may have self-loops), the corresponding neighborhood that does *not* include the ego (node v) as $\bar{N}(v)$, and the general neighbors of node v at exactly i hops/steps away (minimum distance) as $\bar{N}_i(v)$. For example, as shown in Fig. 1, $\bar{N}_1(v) = \{u : (u, v) \in \mathcal{E}\}$ are the immediate neighbors of v. We represent the graph by its adjacency matrix $\mathbf{A} \in \{0, 1\}^{n \times n}$ and its node feature matrix $\mathbf{X} \in \mathbb{R}^{n \times F}$, where the vector $\mathbf{x}_v$ corresponds to the *ego-feature* of node v, and $\{\mathbf{x}_u : u \in \bar{N}(v)\}$ to its *neighbor-features*. We further represent the degree of a node v by d_v, which denotes the number of neighbors in its immediate neighborhood $\bar{N}_1(v)$.

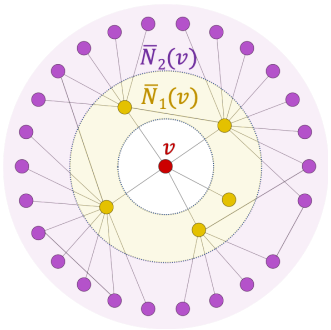


Figure 1: Neighborhoods.

We further assume a class label vector $\mathbf{y}$, which for each node v contains a unique class label $y_v \in \mathcal{Y}$, and the one-hot encoding $\text{onehot}(y_v)$ forms the row vectors of label encoding matrix $\mathbf{Y} \in \{0, 1\}^{n \times |\mathcal{Y}|}$. We further define $\mathcal{V}_i$ as the set of nodes $v \in \mathcal{V}$ with label $y_v = i$. The goal of semi-supervised node classification is to learn a mapping $\ell : \mathcal{V} \to \mathcal{Y}$, given a set of labeled nodes $\mathcal{T}_\mathcal{V} = \{(v_1, y_1), (v_2, y_2), ...\}$ as training data.

Graph Neural Networks (GNNs). From a probabilistic perspective, most GNN models assume the following local Markov property on node features: for each node $v \in \mathcal{V}$, there exists a neighborhood $N(v)$ such that y_v only depends on the ego-feature $\mathbf{x}_v$ and neighbor-features $\{\mathbf{x}_u : u \in N(v)\}$. Most models derive the class label y_v via the following representation learning approach:

$$\mathbf{r}_v^{(k)} = f\left(\mathbf{r}_v^{(k-1)}, \{\mathbf{r}_u^{(k-1)} : u \in N(v)\}\right), \ \mathbf{r}_v^{(0)} = \mathbf{x}_v, \text{ and } y_v = \arg\max\{\text{softmax}(\mathbf{r}_v^{(K)})\mathbf{W}\}, \tag{1}$$

where the embedding function f is applied repeatedly in K total rounds, node v's representation (or hidden state vector) at round k, $\mathbf{r}_v^{(k)}$, is learned from its ego- and neighbor-representations in the previous round, and a softmax classifier with learnable weight matrix $\mathbf{W}$ is applied to the final representation of v. Most existing models differ in their definitions of neighborhoods $N(v)$ and embedding function f. A typical definition of neighborhood is $N_1(v)$—i.e., the 1-hop neighbors of v. As for f, in graph convolutional networks (GCN) [16] each node repeatedly averages its own features and those of its neighbors to update its own feature representation. Using an attention mechanism, GAT [37] models the influence of different neighbors more precisely as a weighted average of the ego- and neighbor-features. GraphSAGE [12] generalizes the aggregation beyond averaging, and models the ego-features distinctly from the neighbor-features in its subsampled neighborhood.

Homophily and heterophily. In this work, we focus on heterophily in class labels. We first define the edge homophily ratio h as a measure of the graph homophily level, and use it to define graphs with strong homophily/heterophily:

Definition 2.1 (Edge Homophily Ratio [1, 58]) *The edge homophily ratio* $h = \frac{|\{(u,v):(u,v)\in\mathcal{E}\wedge y_u=y_v\}|}{|\mathcal{E}|}$ *is the fraction of edges in a graph which connect nodes that have the same class label (i.e., intra-class edges).*

Definition 2.2: Graphs with strong homophily have high edge homophily ratio $h \to 1$, while graphs with strong heterophily (i.e., low/weak homophily) have small edge homophily ratio $h \to 0$.

The edge homophily ratio in Dfn. 2.1 gives an *overall trend* for all the edges in the graph. The actual level of homophily may vary within different pairs of node classes, i.e., there is different tendency of connection between each pair of classes. For instance, in an online purchasing network [28] with three classes—fraudsters, accomplices, and honest users—, fraudsters connect with higher probability to accomplices and honest users. Moreover, within the same network, it is possible that some classes exhibit homophily, while others exhibit heterophily; we give an example in Figure 2. To capture the tendency of connection between each pair of classes, we define the empirical *class compatibility matrix* $\mathbf{H}$ as follows:

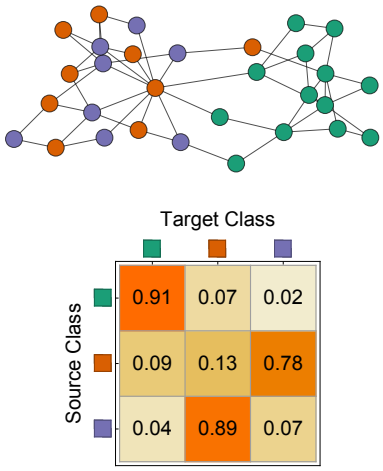


Figure 2: An example graph (top) and its empirical class compatibility matrix $\mathbf{H}$ (bottom). It demonstrates mixed homophily and heterophily, with node colors represent class labels: nodes in green show strong homophily, while nodes in orange and purple show strong heterophily.

Definition 2.3 (Empirical Class Compatibility Matrix [58, 57]) *The empirical class compatibility matrix* $\mathbf{H}$ *has entries* $[\mathbf{H}]_{i,j}$ *that capture the fraction of edges from a node in class* i *to a node in class* j*:*

$$[\mathbf{H}]_{i,j} = \frac{|\{(u,v) : (u,v) \in \mathcal{E} \wedge y_u = i \wedge y_v = j\}|}{|\{(u,v) : (u,v) \in \mathcal{E} \wedge y_u = i\}|}$$

By definition, the class compatibility matrix is a stochastic matrix, with each row summing up to 1.

Heterophily $\neq$ Heterogeneity. We remark that heterophily, which we study in this work, is a distinct network concept from heterogeneity. Formally, a network is heterogeneous [35] if it has at least two types of nodes and different relationships between them, and homogeneous if it has a single type of nodes (e.g., users) and a single type of edges (e.g., friendship). The type of nodes in heterogeneous graphs does *not* necessarily match the class labels y_v, therefore both homogeneous and heterogeneous networks may have different levels of homophily.

3 Progress for Addressing Heterophily in GNNs

In this section, we first present a concise overview of effective design strategies proposed to enhance GNN performance under heterophily (§3.1), and then discuss the implications of these designs for other GNN research in robustness, fairness, and reducing oversmoothing (§3.2).

3.1 Effective Designs for Graph Neural Networks on Heterophilous Graphs

We present an overview of the effective design strategies that have been recently proposed to enhance GNN performance on heterophilous graphs. We initiate our discussion with widely adopted designs (D1-D4) in GNN architectures for heterophily, three of which were initially explored in [58]. Subsequently, we examine two emerging designs (D5-D6) introduced by **(author?)** [46], which offer a novel unified approach to address two significant challenges faced by GNNs: oversmoothing and heterophily. Our primary focus in this section is to discuss the design principles and their underlying intuition for improving learning under heterophily without delving into specific models; we direct interested readers to a comprehensive survey by **(author?)** [52] for more details about particular models.

3.1.1 Ego- and Neighbor-embedding Separation

(author?) [58] identified three designs for improving the performance of GNNs on heterophilous graphs and provided theoretical justifications. At a high level, the first design entails encoding each ego-embedding (i.e., a node's embedding) *separately* from the aggregated embeddings of its neighbors, since they are likely to be dissimilar in heterophily settings. Formally, the representation (or hidden state vector) learned for each node v at GNN layer with depth k is given as:

$$\mathbf{r}_v^{(k)} = \texttt{COMBINE}\left(\mathbf{r}_v^{(k-1)},\ \texttt{AGGR}(\{\mathbf{r}_u^{(k-1)} : u \in \bar{N}(v)\})\right), \tag{2}$$

the neighborhood $\bar{N}(v)$ does *not* include v (no self-loops), the AGGR function aggregates representations *only* from the neighbors (in some way—e.g., average), and AGGR and COMBINE may be followed by a non-linear transformation. For heterophily, after aggregating the neighbors' representations, the definition of COMBINE (akin to 'skip connection' between layers) is critical: the ego-embedding and the aggregated neighbor-embedding should be processed by different sets of weight matrices under COMBINE. A simple way to combine the ego- and the aggregated neighbor-embeddings without 'mixing' them is with concatenation as in GraphSAGE [12]—rather than averaging *all* of them as in the GCN model by **(author?)** [16]. Intuitively, [58] argues that choosing a COMBINE function that separates the representations of each node v and its neighbors $\bar{N}(v)$ allows for more expressiveness, where the skipped or non-aggregated representations can evolve separately over multiple rounds of propagation without becoming prohibitively similar to representations aggregated from neighbors.

While this design was first discussed in [58] as the most critical design in the context of improving GNN performance under heterophily, it had already been proposed and adopted in prior GNN models such as GraphSAGE [12], without addressing the problem of heterophily. GCN-Cheby [9] and MixHop [1] also feature a variant of this design, with the AGGR function operating on $N(v)$ (with self-loops) instead of $\bar{N}(v)$ (no self-loops), while still featuring a separate channel for the ego-embedding. Following H_2GCN proposed in [58], this

design has gained wide adaptation for GNNs designed with heterophilous graphs in mind, such as CPGNN [57], GPR-GNN [6], FAGCN [49], FSGNN [50], JacobiConv [20], GGCN [46], GBK-GNN [10], ACM [24], and OrderedGNN [33]. More recently, **(author?)** [30] conducted benchmark experiments on additional heterophilous datasets and showed that GNNs featuring this design, including GAT [37] and UniMP [32] modified to include this design, achieve the best results in nearly all cases, which further validates the importance of the ego & neighbor embedding separation.

3.1.2 Higher-order Neighborhoods

The second design in [58] involves explicitly aggregating information from higher-order neighborhoods in each GNN layer, beyond the immediate neighbors of each node:

$$\mathbf{r}_v^{(k)} = \texttt{COMBINE}\left(\mathbf{r}_v^{(k-1)},\ \texttt{AGGR}_1(\{\mathbf{r}_u^{(k-1)} : u \in \bar{N}_1(v)\}),\ \texttt{AGGR}_2(\{\mathbf{r}_u^{(k-1)} : u \in \bar{N}_2(v)\}), \ldots\right), \quad (3)$$

where $\bar{N}_i(v)$ denotes the neighbors of v at *exactly* i hops away, and the $\texttt{AGGR}_i$ functions applied to different neighborhoods can be the same or different. This design—first employed in GCN-Cheby [9] and MixHop [1]—augments the *implicit* aggregation over higher-order neighborhoods that most GNN models achieve through multiple layers of first-order propagation based on variants of Eq. equation 2. **(author?)** [58] attribute the effectiveness of this design to observations that even though the immediate neighborhoods may be heterophilous, the higher-order neighborhoods may show homophily in certain datasets (e.g., binary attribute prediction on 2-partite graphs [2, 7]) and thus provide more relevant context to GNNs.

Early implementations of this design, such as GCN-Cheby [9] and MixHop [1], extract embeddings from higher-order neighborhoods $\bar{N}_i(v)$ within each layer by employing "Delta Operators" [1]. These operators differentiate the aggregated embeddings in different orders of the (normalized) adjacency matrices $\mathbf{A}^i$ and $\mathbf{A}^{i-1}$ for improved computational efficiency. In contrast, H_2GCN [58], UGCN [13], TDGNN [39], and OrderedGNN [33] precisely compute the i-hop neighborhoods $\bar{N}_i(v)$ for each node v before applying the $\texttt{AGGR}_i$ functions to prevent mixing nodes from different hops. Notably, the recent approach by **(author?)** [33] achieves state-of-the-art classification accuracy on heterophilous datasets by modeling message passing within higher-order neighborhoods using a rooted-tree hierarchy, and aligning segments of variable length in the resulting node embeddings with specific neighborhood orders.

3.1.3 Combination of Intermediate Representations

The third design proposed in [58] combines the intermediate representations of each node at the final layer:

$$\mathbf{r}_v^{(\text{final})} = \texttt{COMBINE}\left(\mathbf{r}_v^{(1)}, \mathbf{r}_v^{(2)}, \ldots, \mathbf{r}_v^{(K)}\right). \quad (4)$$

This approach explicitly captures both local and global information using $\texttt{COMBINE}$ functions that process each representation individually, such as concatenation or LSTM-attention [45]. This design was initially introduced in jumping knowledge networks [45] and demonstrated to enhance the representation power of GCNs under *homophily*. Intuitively, each GNN layer gathers information with varying degrees of locality—earlier layers focus on local information, while later layers increasingly capture global information (implicitly, through propagation). Similar to D2 (which models explicit neighborhoods), this design models the distribution of neighbor representations in low-homophily networks more accurately. It also allows the class prediction to leverage different neighborhood ranges in different networks, adapting to their structural properties.

The application of this design is often linked to graph spectral theory: **(author?)** [58] provided a theoretical justification for this design from the perspective of graph spectral filtering. Building upon this foundation, GPR-GNN [6], FAGCN [3], and ACM [24] further enhance GNN performance under heterophily by developing additional graph filters and mixture mechanisms to utilize embeddings generated with varying frequency components at the final layer, in conjunction with this design.

3.1.4 Similarity-based Attention and Neighbor Discovery

The designs identified in [58] focus on boosting the effectiveness of message passing on heterophilous graphs without modifying the underlying structure. An alternative approach, however, is to go beyond the original graph adjacency and discover additional connections between the nodes in the graph, based on the similarity their original or latent features (e.g., structural embeddings), which replace or augment the original heterophilous structure of the graph in the message passing. Specifically, UGCN [13], SimP-GCN [14], NL-GNN [22], HOG-GCN [38] and GPNN [47] update the message-passing graph for GNNs by removing or downweighting the heterophilous edges in the original graph (i.e., edges that connect nodes with dissimilar features or structural embeddings), while introducing newly discovered connections that exhibit strong homophily. On the other hand, Geom-GCN [29] and WRGNN [36] leverage for each node both its original graph neighborhood and the derived "structural neighborhood" based on proximity of structural node embeddings in order to augment the message passing and aggregation process.

3.1.5 Signed Messages & Gated Kernel

In most GNN models [16, 37], messages are positively aggregated from neighbors and transformed using a single kernel or weight matrix. However, in heterophilous graphs, this may degrade GNN performance when messages from neighbors of different classes are mixed [58, 46]. Although attention-based GNNs, such as GAT [37], can theoretically reduce aggregation weights on heterophilous edges, they may still accumulate noise in the generated embeddings in practice.

An intuitive solution to address this issue is to learn signed messages (e.g., GGCN [46] and GReTo [54]) or gated kernels (e.g., GBK [10]) that separate message passing between homophilous intra-class and heterophilous inter-class edges. **(author?)** [46] suggested that ideally, messages from neighbors of a different class should be multiplied by a negative sign ("negative messages"), while messages from neighbors of the same class should remain unchanged. However, ground truth node labels are inaccessible in real scenarios, and any approximated sign function may introduce errors. To identify conditions when signed messages can enhance node classification performance, [46] introduced the concept of "error rate" that quantifies the portion of non-ideal messages and analyzed node classification performance under various error rates and homophily levels. The benefits of using signed messages can also be interpreted from the perspective of graph spectrum: signed messages allow negative mixture of certain frequency components [49, 6], helping models better capture high-frequency components in node features. This is especially beneficial for learning on heterophilous graphs as they contain abundant high-frequency components in their node features, unlike homophilous graphs [58].

From the perspective of practical model design, GGCN [46] and GReTo [54] used proximity between node features to approximate the sign function. As an alternative to signed messages, **(author?)** [10] proposed a gated bi-kernel design that applies separately to the message passing of homophilous and heterophilous edges, and adopted a learnable gate function to distinguish between the two types of edges based on the node features.

3.1.6 Degree Corrections

Zhu et al. [58] first noted that the performance divide between low- and high-degree nodes is exacerbated on heterophilous graphs (c.f. Figure 5). Later, **(author?)** [46] provided a thorough theoretical and empirical analysis of how the interplay of degrees and homophily levels affects the node classification accuracy. Specifically, two node-level properties were defined: relative degree $\bar{\theta}_u$, which evaluates the degree of a node compared to its neighbors' degrees; and node-level homophily h_u, which captures the tendency of a node to have the same class as its neighbors. Formally, the relative degree of a node u is defined as $\bar{\theta}_u \equiv \mathbb{E}_{\mathbf{A}|d_u}(\frac{1}{d_u}\sum_{v\in N_1(u)}\theta_{uv}|d_u)$, where $\theta_{uv} \equiv \sqrt{\frac{d_u+1}{d_v+1}}$; and the node-level homophily h_u is defined as $h_u \equiv \mathbb{P}(y_u = y_v | v \in N_1(u))$. The authors discovered that nodes with higher relative degrees outperform the

nodes with lower ones under certain conditions of node features when the design of signed messages (D5) is employed. To improve the performance of nodes with lower relative degrees, they proposed a degree correction strategy which learns to virtually increase the relative degree of the nodes via structure-based edge attention weights $\tau_{uv}^l = \texttt{softplus}\left(\lambda_0^l \left(\frac{1}{\theta_{uv}} - 1\right) + \lambda_1^l\right)$, where λ_0^l and λ_1^l are the learnable parameters at the l-th GNN layer. If θ_{uv} is small, a large τ_{uv}^l is learned, which compensates for the current relative degree.

3.2 Heterophily and Other Objectives of GNN Research

Numerous studies have demonstrated that tackling the limitations of GNNs under heterophily not only enhances their performance on heterophilous datasets, but also improves their properties in other aspects of GNN research. In this section, we provide an overview of the connections that have been investigated between heterophily and adversarial robustness, algorithmic fairness, and oversmoothing, all of which are also important for deployment.

Heterophily & Robustness. Recent works have shown that GNNs have a high sensitivity to adversarial attacks [60, 8, 44, 42, 17, 25]. While most previous works have focused on naturally-occurring heterophily, heterophilous interactions may also be introduced as adversarial noise: as many GNNs exploit homophilous correlations, they can be sensitive to changes that render the data more heterophilous. This relation between adversarial structural attacks and the change of homophily level was first suggested though empirical analyses on homophilous graphs [42, 15], and was later formalized by **(author?)** [55] with theoretical and additional empirical analyses. Specifically, **(author?)** [55] showed that on homophilous graphs, effective structural attacks lead to increased heterophily, while, on heterophilous graphs, they alter the homophily level contingent on node degrees: for low-degree nodes, attacks increasing the heterophily are still effective, but for high-degree nodes, attacks decreasing the heterophily will be effective. By leveraging these relations, the authors further demonstrated that some key architectural designs for effectively handling heterophily—separate aggregators for ego- and neighbor-embeddings (D1) and Combination of Intermediate Representations (D3)—also improve the robustness of GNNs against attacks. Following these relations, a follow-up work proposed a defense framework called CHAGNN that improves the robustness of GNNs against Graph Injection Attacks (GIA) by iteratively pruning the heterophilous edges in the graph and retraining the GNN model [59].

Heterophily & Fairness. Algorithmic fairness is a critical aspect of machine learning that ensures a model does not disproportionately underperform for certain input classes. In the context of link prediction in networks, fairness is desirable to prevent the prediction accuracy from being influenced by sensitive node attributes, such as race or religion in a social network context. To promote fairness in Graph Neural Networks (GNNs), previous research has suggested learning a fair reweighting or rewiring of the graph structure alongside the parameters of the GNN [18, 34]. Theoretical analysis has shown that the effectiveness of these approaches depends on the weights of the intergroup edges (essentially, heterophilous edges according to sensitive attribute), along with the group sizes and other structural attributes of the graph. For node classification, the global homophily ratio of a graph has revealed to be crucial in providing bounds for group fairness concerning a sensitive attribute [40]. Other research has examined GNN fairness with respect to local homophily ratios within individual node neighborhoods [23], revealing that variations in local homophily can impact model fairness, and that GNN designs for heterophily can empirically enhance group fairness.

Heterophily & Oversmoothing. The oversmoothing problem relates to the degenerated performance of GNNs with an increasing number of layers [19]. Though both the heterophily and oversmoothing problems are associated to the unsatisfactory performance of GNNs, they do not appear to be related at a first glance. However, evidence from both empirical [5, 6] and theoretical analysis [46, 4] has found that the two problems may share the same root causes and may be addressed with the same approaches. **(author?)** [5] addressed the oversmoothing problem

via initial residual and identity mapping, but their designs were found empirically to help improve the node classification performance on heterophilous graphs. Vice versa, **(author?)** [6] addressed the heterophily problem via generalized PageRank, but they showed that their designs are also effective for addressing the oversmoothing problem. **(author?)** [46] are the first to explicitly analyze the relationship between the two problems. They found that the two problems can be jointly explained by analyzing the changes in the node representations over the layers, and proposed two designs, namely signed messages (D5) and degree corrections (D6), to address the two problems jointly. Later, **(author?)** [4] used cellular sheaves theory to explain the two problems jointly. They found that the underlying geometry of the graph is related to the performance of GNNs in heterophilous settings and their oversmoothing behavior, and many GNNs implicitly assume a graph with a trivial underlying sheaf. These observations and analyses have shown promising results in addressing the two problems jointly, which is an interesting direction to explore further.

4 Revisiting When is Heterophily Challenging for GNNs

While many works have focused on designing new GNN models with improved performance under heterophily, few of them have probed whether heterophily persistently presents challenges for GNNs. Some of these works have found that GNNs without the aforementioned heterophilous designs (e.g., SGC [41], GCN [16], GAT [37]) can exhibit better or equivalent performance to GNNs possessing such designs on certain datasets [26, 24]. In this section, we first summarize the main findings of these works, and show that the complexity of heterophily can be measured based on the distinguishability of the Neighborhood Label Distributions (NLDs).[1] We then highlight two key factors, low-degree nodes and complex compatibility matrices, which deteriorate the distinguishability of the neighborhood label distributions when coupled with heterophily, thus making heterophily a unique challenge for GNNs in most cases.

4.1 Improved Measures for Complexity of Heterophily

While many works measure the level of homophily/heterophily by the ratio of edges that connect nodes with the same class label (e.g., edge homophily in Dfn. 2.1, node homophily [29], or class homophily [21]), recent works have shown that graphs with high heterophily are not always challenging for GNNs without heterophilous designs. Through independent analyses, **(author?)** [26] and **(author?)** [24] arrive at the conclusion that the complexity of heterophily is closely related to the distinguishability of the neighborhood label distributions, which we define next.

Definition 4.1 (Neighborhood Label Distribution (NLD)) *Given $\mathbf{Y}$ as the label encoding matrix defined in §2 for nodes $\mathcal{V}$ in graph $\mathcal{G}$, the neighborhood label distribution of node v is defined as $\mathbf{D}(v) = \frac{1}{|N_1(v)|}\sum_{u\in N_1(v)} \mathbf{Y}_u$, where $\mathbf{Y}_u = \text{onehot}(y_u)$ is the v-th row of the label encoding matrix $\mathbf{Y}$.*

We now rephrase the two metrics proposed by **(author?)** [26] and **(author?)** [24] with the above definition, both of which measure the complexity of heterophily by quantifying the distinguishability of $\mathbf{D}(v)$.

Definition 4.2 (Class Neighborhood Similarity (CNS) [26]) *The class neighborhood similarity between classes $i, j \in \mathcal{Y}$ is defined as the average cosine similarity between the NLDs $\mathbf{D}(v), \mathbf{D}(u)$ of nodes v, u in class i and j, respectively, i.e.,*

$$S(i,j) = \frac{1}{|\mathcal{V}_i||\mathcal{V}_j|}\sum_{v\in\mathcal{V}_i}\sum_{u\in\mathcal{V}_j} \text{sim}_{\cos}(\mathbf{D}(v), \mathbf{D}(u)), \tag{5}$$

[1] In parallel with these studies, **(author?)** [46] conducted a theoretical analysis of performance degradation in heterophilous networks under the "non-swapping" condition. This condition emerges when the neighboring representations for each node are insufficient to cause the interchange of node representations from two distinct classes across their separation plane in the latent space. Conversely, the case of "easy heterophily" [26, 24] that we address in this section corresponds to the "swapping" condition as articulated in [46].

where $\mathcal{V}_i$ and $\mathcal{V}_j$ are the sets of nodes with class label i and j, and $\text{sim}_{\cos}(\cdot)$ is the function of cosine similarity. We refer to the case of $i = j$ as intra-class neighborhood similarity (intra-CNS) and the case of $i \neq j$ as inter-class neighborhood similarity (inter-CNS).

Definition 4.3 (Graph Aggregation Homophily [24]) *Define the average similarity score of a node $v \in \mathcal{V}$ to nodes $\mathcal{V}_i$ with class label $i \in \mathcal{Y}$ as $g(v, i) = \text{mean}\left(\{\text{sim}(\mathbf{D}(v), \mathbf{D}(u)) : v, u \in \mathcal{V}, y_u = i\}\right)$, where sim is a function (e.g., dot product) that measures the similarity between two neighborhood label distributions. The graph aggregation homophily is then defined as the ratio of nodes $v \in \mathcal{V}$ where the neighborhood label distribution $\mathbf{D}(v)$ is more similar for nodes in the same class than for nodes in any other class, i.e.,*

$$h_{\text{agg}} = \frac{1}{|\mathcal{V}|} \left| \left\{ v \in \mathcal{V} : g(v, y_v) \geq \max_{j \neq y_v \in \mathcal{Y}} g(v, j) \right\} \right|. \tag{6}$$

We note that while h_{agg} measures the *proportion* of nodes with NLD exhibiting greater similarity (regardless of the extent) to nodes within the same class compared to nodes from different classes, it does not quantify the *degree of similarity* between NLDs of nodes within the same class or across different classes, which is captured by the CNS metric. Consequently, as we show in our empirical analysis in §4.2.4, CNS provides a more comprehensive and accurate assessment of the complexity of heterophily on synthetic datasets, and thus we focus on CNS in our empirical analysis below.

4.2 Factors Determining the Complexity of Heterophily

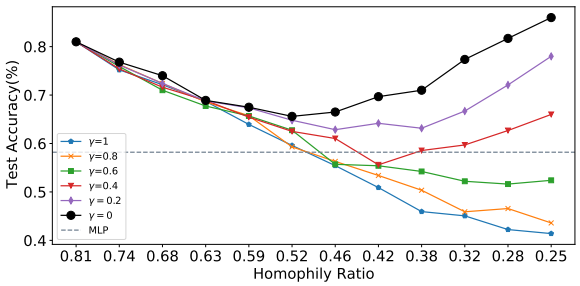
It has been shown that it is possible to have graphs with high level of heterophily but low complexity for GNNs as measured by CNS or aggregation homophily [26, 24]: when nodes in the same class have strong similarity with respect to neighborhood label distributions, and nodes from different classes have weak or no similarity, GCN models are able to perform well due to the high distinguishability of the neighborhood label distributions, even when the graphs are heterophilous. These are important findings, but this type of analysis does not provide a complete picture of the complexity of heterophily for GNNs, as the high distinguishability of the class label distributions under heterophily is largely dependent on key graph properties, such as degree distributions and the compatibility matrices that drive the generation of the graph. In this section, we provide a detailed analysis of the above two factors that determine how challenging the data heterophily is for GNNs.

4.2.1 Motivating Example: Differences in Synthetic Datasets

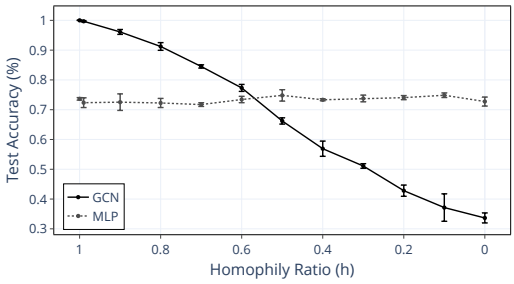
Prior research exploring the impact of heterophily on GNN performance frequently incorporates experiments on synthetic datasets with controlled homophily/heterophily levels [1, 58, 26, 24]. In line with this research, in this section we provide a motivating example based on synthetic data that showcases the role of the two factors—namely, degree distribution and compatibility matrices—in characterizing how challenging heterophily is for GNN models. We analyze the seemingly contradictory observations arising from the results on two distinct synthetic datasets based on the `Cora` dataset: `syn-cora` [58] and `necessity-cora` [26].

Before diving into the analysis of the factors that affect the complexity of heterophily, we first provide a brief overview on the setup and key results on the two synthetic datasets [58, 26].

Data Generation: `syn-cora` vs. `necessity-cora`. While both synthetic datasets are generated based on `Cora`, their generation processes are largely different. The `syn-cora` dataset [58, 57] follows a modified preferential attachment process. In this process, the probability of a new node u with class label c to attach to existing node v with class label c' is proportional to: (1) the ratio $\mathcal{D}_{c,c'}$ specified in the underlying compatibility matrix $\mathcal{D}$, which determines the homophily level in the resulting graph, as empirically measured by the edge homophily ratio h (Dfn. 2.1) and the compatibility matrix $\mathbf{H}$ (Dfn. 2.3), and (2) the degree d_v of the existing



(a) Accuracy on `necessity-cora` under different noise levels γ. Figure is reproduced from [26]; shared with permission by the authors.



(b) Accuracy on `syn-cora` for GCN and MLP reported by [58]. Figure is adapted from [58].

Figure 3: Semi-supervised node classification accuracy of GCN and MLP observed in [26] and [58] on `necessity-cora` and `syn-cora`, respectively, under increasing level of heterophily (i.e., decrease of the edge homophily ratio h).

node v. This process results in a power-law degree distribution in the generated graph. On the other hand, `necessity-cora` [26] varies the level of homophily by adding heterophilous (cross-label) edges on top of the existing (homophilous) edges in `Cora`. To control the randomness of the added heterophilous edges, `necessity-cora` adds: (1) non-random heterophilous edges based on an underlying compatibility matrix $\mathcal{D}$, and (2) random edges that do not follow the underlying compatibility matrix $\mathcal{D}$, but are controlled by a noise parameter γ.

Observations: `syn-cora` vs. `necessity-cora`. When the level of heterophily is varied, largely different observations are reported on the two sets of synthetic graphs with respect to the GNN performance: [26] shows that on synthetic graphs with none or few randomly-added heterophilous connections (i.e., with the noise parameter γ close to 0), the performance of GCNs can even increase as the level of heterophily in the graph gets stronger (i.e., when the edge homophily ratio h decreases), as shown in Figure 3(a); on the other hand, [58] shows that the performance of GCNs significantly decreases as the heterophily increases, which we show in Figure 3(b). As our follow-up analysis below shows, these seemingly contradictory results are due to the different processes used to generate the synthetic graphs, which lead to very different graph properties (i.e., degree distribution, class compatibility matrix); these in turn affect the model performance. We analyze the effects of the (F1) degree distribution in §4.2.2 and the (F2) compatibility matrix in §4.2.3.

4.2.2 Factor (F1): Degree Distributions & Heterophily

In §4.1, we revisit the findings from recent works [26, 24] that the complexity of heterophily for GNNs is largely determined by the distinguishability of the Neighborhood Label Distributions (NLDs) of nodes with different class labels. Under the generation process of `necessity-cora` with noise $\gamma = 0$ (§4.2.1), when classes are different $c \neq c'$ and the distributions $\mathcal{D}_c$ and $\mathcal{D}_{c'}$ are distinguishable from each other, one would state that the GCN models can perform well to distinguish the nodes with class label c from the nodes with class label c' from the perspective of NLD distinguishability.

However, we argue that the aforementioned statement ignores the critical factor of degree for each node $v \in \mathcal{V}_c$ that impacts the quality of the samples of the distribution $\mathcal{D}_c$: when all the nodes have sufficiently large degrees, it is expected that $\mathcal{D}_c$ can be recovered well in the node neighborhoods due to sufficient samples of the distributions; however, when many low-degree nodes are present in the graph (which is the case for many

real-world graphs, which usually follow power-law degree distributions), $\mathcal{D}_c$ may not be consistently recovered in the neighborhood of the low-degree nodes under heterophily due to the insufficiency of the samples. This affects the intra-class and inter-class similarity of the NLDs in heterophilous settings.

Empirical Analysis. We can further explain how low-degree nodes affect the intra-class and inter-class distinguishability of the NLD for GCNs under heterophily with a simple empirical analysis: Suppose the neighborhood label distributions $\mathcal{D}_c$ and $\mathcal{D}_{c'}$ for two classes $c \neq c'$ are given in the red dashed boxes in Figure 4. Following the distributions $\mathcal{D}_c$ and $\mathcal{D}_{c'}$, we randomly generate the NLDs of 200 nodes with degree 2 for both classes c and c'; then, we sample the labels of their 2 neighbors, and we visualize a random set of 5 of the 200 synthetic NLDs in Figure 4(a)-(b), for $\mathcal{D}_c$ and $\mathcal{D}_{c'}$, respectively. We note that since GCN aggregators additionally consider a self-loop for each node, the NLD observed by GCN models should be considered with self-loops added to the graphs, even when $\mathcal{D}_c$ and $\mathcal{D}_{c'}$ dictate purely heterophilous connections. To visualize the contributions of self-loops in the NLDs, we show them in gray in Figure 4.

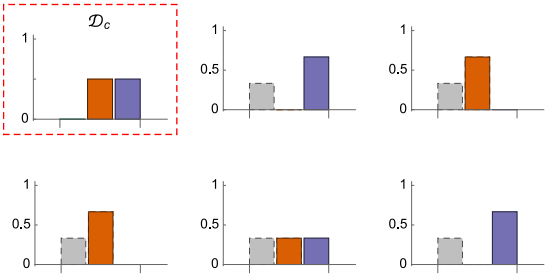
- *Case 1: Low-degree nodes & heterophily.* Figure 4(a)-(b) show that the existence of low-degree nodes reduces the distinguishability of the NLDs. Specifically, we observe that: (1) the intra-class NLDs have a high variance and can be very different from the corresponding ground-truth distributions (even when not considering the self-loops). In fact, the mean and standard deviation of the intra-class pairwise cosine similarity among the 200 synthetic neighborhoods of class c and c' are 0.79 ± 0.24, where the high standard deviation reflects the strong variance among the sampled neighborhood distributions. (2) Many of the NLDs from nodes in class c, c' are the same when considering the self-loops, which affects their distinguishability across different classes; the inter-class pairwise cosine similarity for our synthetic neighborhoods of class c and c' is 0.79 ± 0.17 in our analysis, which is the same as the intra-class pairwise similarity with even smaller standard deviation.

- *Case 2: High-degree nodes & heterophily.* On the other hand, when the node degrees are high, the NLDs are more similar to the underlying distributions $\mathcal{D}_c$ and $\mathcal{D}_{c'}$ (even with self-loops considered) and thus have much smaller variances. In our example, we randomly sampled NLDs of another 200 nodes with degree 10 (instead of 2), and illustrate them for 5 randomly selected nodes in Figure 4(c)-(d). The mean and standard deviation of the pairwise cosine similarity among the 200 generated neighborhoods of class c and c' are 0.93 ± 0.09, while the inter-class pairwise similarity is only 0.64 ± 0.15. These changes in intra-class and inter-class similarities can also be observed in the sampled distributions shown in Figure 4(c)-(d).

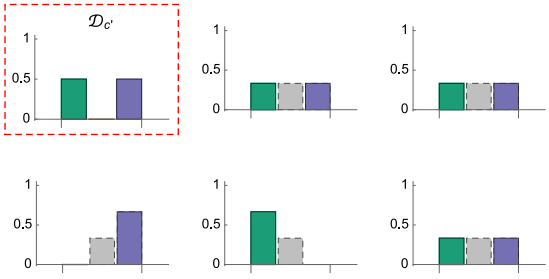
- *Case 3: Low- / high-degree nodes & strong homophily.* We note that the presence of low-degree nodes does not affect the similarity of NLDs as much in strong homophilous settings as in the heterophilous settings. To show this empirically, we similarly generate the neighborhood label distributions of 200 nodes with degrees of 2 for class c, c', but this time with distributions $\mathcal{D}_c$ and $\mathcal{D}_{c'}$ showing strong homophily. In Figure 4(e)-(f), we observe that, unlike the heterophilous settings, almost all synthetic distributions of nodes from the same class c (or c') are close to the expected distribution $\mathcal{D}_c$ (or $\mathcal{D}_{c'}$); most neighbors (considering self-loops) have the same class label c (or c') as the ego node, even for low-degree nodes. Numerically, the intra-class pairwise cosine similarity among the synthetic neighborhoods of class c and c' is 0.88 ± 0.15 and 0.91 ± 0.13, respectively. On the other hand, the inter-class pairwise similarity is 0.21 ± 0.29, which shows good separability. This example shows that **the presence of low-degree nodes is a challenge that is more pronounced in heterophilous settings than in homophilous settings.**

Summary & connections to other works. From the above analysis, we see that **the existence of low-degree nodes can lead to weak distinguishability of inter-class NLDs, thus affecting the performance of GCNs**. The significant performance gap between low-degree and high-degree nodes is also observed in [58], as shown

Case 1: Low-degree nodes & heterophily

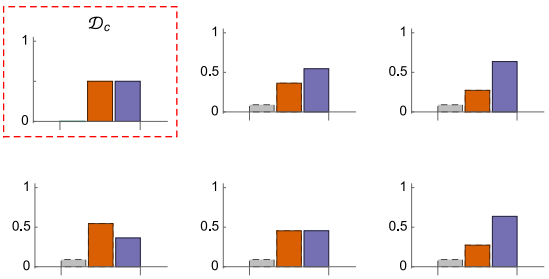


(a) Heterophilous distribution $\mathcal{D}_c$ of class c (in red box) and 5 sampled NLDs for nodes with degree 2.

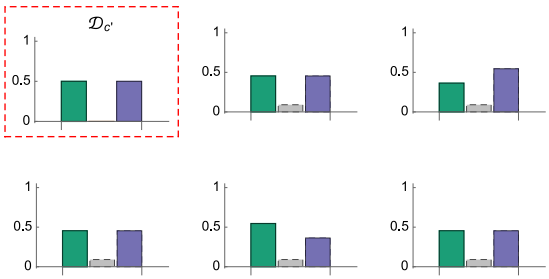


(b) Heterophilous distribution $\mathcal{D}_{c'}$ of another class c' (in red box) and 5 sampled NLDs for nodes with degree 2.

Case 2: High-degree nodes & heterophily

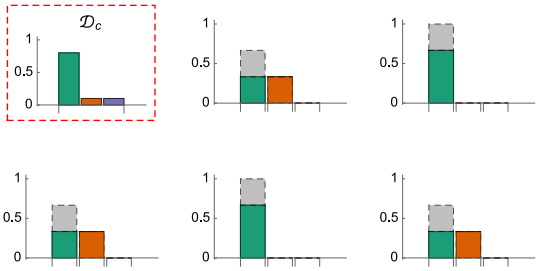


(c) Heterophilous distribution $\mathcal{D}_c$ of class c (in red box) and 5 sampled NLDs for nodes with degree 10.

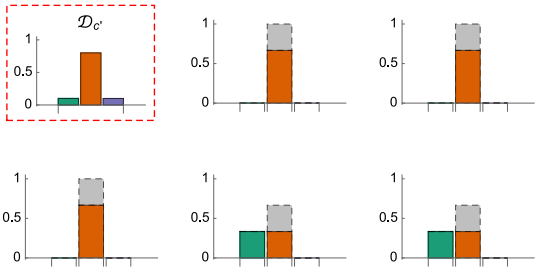


(d) Heterophilous distribution $\mathcal{D}_{c'}$ of another class c' (in red box) and 5 sampled NLDs for nodes with degree 10.

Case 3: Low-degree nodes & homophily

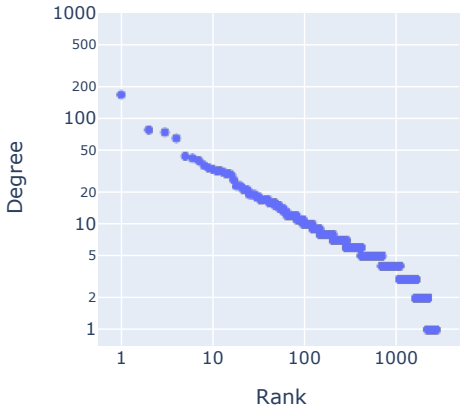


(e) Homophilous distribution $\mathcal{D}_c$ of class c (in red box) and 5 sampled NLDs for nodes with degree 2.

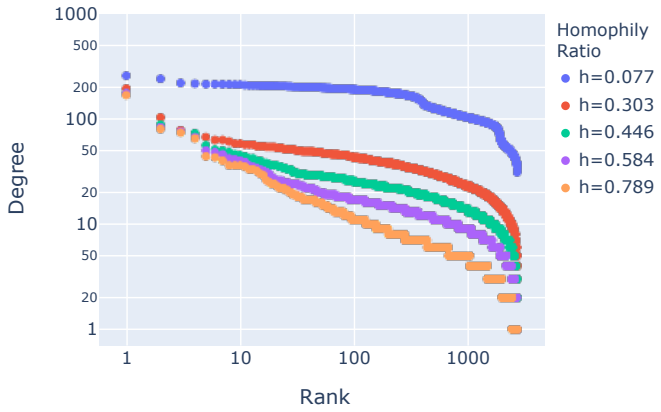


(f) Homophilous distribution $\mathcal{D}_{c'}$ of another class c' (in red box) and 5 sampled NLDs for nodes with degree 2.

Figure 4: (Factor F1) Degree Distributions: Per case, we sample 200 NLDs from distribution $\mathcal{D}_c$ (and $\mathcal{D}_{c'}$) for nodes with specific degrees, and visualize 5 sampled NLDs. The gray parts correspond to the contributions of self-loops in the NLDs aggregated by GCN. **(a)-(b) Case 1:** Low degrees reduce the distinguishability of NLDs: for all synthetic NLDs of c and c', inter-CNS is $S(c, c') = 0.79 \pm 0.17$, with even smaller standard deviation than intra-CNS $S(c, c) = S(c', c') = 0.79 \pm 0.24$. **(c)-(d) Case 2:** Higher node degrees improve the distinguishability of NLDs: for all synthetic NLDs of c and c', inter-CNS is $S(c, c') = 0.64 \pm 0.15$, which is smaller than the intra-CNS $S(c, c) = S(c', c') = 0.93 \pm 0.09$. **(e)-(f) Case 3:** Low node degrees matter less under homophily: for all synthetic NLDs of c and c', inter-CNS is $S(c, c') = 0.21 \pm 0.29$, which is significantly smaller than the intra-CNS $S(c, c) = 0.88 \pm 0.15$ and $S(c', c') = 0.91 \pm 0.13$.



(a) Degree distribution for `cora`: it follows a typical power-law degree distribution.



(b) Degree distribution of `necessity-cora`, the cora-based synthetic graphs in [26] with $\gamma = 0$ and edge homophily ratio $h \in \{0.077, 0.303, 0.446, 0.584, 0.789\}$.

Figure 6: Degree distributions of `cora` and `necessity-cora`. As the level of heterophily increases (i.e., edge homophily ratio h decreases), the degrees for all the nodes increase in `necessity-cora`, and the degree distributions move further away from the original degree distribution of `cora`. The shift in degree distribution explains the increase of GCN performance with the level of heterophily for the $\gamma = 0$ case in Figure 3(a).

in Figure 5. As a follow-up to our analysis[2], **(author?)** [26] formalized the effects of node degrees on the distinguishability of NLDs for Contextual Stochastic Block Model (CSBM), and derived a lower bound of node degrees for GCN-style aggregation to improve the distinguishability of NLDs.

Revisiting the `necessity-cora` dataset. The degree distributions of `necessity-cora` also explain why GCN performance starts to increase as the level of homophily h decreases in the range of $h < 0.5$ (for noise $\gamma < 0.5$): the `necessity-cora` graphs with homophily ratio $h < 0.5$ have a significantly higher average degree compared to their corresponding base graph `cora`, as a large amount of edges needs to be added in order to decrease the edge homophily ratio in the (strongly homophilous) base graph. In Figure 6, we show the degree distribution of base graph `cora` in comparison to the degree distributions of the `necessity-cora` graphs with $\gamma = 0$ (i.e., when all the heterophilous edges are added according to the underlying compatibility matrix $\mathcal{D}$, without any randomness) for varying edge homophily ratio h. We see that as h decreases, the degrees for all nodes in the graph increase, and the degree distributions move further away from the degree distribution of `cora`; for the $h = 0.077$ instance, even the minimum node degree in the `necessity-cora` graph has exceeded the degree of most nodes in `cora`. In our additional empirical analysis (§4.2.4), we show that the lack of low-degree nodes is indeed a necessary condition that contributes to the observed high performance of GCNs on `necessity-cora`.

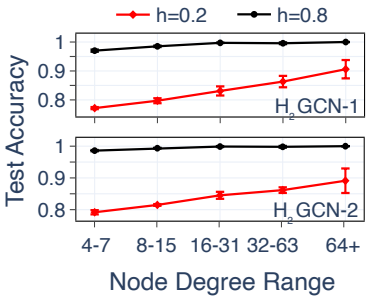
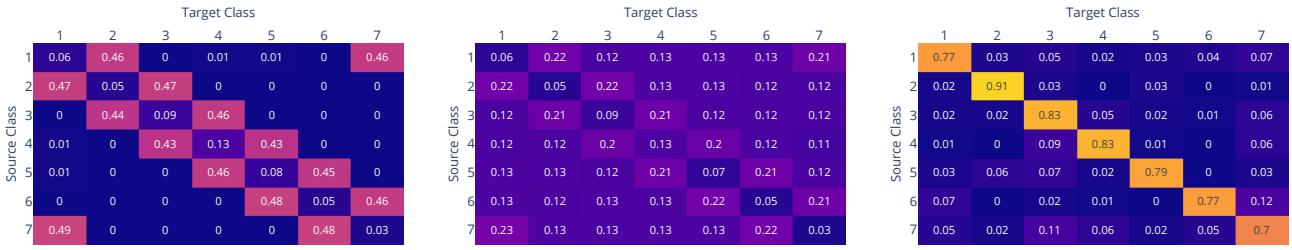


Figure 5: H_2GCN accuracy per degree range on synthetic heterophilous ($h = 0.2$) and homophilous ($h = 0.8$) graphs. Figure from [58].

[2]These analyses were first made available in the form of a blog post: Zhu, J. and Koutra, D. (2021) Revisiting the problem of heterophily for GNNS. Available at: https://www.jiongzhu.net/revisiting-heterophily-gnns/.



(a) Compatibility matrix with $\gamma = 0$ and $h = 0.16$ in `necessity-cora`.

(b) Compatibility matrix with $\gamma = 0.8$ and $h = 0.16$ in `necessity-cora`.

(c) Compatibility matrix of `cora` with strong homophily.

Figure 7: Comparison of compatibility matrices $\mathbf{H}$ of different synthetic graphs in `necessity-cora` with homophily ratio $h = 0.16$ but different noise ratio γ, with comparison to the compatibility matrix of `cora` (the homophilous graph which `necessity-cora` is based on).

4.2.3 Factor (F2): Compatibility Matrices & Heterophily

Another factor that affects the distinguishability of NLDs is the distinguishability of the compatibility matrices for different classes: Under the generation process of `necessity-cora` (§4.2.1), when the node degrees are sufficiently high in the generated graphs (Case 2 of §4.2.2), the NLDs for nodes $v \in \mathcal{V}_c$ are expected to be similar to $\mathcal{D}_c$. In this case, the distinguishability of NLDs between nodes in class c and c' mostly depends on the distinguishability of $\mathcal{D}_c$ and $\mathcal{D}_{c'}$, which can also be observed empirically in the compatibility matrices $\mathbf{H}$ of the generated graphs. In this section, we discuss how complex compatibility patterns in the rows of $\mathbf{H}$ can reduce the distinguishability of NLDs and contribute to the complexity of heterophily, in addition to (F1) node degrees.

Observations on heterophilous datasets: `necessity-cora` under different noise levels. The differences in the observed performance of GCN on `necessity-cora` under different noise levels γ (i.e., randomness of the heterophilous edges) in Figure 3(a) can be explained by the differences in the distinguishability of the empirical compatibility matrices $\mathbf{H}$ (and the underlying compatibility matrices $\mathcal{D}$ by extension). In Figure 7, we visualize the compatibility matrices of graphs from `necessity-cora` with homophily ratio $h = 0.16$, and compare between graphs with noise levels $\gamma = 0$ and $\gamma = 0.8$: (1) when $\gamma = 0$, the compatibility matrices in `necessity-cora` are formulated to resemble a "loop", where almost all connections for nodes of class c are limited to the two adjacent classes in the "circle" of classes (e.g., nodes in Class 2 almost exclusively connect to nodes in Class 1 and 3 in Figure 7(a)). This "loop"-pattern helps maintain the high distinguishability of compatibility patterns among different classes, and thus provides an easier node classification problem for GCNs compared to more general heterophilous patterns. (2) In comparison, when $\gamma = 0.8$, the heterophilous connections of class c are distributed to all classes $c' \neq c$, and the rows $\mathbf{H}_c$ of the compatibility matrix are less distinct, which is similar to the case of `syn-cora` (c.f. Figure 8(h)). The high similarity of $\mathbf{H}_c$ among different classes makes it challenging to distinguish different classes from the NLDs even for high-degree nodes, as many heterophilous connections from class c are uniformly distributed to other classes $c' \neq c$. This explains the decrease of GCN accuracy under the same homophily level (and degree distribution[3]) in `necessity-cora` when γ increases, as shown in Figure 3(a).

Observations on homophilous datasets. We also note that, echoing the observation in [26], the **distinguishability** of the rows in compatibility matrix $\mathbf{H}$ **is guaranteed for graphs with strong homophily**, as the largest entries in the distributions are concentrated on the diagonal elements of the compatibility matrix $\mathbf{H}$ as shown

[3]Graphs with the same edge homophily ratio h in `necessity-cora` also have highly similar degree distributions by extension, since the level of homophily is varied by edge addition.

in Figure 7(c). **Thus, the distinguishability of the compatibility matrices is also a challenge specific to heterophilous settings.**

4.2.4 The Interplay of Degree Distribution, Compatibility Matrices & NLDs

In Sections §4.2.2–4.2.3, we discussed two key factors that determine how challenging heterophily is for GNN models. Here, we explore the interplay of these factors and NLDs via an empirical study. To this end, we construct additional synthetic data with properties complementary to those in [58, 26].

Data generation: "loop"-style schema & power-law degree distribution. To study the interplay of factors (F1) & (F2), we generate synthetic graphs which have: (1) (mostly) "loop"-style compatibility matrices (where nodes in each class only connect to nodes in its nearby classes, as if all classes are arranged in a circle), e.g., Figure 7(a). This schema is similar to that used for `necessity-cora`; we leverage the same or similar compatibility matrices as specified in [26] in the generation process. (2) the same power-law degree distribution as `syn-cora` by following the same modified preferential attachment generation process as in [58, 57].

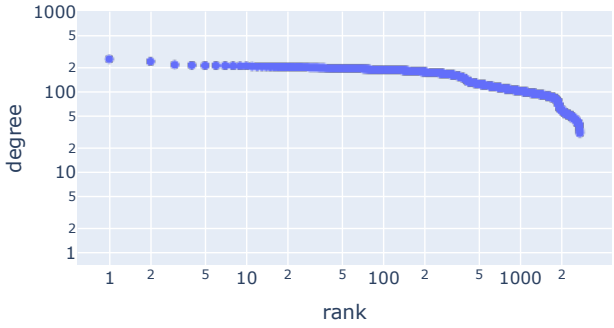
We refer to these synthetic graphs as `syn-cora-loop`, and consider two variants: `syn-cora-loop-7` with 7 classes as in `necessity-cora`, and `syn-cora-loop-5` with 5 classes as in `syn-cora`. In Figure 8, we visualize the degree distributions and compatibility matrices of our `syn-cora-loop` datasets, along with the visualizations for `necessity-cora` and `syn-cora`.

Models. We assess the influence of degree distributions and compatibility matrices on the performance of three GNN models: H_2GCN [58], GCN [16], and MLP. H_2GCN represents GNN models that incorporate one or more heterophilous designs as discussed in §3.1; we examine two variants of H_2GCN, namely H_2GCN-1 and H_2GCN-2, with one or two layers of aggregation respectively. In contrast, GCN serves as the GNN baseline model which does not incorporate any heterophilous designs, while MLP functions as the graph-agnostic baseline that does not consider the graph structure. For GCN, we adopt the same hyperparameter tuning as in [26], and further tune the dimension of hidden embeddings between 16 and 64. For H_2GCN, we only tune a subset of the hyperparameters that we tune for GCN (16 vs. 112 combinations), which are more hyperparameter combinations than those explored in [58]. For each model, we present the mean and standard deviation of the classification accuracy under five runs with different random seeds per dataset.

Data setup. Our experiments incorporate four sets of synthetic graphs: `necessity-cora` provided by **(author?)** [26], `syn-cora` from [58], and the newly generated `syn-cora-loop-7` and `syn-cora-loop-5`. For `syn-cora`, we select the graph with homophily level $h = 0$; for `necessity-cora`, we select the graph with noise parameter $\gamma = 0$ and h nearest to 0 (i.e., $h = 0.03$) as permitted by its generation process. We generate `syn-cora-loop-7` and `syn-cora-loop-5` with $h = 0$. In Table 1, we present the statistics for each dataset; the degree distributions and compatibility matrices for all datasets are visualized in Figure 8. For the train/validation/test splits, we utilize the provided splits for `necessity-cora` [26], and create splits for the other datasets using identical sizes as in `necessity-cora`. Specifically, we randomly select 20 nodes per class for the training set, 500 nodes throughout the graph for the validation set, and allocate the remaining nodes to the test set[4].

GNN performance & graph properties. In Table 1, we list the performance of each model along with the corresponding properties of the graphs. We observe the effects of the interplay between low-degree nodes and more complex compatibility matrices to the performance of GNN models when the graphs share similar edge homophily ratio h (0 or as close to 0 as the generation process allows):

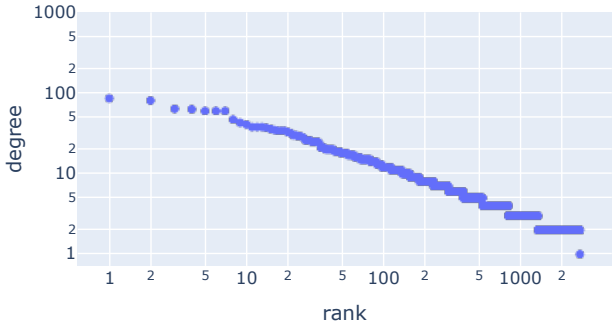
[4]This setup is identical to [16], but differs from [58, 57] (where Figure 3(b) was generated), which utilized a larger training set.



(a) Degree distribution of `necessity-cora` ($\gamma = 0$).

0.03	0.48	0.0	0.0	0.0	0.0	0.48
0.48	0.02	0.5	0.0	0.0	0.0	0.0
0.0	0.46	0.04	0.49	0.0	0.0	0.0
0.0	0.0	0.46	0.06	0.47	0.0	0.0
0.0	0.0	0.0	0.5	0.03	0.47	0.0
0.0	0.0	0.0	0.0	0.51	0.02	0.46
0.5	0.0	0.0	0.0	0.0	0.48	0.01

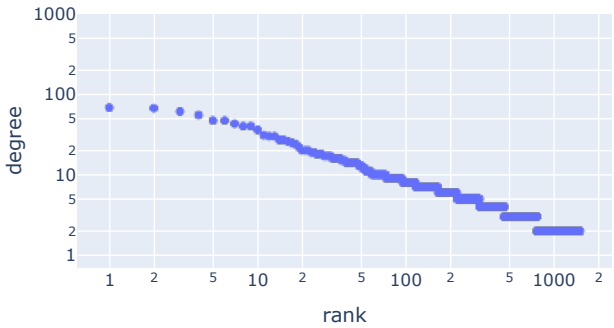
(b) Compatibility matrix of `necessity-cora` ($\gamma = 0$).



(c) Degree distribution of `syn-cora-loop-7`.

0.0	0.55	0.0	0.0	0.0	0.0	0.45
0.5	0.0	0.5	0.0	0.0	0.0	0.0
0.0	0.3	0.0	0.7	0.0	0.0	0.0
0.0	0.0	0.48	0.0	0.52	0.0	0.0
0.0	0.0	0.0	0.68	0.0	0.32	0.0
0.0	0.0	0.0	0.0	0.67	0.0	0.33
0.57	0.0	0.0	0.0	0.0	0.43	0.0

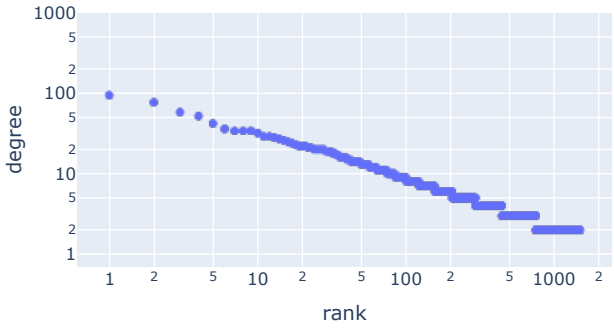
(d) Compatibility matrix of `syn-cora-loop-7`.



(e) Degree distribution of `syn-cora-loop-5`.

0.0	0.48	0.0	0.0	0.52
0.5	0.0	0.5	0.0	0.0
0.0	0.51	0.0	0.49	0.0
0.0	0.0	0.49	0.0	0.51
0.52	0.0	0.0	0.48	0.0

(f) Compatibility matrix of `syn-cora-loop-5`.



(g) Degree distribution of `syn-cora`.

0.0	0.33	0.19	0.16	0.33
0.33	0.0	0.35	0.14	0.18
0.19	0.35	0.0	0.29	0.17
0.18	0.15	0.32	0.0	0.35
0.33	0.18	0.17	0.32	0.0

(h) Compatibility matrix of `syn-cora`.

Figure 8: Synthetic networks used to study the interplay of factors (F1) and (F2). `syn-cora-loop` datasets have the "loop"-style structure of `necessity-cora` graphs and the power law degree distribution of the `syn-cora` graphs.

Table 1: Dataset statistics and effectiveness of models for node classification. We report the min, median, and max values for the intra-class and inter-CNS, and the mean accuracy ± standard deviation for each model. The best result for each dataset is highlighted in blue.

	necessity-cora	**syn-cora-loop-7**	**syn-cora-loop-5**	**syn-cora**
#Nodes	2,708	2,708	1,490	1,490
#Edges	132,196	5,394	2,968	2,968
# Classes	7	7	5	5
Edge Hom. h	0.03	0	0	0
(F1) High-degree Nodes Only	✓	✗	✗	✗
(F2) Compatibility "Loop"	✓	✓	✓	✗
Heterophily Type	Easy	Challenging	Challenging	Most challenging
Agg. Hom. h_{agg}	1.00	0.90	1.00	0.41
Intra-CNS (min/median/max)	0.97/0.99/1.00	0.79/0.82/0.84	0.78/0.80/0.80	0.62/0.63/0.64
Inter-CNS (min/median/max)	0.00/0.08/0.52	0.00/0.31/0.57	0.30/0.39/0.47	0.53/0.57/0.60
H_2GCN-2	99.26 ± 0.28	88.45 ± 1.26	87.98 ± 1.49	68.95 ± 1.88
H_2GCN-1	93.48 ± 0.93	80.85 ± 1.69	82.40 ± 1.77	66.82 ± 2.13
GCN	100.00 ± 0.00	65.10 ± 1.80	59.26 ± 2.17	27.27 ± 1.72
MLP	59.16 ± 0.52	58.20 ± 2.05	64.16 ± 1.61	63.84 ± 2.17

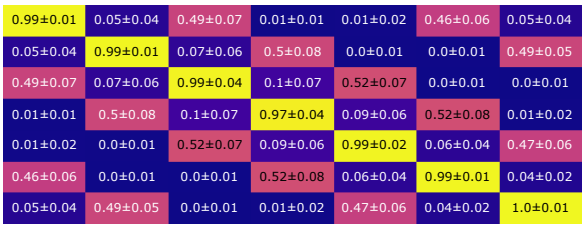
(1) On `necessity-cora`, with no low-degree nodes and the simpler "loop"-style compatibility matrices (Figure 8(a)-(b)), models like GCN and H_2GCN-2 can achieve near-perfect accuracy.
(2) On `syn-cora-loop` variants, where we keep the "loop"-style compatibility matrices but modify the degree distributions to follow a power law, we observe 34.90% to 40.74% decrease in accuracy for GCN, which falls below the accuracy of H_2GCN. As we discussed in §4.2.2, this heterophilous case is challenging; this is also confirmed by the performance drop for the graph-aware methods, including H_2GCN.
(3) On `syn-cora`, which further strips the "loop"-style compatibility matrices for heterophilous connections and has more complex connectivity patterns across different classes, the performance of GCN further decreases by 31.99% and falls much below the performance of the graph-agnostic MLP in this case; though the accuracy of H_2GCN variants also decreases significantly in this challenging case, they still outperform MLP in this case.

NLD distinguishability & graph properties. The significant changes in the accuracy of GCN can also be explained by the changes in the distinguishability of NLDs caused by different graph properties. In Table 1, we report the Class Neighborhood Similarity (CNS) and Graph Aggregation Homophily h_{agg} for the all synthetic graphs (as defined in §4.1, where we consider self-loops in accordance with GCN aggregation). We also visualize the CNS and its standard deviation (following Eq. equation 5) between pairs of classes on each dataset in Figure 9.

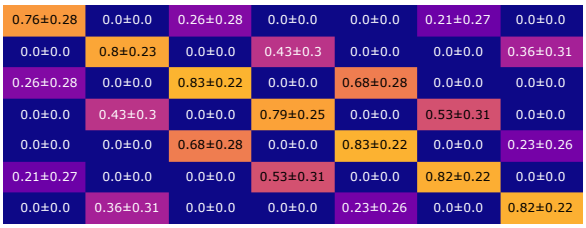
Based on the intra-class and inter-CNS in Table 1, we observe that:

(1) With the presence of low-degree nodes, the `syn-cora-loop` variants have reduced intra-CNS with higher variances compared to `necessity-cora`, while the inter-CNS also increases, though they share similar "loop"-style compatibility matrices;

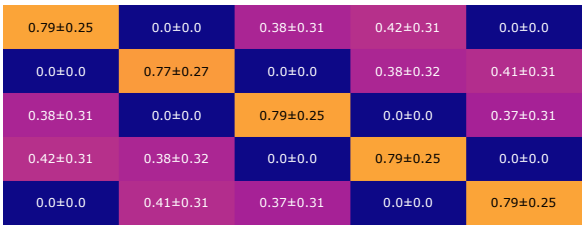
(2) The removal of the "loop" pattern in the compatibility matrices of `syn-cora` further reduces the intra-CNS to a level similar to the inter-CNS, which leads to weak distinguishability of the neighborhood label distributions



(a) CNS of `necessity-cora` with $\gamma = 0$.



(b) CNS of `syn-cora-loop-7`.



(c) CNS of `syn-cora-loop-5`.

(d) CNS of `syn-cora`.

Figure 9: Class neighborhood similarities (CNS) of the synthetic datasets in Table 1.

between nodes from different classes. These observations explain the decrease of GCN performance observed in our experiments, and show how that the distinguishability of the neighborhood label distributions can depend on other properties like degree distributions and class compatibility matrices in the underlying graphs.

Additionally, we note that while the Aggregation Homophily h_{agg} is a good indicator of the performance of GCN on `necessity-cora` and `syn-cora`, it does not correlate well with the performance changes of GCN on `syn-cora-loop` variants. While h_{agg} is defined as the ratio of nodes with NLD more similar to nodes from the same class than nodes from other classes (Eq. equation 6), it does not measure the level of similarity between the NLDs of nodes from the same or different classes as CNS does. Therefore, we believe that CNS is a more accurate and comprehensive indicator of the complexity of heterophily, as Table 1 shows.

Effectiveness of heterophilous GNN designs. With H_2GCN as an example that incorporates three heterophilous designs (D1), (D2) and (D3) (discussed in §3.1), we observe that these heterophilous designs can largely improve the performance of GNNs compared to GCNs even when the heterophilous connections do not have the ideal distinguishability in the NLDs as in the `necessity-cora` ($\gamma = 0$) case. When the distinguishability of NLDs among different classes is low (i.e., when intra-CNS is low and inter-CNS is high), the H_2GCN variants largely outperform GCN under heterophilous settings. While our experiments focus more on the effects of graph properties to NLD distinguishability and GNN performance and only considered H_2GCN as an example for heterophilous GNNs, more comprehensive experiments have been conducted in recent works [30, 46, 21] which support the effectiveness of these heterophilous GNN designs.

5 Conclusion & Future Directions

In this work, we revisited the debate of whether heterophily is a challenge for GNNs. We first reviewed representative architectural designs that have been proposed in the literature for improving the performance of GNNs on heterophilous data, and then discussed the connections with other objectives of GNN research, such as robustness, fairness, and reducing oversmoothing. To address the debate and reconcile seemingly contradictory statements in the literature, we conducted an extensive empirical analysis that aimed to provide a better understanding of when heterophily is challenging and when it does not pose significant additional challenges

compared to handling graphs with homophily. We also considered recently proposed measures for quantifying the complexity of heterophily and evaluated their effectiveness across synthetic datasets based on different generation processes. Our analysis revealed two key factors that increase the complexity of heterophily: (F1) the presence of low-degree nodes, and (F2) the complexity of the class compatibility matrices of the underlying graphs. These factors present unique challenges for GNNs under heterophilous settings, and necessitate architectural designs that can improve the performance of GNNs. We hope that our review and empirical analysis will inspire future research on better understanding the unique challenges of heterophily in GNNs and developing more effective GNN models that can handle well both graphs with homophily and heterophily (of variable complexity).

Future Directions. There are many promising research directions towards understanding the unique challenges that heterophily poses to GNN models. Next we discuss some representative open problems:

- **Beyond node classification and global homophily.** Most existing works on GNNs and heterophily (including the ones we review in this work) focus on node classification, where heterophily can be defined and measured with respect to the agreement of class labels for connected nodes. However, many important applications on graphs, such as recommendation systems, query matching, and the prediction of molecular properties, are based on other learning tasks such as link prediction and graph classification. It is thus important to understand the effects of heterophily on these tasks and inform the design of tailored GNN models that can handle heterophily. While few works have discussed heterophily in the settings of link prediction [53, 56] and graph classification [48], their definition of heterophily is still based on node class labels, which are often not available for these tasks. Measuring homophily in the absence of node is an interesting problem for these graph learning tasks. Moreover, going beyond a global perspective and exploring the effect of different mixing patterns across different neighborhoods is an important research direction that has started to gain reaction [23].

- **More datasets & applications.** Despite recent efforts in collecting and introducing new datasets that address the drawbacks of existing heterophilous ones [21, 30], we believe that the call for more heterophilous graph datasets and applications is still important and timely. Many existing works on GNN and heterophily rely on the six heterophilous graph datasets which were first adopted by **(author?)** [29]. While these datasets were useful during the early stages of research on GNNs and heterophily, multiple works [58, 21, 30] have pointed out the drawbacks of these commonly adopted benchmark datasets, namely their small sizes, artificial class labels, imbalanced class sizes, unusual network structure, and even leakage of test nodes in the training set. In light of these, **(author?)** [21] and **(author?)** [30] proposed a set of mid- to large-scale social, citation and web networks with more diverse node features and realistic class labels, but these datasets have yet to gain widespread adoption, and the relationship between the (heterophilous) links and the class labels is often ambiguous (e.g., predicting product ratings on Amazon based on edges connecting frequently bought items). Thus, we believe that there is still a need for datasets that have naturally-occurring heterophilous connections that align better with defined node class labels. In terms of application domains, it would be useful to go beyond social, citation, and webpage networks and introduce benchmarks that capture molecular or protein structures, which could also aid the investigation of more graph learning tasks that we discuss above.

- **Connections between heterophily & heterogeneity.** Although we highlighted in §2 that heterophily and heterogeneity are two distinct concepts that should not be confused, heterogeneity may introduce unique forms and challenges of heterophily that are worth investigating: connected nodes of different types could imply dissimilarity in their embeddings, resembling the concept of heterophily, while the level of homophily may also vary across different local mixing patterns. As a result, GNN models operating on heterogeneous graphs have already adopted designs similar to those tailored for heterophily, such as the separation of ego- and neighbor-embeddings and the use of type-specific kernels in message passing [31], in order to address the challenges of heterogeneity. Moreover, recently, **(author?)** [11] also discussed how enhancing the homophily level in the meta-paths of heterogeneous graphs can improve GNN performance. Therefore, we believe that

further research on the connections between heterophily and heterogeneity can help better understand the connections between the methodologies and findings of these two settings, which in turn may lead to the development of more effective GNNs for both scenarios.

Acknowledgements

We thank Yao Ma, Xiaorui Liu, Neil Shah and Jiliang Tang for sharing the synthetic datasets generated in [26] and the constructive discussions regarding the takeaways of our analysis in §4. This material is based upon work supported by the National Science Foundation under IIS 2212143, CAREER Grant No. IIS 1845491, IIS 1452425, an Adobe Digital Experience research faculty award, an Amazon faculty award, a Google faculty award, and AWS Cloud Credits for Research. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Quadro P6000 GPU used for this research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or other funding parties.

References

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing. In international conference on machine learning, pages 21–29. PMLR, 2019.

[2] Kristen M Altenburger and Johan Ugander. Monophily in social networks introduces similarity among friends-of-friends. Nature human behaviour, 2(4):284–290, 2018.

[3] Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. Beyond low-frequency information in graph convolutional networks. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, pages 3950–3957, 2021.

[4] Cristian Bodnar, Francesco Di Giovanni, Benjamin Chamberlain, Pietro Lio, and Michael Bronstein. Neural sheaf diffusion: A topological perspective on heterophily and oversmoothing in gnns. Advances in Neural Information Processing Systems, 35:18527–18541, 2022.

[5] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In ICML, 2020.

[6] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In International Conference on Learning Representations, 2021.

[7] Alex Chin, Yatong Chen, Kristen M. Altenburger, and Johan Ugander. Decoupled smoothing on graphs. In Proceedings of the 2019 World Wide Web Conference, pages 263–272, 2019.

[8] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In ICML, 2018.

[9] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In Advances in Neural Information Processing Systems (NeurIPS), pages 3844–3852, 2016.

[10] Lun Du, Yujie Li, Yifan Zhang, Xianglong Liu, and Meng Wang. Gbk-gnn: Gated bi-kernel graph neural networks for modeling both homophily and heterophily. In Proceedings of The Web Conference 2022, 2022.

[11] Jiayan Guo, Lun Du, Wendong Bi, Qiang Fu, Xiaojun Ma, Xu Chen, Shi Han, Dongmei Zhang, and Yan Zhang. Homophily-oriented heterogeneous graph rewiring. In Proceedings of the ACM Web Conference 2023, pages 511–522, 2023.

[12] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In Advances in neural information processing systems (NeurIPS), pages 1024–1034, 2017.

[13] Di Jin, Zhizhi Yu, Cuiying Huo, Rui Wang, Xiao Wang, Dongxiao He, and Jiawei Han. Universal graph convolutional networks. Advances in Neural Information Processing Systems, 34:10654–10664, 2021.

[14] Wei Jin, Tyler Derr, Yiqi Wang, Yao Ma, Zitao Liu, and Jiliang Tang. Node similarity preserving graph convolutional networks. In Proceedings of the 14th ACM international conference on web search and data mining, pages 148–156, 2021.

[15] Wei Jin, Yaxing Li, Han Xu, Yiqi Wang, Shuiwang Ji, Charu Aggarwal, and Jiliang Tang. Adversarial attacks and defenses on graphs: A review, a tool and empirical studies. SIGKDD Explor. Newsl., 22(2):19–34, Jan 2021.

[16] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In International Conference on Learning Representations (ICLR), 2017.

[17] Jia Li, Honglei Zhang, Zhichao Han, Yu Rong, Hong Cheng, and Junzhou Huang. Adversarial attack on community detection by hiding individuals. In WebConf, 2020.

[18] Peizhao Li, Yifei Wang, Han Zhao, Pengyu Hong, and Hongfu Liu. On dyadic fairness: Exploring and mitigating bias in graph connections. In International Conference on Learning Representations, 2021.

[19] Qimai Li, Zhichao Han, and Xiaoming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. AAAI, 2018.

[20] Yujie Li, Lun Du, Yifan Zhang, Xianglong Liu, and Meng Wang. Jacobi convolutional neural networks. In International Conference on Machine Learning, pages 6219–6228. PMLR, 2021.

[21] Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser-Nam Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. In Advances in Neural Information Processing Systems, 2021.

[22] Meng Liu, Zhengyang Wang, and Shuiwang Ji. Non-local graph neural networks. IEEE transactions on pattern analysis and machine intelligence, 44(12):10270–10276, 2021.

[23] Donald Loveland, Jiong Zhu, Mark Heimann, Ben Fish, Michael T Schaub, and Danai Koutra. On graph neural network fairness in the presence of heterophilous neighborhoods. In the 8th International Workshop on Deep Learning on Graphs (DLG-KDD'22), 2022.

[24] Sitao Luan, Yuchen Zhang, Ziqi Wang, Yujie Li, Yifan Zhang, Xinyu Zhang, Zhiyuan Liu, and Maosong Sun. Revisiting heterophily for graph neural networks. In Advances in Neural Information Processing Systems, 2022.

[25] Jiaqi Ma, Shuangrui Ding, and Qiaozhu Mei. Towards more practical adversarial attacks on graph neural networks. In NeurIPS, 2020.

[26] Yao Ma, Xiaorui Liu, Neil Shah, and Jiliang Tang. Is homophily a necessity for graph neural networks? In International Conference on Learning Representations, 2022.

[27] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. Annual Review of Sociology, 27(1):415–444, 2001.

[28] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. NetProbe: A Fast and Scalable System for Fraud Detection in Online Auction Networks. In Proceedings of the 16th international conference on World Wide Web, pages 201–210. ACM, 2007.

[29] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In International Conference on Learning Representations (ICLR), 2020.

[30] Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. A critical look at evaluation of gnns under heterophily: Are we really making progress? In International Conference on Learning Representations, 2023.

[31] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15, pages 593–607. Springer, 2018.

[32] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjing Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification. In Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI-21), pages 1548–1554. International Joint Conferences on Artificial Intelligence Organization, 2021.

[33] Yunchong Song, Chenghu Zhou, Xinbing Wang, and Zhouhan Lin. Ordered GNN: Ordering message passing to deal with heterophily and over-smoothing. In The Eleventh International Conference on Learning Representations, 2023.

[34] Indro Spinelli, Simone Scardapane, Amir Hussain, and Aurelio Uncini. Fairdrop: Biased edge dropout for enhancing fairness in graph representation learning. IEEE Transactions on Artificial Intelligence, 3(3):344–354, 2021.

[35] Yizhou Sun and Jiawei Han. Mining Heterogeneous Information Networks: Principles and Methodologies. Synthesis Lectures on Data Mining and Knowledge Discovery. Morgan & Claypool Publishers, 2012.

[36] Susheel Suresh, Vinith Budde, Jennifer Neville, Pan Li, and Jianzhu Ma. Breaking the limit of graph neural networks by improving the assortativity of graphs with local mixing patterns. In Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pages 1541–1551, 2021.

[37] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. International Conference on Learning Representations (ICLR), 2018.

[38] Tao Wang, Di Jin, Rui Wang, Dongxiao He, and Yuxiao Huang. Powerful graph convolutional networks with adaptive propagation mechanism for homophily and heterophily. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 36, pages 4210–4218, 2022.

[39] Yu Wang and Tyler Derr. Tree decomposed graph neural network. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pages 2040–2049, 2021.

[40] Yu Wang, Yuying Zhao, Yushun Dong, Huiyuan Chen, Jundong Li, and Tyler Derr. Improving fairness in graph neural networks via mitigating sensitive attribute leakage. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pages 1938–1948, 2022.

[41] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In International conference on machine learning, pages 6861–6871. PMLR, 2019.

[42] Huijun Wu, Chen Wang, Yuriy Tyshetskiy, Andrew Docherty, Kai Lu, and Liming Zhu. Adversarial examples for graph data: Deep insights into attack and defense. In IJCAI, 2019.

[43] 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, 32(1):4–24, 2020.

[44] Kaidi Xu, Hongge Chen, Sijia Liu, Pin-Yu Chen, Tsui-Wei Weng, Mingyi Hong, and Xue Lin. Topology attack and defense for graph neural networks: an optimization perspective. In IJCAI, 2019.

[45] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In Proceedings of the 35th International Conference on Machine Learning, ICML, volume 80, pages 5449–5458. PMLR, 2018.

[46] Yujun Yan, Milad Hashemi, Kevin Swersky, Yaoqing Yang, and Danai Koutra. Two sides of the same coin: Heterophily and oversmoothing in graph convolutional neural networks. In 2022 IEEE International Conference on Data Mining (ICDM), pages 1287–1292. IEEE, 2022.

[47] Zhiqing Yang, Yiqi Wang, Carl Yang, and Yuandong Tian. Graph pointer neural networks. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 36, pages 8832–8839, 2022.

[48] Wei Ye, Jiayi Yang, Sourav Medya, and Ambuj Singh. Incorporating heterophily into graph neural networks for graph classification. arXiv preprint arXiv:2203.07678, 2022.

[49] Yifan Zhang, Yuxin Chen, Nan Du, Xianglong Liu, and Meng Wang. Beyond low-frequency information in graph convolutional networks. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, pages 10842–10850, 2021.

[50] Yifan Zhang, Yuxin Chen, Nan Du, Xianglong Liu, and Meng Wang. Improving graph neural networks with simple architecture design. In arXiv preprint arXiv:2105.07634, 2021.

[51] Z. Zhang, P. Cui, and W. Zhu. Deep learning on graphs: A survey. IEEE Transactions on Knowledge and Data Engineering (TKDE), 2020.

[52] Xin Zheng, Yixin Liu, Shirui Pan, Miao Zhang, Di Jin, and Philip S Yu. Graph neural networks for graphs with heterophily: A survey. arXiv preprint arXiv:2202.07082, 2022.

[53] Shijie Zhou, Zhimeng Guo, Charu Aggarwal, Xiang Zhang, and Suhang Wang. Link prediction on heterophilic graphs via disentangled representation learning. arXiv preprint arXiv:2208.01820, 2022.

[54] Zhengyang Zhou, qihe huang, Gengyu Lin, Kuo Yang, LEI BAI, and Yang Wang. GReto: Remedying dynamic graph topology-task discordance via target homophily. In The Eleventh International Conference on Learning Representations, 2023.

[55] Jiong Zhu, Junchen Jin, Donald Loveland, Michael T Schaub, and Danai Koutra. How does heterophily impact the robustness of graph neural networks? theoretical connections and practical implications. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), 2022.

[56] Jiong Zhu, Aishwarya Reganti, Edward Huang, Charles Dickens, Nikhil Rao, Karthik Subbian, and Danai Koutra. Simplifying distributed neural network training on massive graphs: Randomized partitions improve model aggregation. arXiv preprint arXiv:2305.09887, 2023.

[57] Jiong Zhu, Ryan A Rossi, Anup Rao, Tung Mai, Nedim Lipka, Nesreen K Ahmed, and Danai Koutra. Graph neural networks with heterophily. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 35, pages 11168–11176, 2021.

[58] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. Advances in Neural Information Processing Systems, 33, 2020.

[59] Zhihao Zhu, Chenwang Wu, Min Zhou, Hao Liao, Defu Lian, and Enhong Chen. Resisting graph adversarial attack via cooperative homophilous augmentation. In Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2022, Grenoble, France, September 19–23, 2022, Proceedings, Part III, pages 251–268. Springer, 2023.

[60] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In KDD, 2018.

A Survey on Explainability of Graph Neural Networks

Jaykumar Kakkad[†¶] Jaspal Jannu[†*] Kartik Sharma[‡] Charu Aggarwal[+] Sourav Medya[†]
[†] University of Illinois Chicago, USA {jkakka4,jjannu2,medya}@uic.edu
[‡] Georgia Institute of Technology, Atlanta, USA ksartik@gatech.edu
[+] IBM T. J. Watson Research Center, New York, USA charu@us.ibm.com

Abstract

Graph neural networks (GNNs) are powerful graph-based deep-learning models that have gained significant attention and demonstrated remarkable performance in various domains, including natural language processing, drug discovery, and recommendation systems. However, combining feature information and combinatorial graph structures has led to complex non-linear GNN models. Consequently, this has increased the challenges of understanding the workings of GNNs and the underlying reasons behind their predictions. To address this, numerous explainability methods have been proposed to shed light on the inner mechanism of the GNNs. Explainable GNNs improve their security and enhance trust in their recommendations. This survey aims to provide a comprehensive overview of the existing explainability techniques for GNNs. We create a novel taxonomy and hierarchy to categorize these methods based on their objective and methodology. We also discuss the strengths, limitations, and application scenarios of each category. Furthermore, we highlight the key evaluation metrics and datasets commonly used to assess the explainability of GNNs. This survey aims to assist researchers and practitioners in understanding the existing landscape of explainability methods, identifying gaps, and fostering further advancements in interpretable graph-based machine learning.

1 Introduction

Recent years have seen a tremendous rise in the use of Graph Neural Networks (GNNs) for real-world applications, ranging from healthcare [128, 129], drug design [110, 52, 90], recommender systems [9], and fraud detection [78]. Predictions made in these domains have a substantial impact and therefore require to be highly trustworthy. In the realm of deep learning, one effective approach to enhance trust in these predictions is to provide an explanation supporting them [86]. These explanations elucidate the model's predictions for human understanding and can be generated through various methods. For instance, they may involve identifying important substructures within the input data [56, 81, 121], providing additional examples from the training data [12], or constructing counterfactual examples by perturbing the input to produce a different prediction outcome [55, 92, 9].

The interpretability of deep learning models is influenced by the characteristics of the input domain as the content and the complexity of explanations can vary depending on the inputs. When it comes to explaining predictions made by graph neural networks (GNNs), several challenges arise. First, since graphs are combinatorial data structures, finding important substructures by evaluating different combinations that maximize a certain prediction becomes difficult. Second, attributed graphs contain both node attributes and edge connectivity, which can influence the predictions and they should be considered together in explanations. Third, explanations must be adaptable to different existing GNN architectures. Lastly, explanations for the local tasks (e.g., node or edge level) may differ from those for global tasks (e.g., graph level). Due to these challenges, explaining graph neural networks is non-trivial and a large variety of methods have been proposed in the literature to tackle

[¶]Both authors contributed equally.

it [114, 68, 106, 76, 5, 96, 56, 118, 92, 4, 72]. With the increasing use of GNNs in critical applications such as healthcare and recommender systems, and the consequent rise in their explainability methods, we provide an updated survey of the explainability of GNNs. Additionally, we propose a novel taxonomy that categorizes the explainability methods for GNNs, providing a comprehensive overview of the field.

Existing surveys on GNN explainability predominantly focus on either factual methods [119, 13, 47] or counterfactual methods [26] but not both. These surveys thus lack a comprehensive overview of the different methods in the literature by limiting the discussion to specific methods [119, 47] or only discussing them under the broad umbrella of trustworthy GNNs [13, 103]. Our survey aims to bridge this gap by providing a comprehensive and detailed summary of existing explainability methods for GNNs. We include both factual as well as counterfactual methods of explainability in GNNs. To enhance clarity and organization, we introduce a novel taxonomy to categorize these methods for a more systematic understanding of their nuances and characteristics.

1.1 Graph Neural Networks

Graph neural networks (GNNs) have been used to learn powerful representations of graphs [42, 95, 28]. Consider a graph G given by $G = (V, E)$, where V denotes the set of n nodes and E denotes the set of m edges. We can create an adjacency matrix $\mathbf{A} \in [0, 1]^{n \times n}$ such that $A_{ij} = 1$ if $(i, j) \in E$ and 0 otherwise. Each node may have attributes, given by the matrix $\mathbf{X} \in \mathbb{R}^{n \times F}$ such that each row i stores the F-dimensional attribute vector for node i. A GNN model $\mathcal{M}$ embeds each node $v \in V$ into a low-dimensional space $\mathbf{Z} : \mathbb{R}^{n \times d}$ by following this message passing rule for k steps as

$$\mathbf{Z}_v^{(k+1)} = \textsc{Update}_{\Phi}(\mathbf{Z}_v^{(k)}, \textsc{Agg}(\{\textsc{Msg}_{\Theta}(\mathbf{Z}_v^{(k)}, \mathbf{Z}_u^{(k)}) : (u, v) \in E\})), \tag{7}$$

such that $\mathbf{Z}^0 = \mathbf{X}$ and $\mathbf{Z} := \mathbf{Z}^{(k)}$. Different instances of the update $\textsc{Update}_{\Phi}$, aggregation $\textsc{Agg}_{\Phi}$ and message generator $\textsc{Msg}_{\Theta}$ functions give rise to different GNN architectures. For example, GCN [42] has an identity message, a mean aggregation, and weighted update functions, while GAT [95] learns an attention-based message generation instead. These embeddings are trained for a specific task $\mathcal{T}$ that can be either supervised (e.g., node classification, graph classification, etc.) or unsupervised (e.g., self-supervised link prediction, clustering, etc.).

1.2 Explainability in ML

Problem 1 (Explainability [8]) *Consider a supervised task $\mathcal{T}$ with the aim of learning a mapping from $\mathcal{X}$ to $\mathcal{Y}$, and a model $\mathcal{M}$ trained for this task. Given a set of $(\mathbf{x}, y)$ pairs $\subseteq (\mathcal{X}, \mathcal{Y})$ and the model $\mathcal{M}$, generate an explanation $\mathbf{e}$ from a given set $\mathcal{D}_E$ such that $\mathbf{e}$ "explains" the prediction $\hat{y} = \mathcal{M}(\mathbf{x})$.*

These explanations can be either *local* to a single test input $(\mathbf{x}, y)$ or *global* when they explain prediction over a specific dataset $\mathcal{D}' \subseteq (\mathcal{X}, \mathcal{Y})$. Further, the explanation can be generated either *post-hoc* (i.e., after the model training) or *ante-hoc* where the model itself is *self-interpretable*, i.e., it explains its predictions. With some exceptions, post-hoc explanations usually consider a black-box access to the model while self-interpretable methods update the model architecture and/or training itself. We can further differentiate the explanation methods based on their content, i.e., the explanation set $\mathcal{D}_E$. *Local explanations* only consider the local neighborhood of the given data instance while *global explanations* are concerned about the model's overall behavior and thus, searches for patterns in the model's predictions. On the other hand, explanations can also be *counterfactual*, where the aim is to explain a prediction by providing a contrasting example that changes it.

2 Overview

With the widespread adoption of GNNs across various applications, the demand for explaining their predictions has grown substantially. Recently, the community has witnessed a surge in efforts dedicated to the explainability of GNNs. These methods exhibit variations in terms of explanation types, utilization of model information, and training procedures, among other factors. We organize and categorize these methods to develop a deeper understanding of the existing works and provide a broad picture of their applicability in different scenarios.

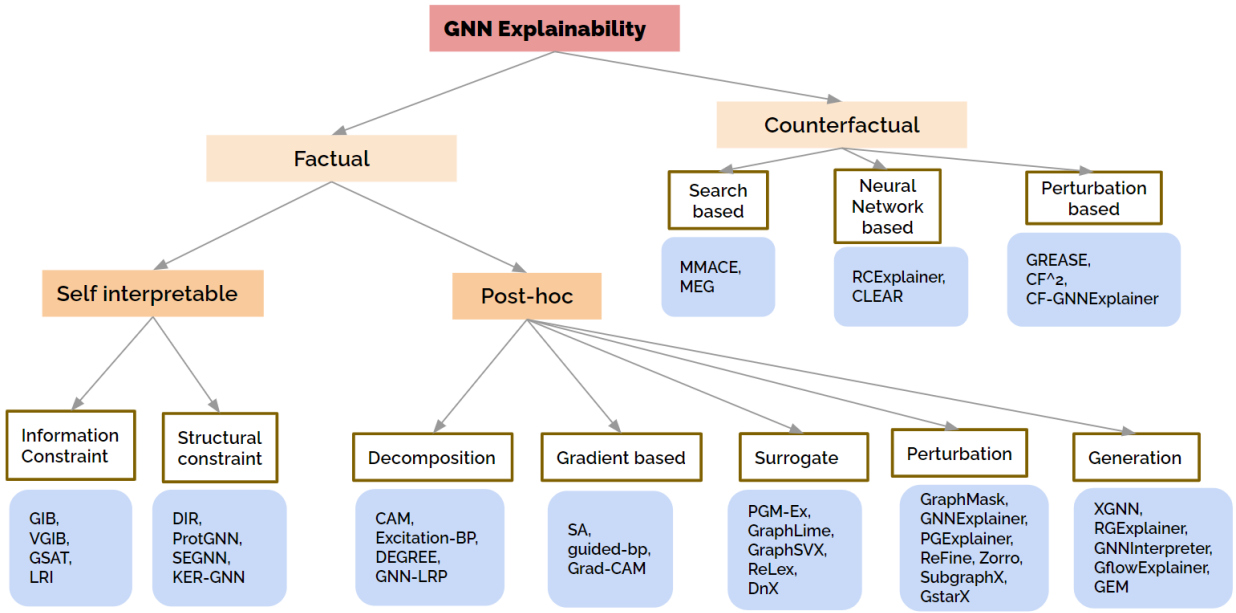
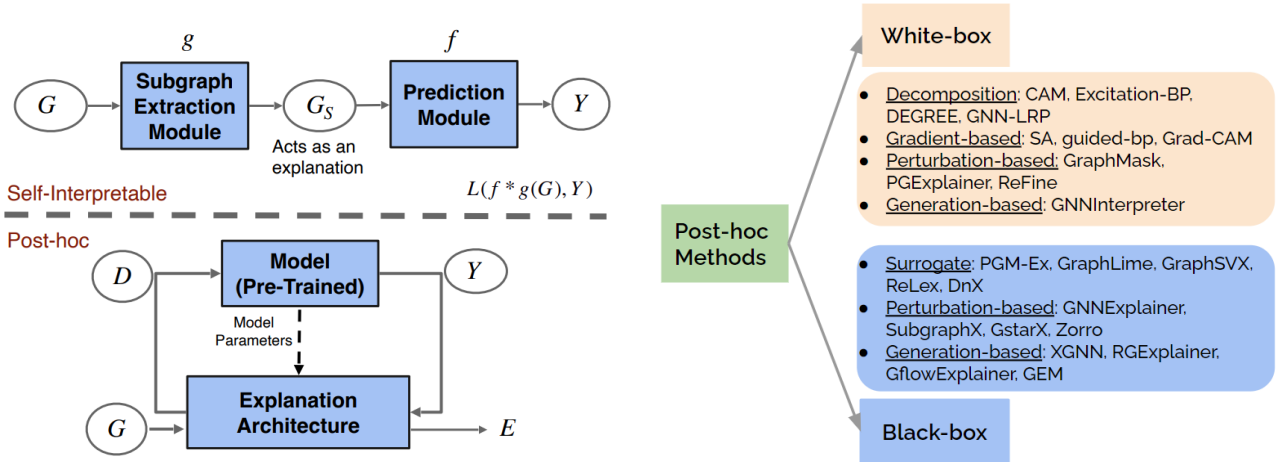


Figure 1: **Overview of the Schema. (1) Factual.** Information constraints: GIB [117], VGIB [115], GSAT [68], LRI [69]; Structural Constraints: DIR [106], ProtGNN [124], SEGNN [12], KER-GNN [21]; Decomposition: CAM [76], Excitation-BP [76], DEGREE [22], GNN-LRP [83]; Gradient-based: SA [5] , Guided-BP [5] , Grad-CAM [76]; Surrogate: PGM-Ex [56], GraphLime [34], GraphSVX [17], ReLex [123], DnX [74]; Perturbation-based: GNNExplainer[114], GraphMask [81], PGExplainer [56], ReFine [99], ZORRO [23], SubgraphX [121], GstarX [122]; Generation: XGNN [118], RGExplainer [84], GNNInterpreter [100], GFlowExplainer [46], GEM [49]; **(2) Counterfactual.** Search-based: MMACE [101] , MEG [72]; Neural Network-based: RCExplainer [4], CLEAR [57]; Perturbation-based: GREASE [9], CF2 [92], CF-GNNexplainer [55]

Main Schema: Factual and Counterfactual Methods. Figure 1 provides an overview of the broad categorization of the existing works. Based on the type of explanations, we first make two broad categories: (1) Factual and (2) Counterfactual. Factual methods aim to find an explanation in the form of input features with the maximum influence over the prediction. These explanations can be a set of either node features or a substructure (set of nodes/edges) or both. On the other hand, counterfactual methods provide an explanation by finding the smallest change in the input graph that changes the model's prediction. Hence, counterfactual explanations can be used to find a set of similar features that can alter the prediction of the model.

Organization. In the following sections, we describe each category in detail and provide summary of various explainability methods in each category. In Sec. 3, we describe the factual approaches which are further classified into self-interpretable and post-hoc categories. In Sec. 4, the counterfactual methods are categorized into perturbation-based, neural network-based and search-based methods. Sec. 5 presents three special categories of explainers such as temporal, global and causality-based. In Sec. 6, we overview the explainer methods that are relevant for specific applications in different domains such as in social networks, biology, and computer security. Lastly, we review widely used datasets in Sec. 7 and evaluation metrics in Sec. 8.



(a) Self-Interpretable & Post-hoc

(b) Categorization of Post-hoc methods

Figure 2: (a) **Self-interpretable and post-hoc architectures :** In self-interpretable methods, the subgraph extraction module g uses constraints to find an informative subgraph G_s from the input graph G. The prediction module f uses this G_s to predict the label Y. In contrast, Post-hoc methods consider model as pre-trained with fixed weights. For any instance G, post-hoc methods generate explanation using model's input $\mathcal{D}$, output Y and in some cases the model's internal parameters. (b) **White-box and Black-box post-hoc methods:** Methods are shown in the individual categories. Decomposition-based: CAM [76], Excitation-BP [76], DEGREE [22], GNN-LRP [83]; Gradient-based: SA [5] , Guided-BP [5] , Grad-CAM [76]; Surrogate: PGM-Ex [56], GraphLime [34], GraphSVX [17], ReLex [123], DnX [74]; Perturbation-based: GNNExplainer [114], GraphMask [81], PGExplainer [56], ReFine [99], ZORRO [23], SubgraphX [121], GstarX [122]; Generation-based: XGNN [118], RGExplainer [84], GNNInterpreter [100], GFlowExplainer [46], GEM [49].

3 Factual

We classify the factual explainer methods broadly into two categories based on the nature of the integration of the explainability architecture with the main model as follows.

- **Post-hoc:** Post-hoc methods do not have the explainable architecture inbuilt into the model to attribute a model's prediction to the input. As seen in Figure 2a, the explainability architecture (EA) is separated from the model which is pre-trained with fixed weights. For any instance G, post-hoc methods generate an explanation using the model's input $\mathcal{D}$, output Y and sometimes even internal parameters of the model. Note that different EAs use different inputs $\mathcal{D}$ that are fed to the model. Post-hoc methods might not be always accurate as they may end up extracting features that are spuriously correlated with the task [124, 80, 68, 45].
- **Self-interpretable:** Contrary to post-hoc methods, self-interpretable explainable methods design explainability architecture directly inside the model. As seen in Figure 2a, these methods usually have two modules. The subgraph extraction module (the function g) uses constraints to find an informative subgraph G_s from the input graph G. Then, the prediction module f uses G_s to predict label Y. G_s also acts as an explanation. Both modules are trained together with an objective $L(f \circ g(G), Y)$ to minimize the loss between prediction $f \circ g(G)$ and label Y. One major drawback of self-interpretable models is that the good interpretability of the models is often at the cost of the prediction accuracy [68].

3.1 Post-hoc

We divide the post-hoc methods based on their approaches used to find explanation into the following categories: a) Decomposition-based methods (Sec. 3.1.1), b) Gradient-based methods (Sec. 3.1.2), c) Surrogate methods

Table 1: Key highlights of *decomposition-based* methods

Method	Parameters	Form of Explanation	Task
CAM [76]	Node embedding of last layer and MLP weights	Node importance	Graph Classification
Excitation-BP [76]	Weights of all GNN layers	Node importance	Node and Graph Classification
DEGREE [22]	Decomposes messages and requires all parameters	Similar nodes	Node and Graph Classification
GNN-LRP [83]	Weights of all layers	Collection of edges	Node and Graph Classification

(3.1.3), d) Perturbation-based methods (Sec. 3.1.4), e) Generation-based methods (3.1.5). The post-hoc methods can also be categorized based on their requirement to access the internal parameters of the model. As seen in figure 2b, this division results into the following categories of the methods: **white-box** and **black-box**.

White-box: These methods require access to internal model parameters or embeddings to provide explanations. For instance, all decomposition-based methods (Sec. 3.1.1) require model parameters such as node weights of each layer to compute an importance score of different parts of the input. Even gradient based methods (Sec. 3.1.2) require access to the gradients. Thus, all methods in these categories are considered as white-box methods and they are not suitable in cases where a model's internal parameters are inaccessible.

Black-box: Contrary to the white-box methods, black-box methods do not require access to the model's internal parameters. For instance, all approaches in the category of surrogate methods (Sec. 3.1.3) generate a local dataset using the model's input and output. Since these methods do not require access to the model parameters, all of them can be categorized as black-box methods.

3.1.1 Decomposition-based Methods

These methods consider the prediction of the model as a score that is decomposed and distributed backwards in a layer by layer fashion till it reaches the input. The score of different parts of the input can be construed as its importance to the prediction. However, the decomposition technique can vary across methods. They also require internal parameters of the model to calculate the score. Hence, these explanation methods are considered as white-box methods. Table 1 provides a summary of these methods.

One of the decomposition-based methods, **CAM** [76] aims at constructing the explanation of GNNs that have a Global Average Pooling (GAP) layer and a fully connected layer as the final classifier. Let e_n be the final embedding of node n just before the GAP layer and w^c be the weight vector of the classifier for the class C. The importance score of the node n is computed as $(w^c)^T e_n$. This means that the node's contribution to the class score y^c is taken as the importance. It is clear that this method is restricted to GNNs that have a GAP layer and perform only the graph classification task.

Another method, **Excitation-BP** [76] considers that the final probability of the prediction can be decomposed into excitations from different neurons. The output of a neuron can be intuitively understood as the weighted sum of excitations from the connected neurons in the previous layer combined with a non-linear function where the weights are the usual neural network parameters. With this, the output probability can be distributed to the neurons in the previous layer according to the ratios of these weights. Finally, the importance of a node is obtained by combining the excitations of all the feature maps of that node.

Contrary to other methods, **DEGREE** [22] finds explanation in the form of subgraph structures. First, it decomposes the message passing feed-forward propagation mechanism of the GNN to find a contribution score of a group of target nodes. Next, it uses an agglomeration algorithm that greedily finds the most influential subgraph as the explanation. **GNN-LRP** [83] is based on the concept that the function modeled by GNN is a polynomial function in the vicinity of a specific input. The prediction score is decomposed by approximating the higher order Taylor expansion using layer-wise relevance propagation [3]. This differs from other decomposition-based methods not only in the decomposition technique but also in the score attribution. While other methods attribute

Table 2: Key highlights of *gradient-based* methods

Method	Explanation Type	Task	Explanation Target	Datasets Evaluated
SA [5]	Instance level	Graph classification Node classification	Nodes, Node features Edges, Edge features	Infection, ESOL [15]
Guided-BP [5]	Instance level	Graph classification Node classification	Nodes, Node features Edges, Edge features	Infection, ESOL [15]
Grad-CAM [76]	Instance level	Node classification	Nodes, Node features	BBBP, BACE, TOX21 [40]

scores to nodes or edges, GNN-LRP attributes scores to walks i.e., a collection of edges.

3.1.2 Gradient-based Methods

The gradient-based explainer methods follow the following key idea. Gradients represent the rate of change, and the gradient of the prediction with respect to the input represents how sensitive the prediction is to the input. This sensitivity is seen as a measure of importance. We provide a summary of these methods in Table 2.

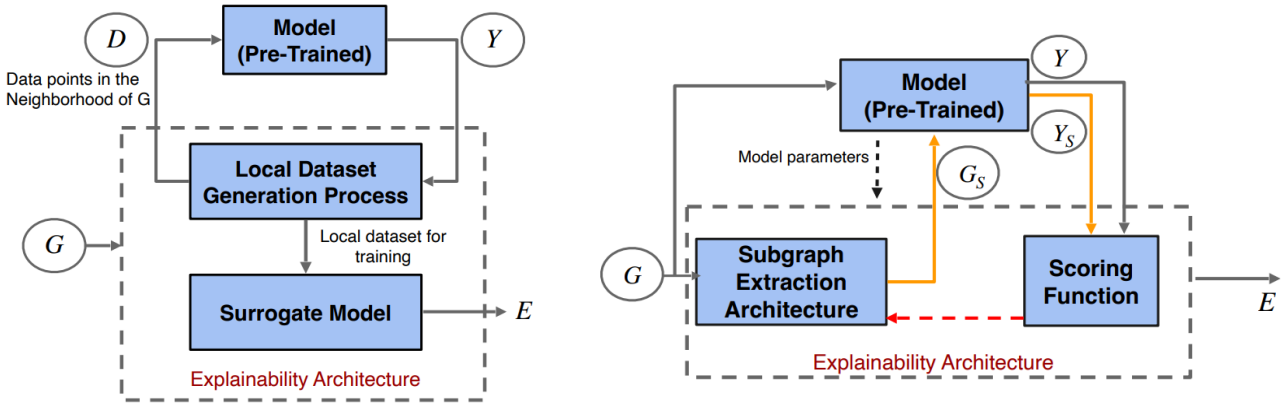
Sensitivity Analysis (SA) [5] is one of the earlier methods to use gradients to explain GNNs. Let x be an input, which can be a node or an edge feature vector, $SA(x)$ be its importance, and ϕ be the GNN model, then the importance is computed as $SA(x) \propto ||\nabla_x \phi(x)||^2$. The intuition behind this method is based on the aforementioned sensitivity to the input. **Guided Backpropagation (Guided-BP)** [5], a slightly modified version of the previous method SA, follows a similar idea except the fact that the negative gradients are clipped to zero during the backpropagation. This is done to preserve only inputs that have an excitatory effect on the output. Intuitively, since positive and negative gradients have opposing effect on the output, using both of them could result in less accurate explanations.

The method **Grad-CAM** [76] builds upon **CAM** [76] (see Section 3.1.1), and uses gradients with respect to the final node embeddings to compute the importance scores. The importance score is $(\frac{1}{N}\sum_{n=1}^{N} \nabla_{e_n}(y^c))^T e_n$, where e_n is the final embedding of node n just before the GAP layer, and w^c is the weight vector of the classifier for class C, $g = \frac{1}{N}\sum_{n=1}^{N} e_n$ is the vector after the GAP layer, and the final class score y^c of C is $w^T g$. This removes the restriction about the necessity of GAP layer. This equation shows that the importance of each node is computed as the weighted sum of the feature maps of the node embeddings, where the weights are the gradients of the output with respect to the feature maps.

All the above methods depend on this particular intuition that the gradients can be good indicators of importance. However, this might not be useful in many settings. Gradients indicates sensitivity which does not reflect importance accurately. Moreover, saturation regions where the prediction of the model does not change significantly with the input, can be seen as another issue in SA and Guided-BP.

3.1.3 Surrogate Methods

Within a large range of input values, the relationship between input and output can be complex. Hence, we need complex functions to model this relationship and the corresponding model might not be interpretable. However, in a smaller range of input values, the relationship between input and output can be approximated by simpler and interpretable functions. This intuition leads to *surrogate methods* that fit a simple and interpretable surrogate model in the locality of the prediction. Table 3 shows different locality-based data extraction techniques and surrogate models used by surrogate methods. This surrogate model can then be used to generate explanations. As seen in the figure 3a, these methods adopt a two-step approach. Given an instance G, they first generate data from the prediction's neighborhood by utilizing multiple inputs D within the vicinity and recording the model's prediction Y. Subsequently, a surrogate model is employed to train on this data. The explanation E provided by the surrogate model serves as an explanation for the original prediction.



(a) **Generic schema of surrogate methods** (b) **Generic schema of perturbation-based methods**

Figure 3: **a) Surrogate:** These methods follow a two-step process. For any instance G, they generate data from the neighbourhood of the prediction by using multiple inputs D in the locality and recording its model prediction Y. Then a surrogate model is used to fit this data. Explanation E for the surrogate model is the explanation for the prediction, **b) Perturbation-based:** They have two key modules: a subgraph extraction architecture and a scoring function. For an input G, the subgraph extraction module extracts a subgraph G_s. The model prediction Y_s for subgraph G_s are scored against the actual predictions Y using a scoring function. The feedback from the scoring function can be used to train the subgraph extraction module. Sometimes model parameters are also used as the training input to the subgraph extraction module. The optimal subgraph G_s^* acts as the final explanation E.

PGMExplainer constructs a Bayesian network to explain the prediction. First, it creates a tabular dataset by random perturbations on node features of multiple nodes of the computational graph and records its influence on the prediction. A grow-shrink algorithm is used to select top influential nodes. Using structure learning, a Bayesian network is learnt that optimizes the Bayesian Information Criterion (BIC) scores and the DAG of conditional probabilities act as an explanation. In **GraphLime** [34], the local explanations are based on Hilbert-Schmidt Independence Criterion Lasso (HSIC Lasso) model, which is a kernel-based nonlinear interpretable feature selection algorithm. This method assumes that the node features in the original graph are easily interpretable. The HSIC model takes a node and its N-hop neighbourhood (for some N), and selects a subset of node features that are the most influential to the prediction. These selected features act as the explanation. To construct a local dataset, **GraphSVX** [17] uses a mask generator to jointly perturb the nodes and the features and observes its effects on the predictions. The mask generator isolates the masked nodes and replaces masked features by its expected values. It then fits a weighted linear regression model (WLR) on the local dataset. The coefficients of WLR act as explanations.

The next two approaches use GNN based models to act as surrogate models. **RelEx** [123] uses a BFS-based sampling strategy to select nodes and then perturb them to create the local dataset. Then, a GCN model with residual connections is used to fit this dataset. In contrast to other methods in this category, the surrogate model of RelEx is not interpretable. Hence, it uses perturbation-based strategy to find a mask that acts as explanation. We note that the surrogate model is more complex compared to other methods and it requires the use of another explanation method to derive explanations from the surrogate model. **DistilnExplain (DnX)** [74] first learns a surrogate GNN via knowledge distillation and then provides an explanation by solving a simple convex program. In contrast to RelEx, DnX uses a simpler surrogate model which is a linear architecture termed as Simplified Graph convolution (SGC) [104]. SGC does not have any non-linear activation layers and uses a single parameter matrix across layers. The parameters in SGC are learned via knowledge distillation with an objective to minimize the KL-divergence between the predictions by SGC and the model. Furthermore, explanations can be derived from SGC by solving a simple convex program.

Table 3: Key highlights of *surrogate methods*

Method	Local Dataset extraction	Surrogate model	Explanation
GraphLime [34]	N-hop neighbor nodes	HSIC Lasso	Weights of the model
PGMExplainer [96]	Random node feature perturbation	Bayesian network	DAG of conditional dependence
ReLex [123]	Random sampling of connected subgraphs	GCN	Perturbation-based method
DistilnExplain [74]	Entire dataset	Knowledge distilled Simple GCN [104]	Convex programming, decomposition
GraphSVX [17]	Input perturbations via mask generator	Weighted Linear Regression (WLR)	Weights of WLR

3.1.4 Perturbation-based Methods

These methods find *important subgraphs* as explanations by perturbing the input. Fig. 3b presents two key modules of these methods: the subgraph extraction module and the scoring function module. For an input G, the subgraph extraction module extracts a subgraph G_s. The model predictions Y_s for subgraphs are scored against the actual predictions Y using a scoring function. The feedback from the scoring function can be used to train the subgraph extraction module. In some cases, model parameters are also used as the training input for the subgraph extraction module. These methods provide explanations E in the form of a subgraph structure and some also provide node features as explanations. Table 4 presents a summary of these methods.

GNNExplainer [114] is one of the initial efforts towards the explainability of GNNs. It identifies an explanation in the form of a Subgraph including a subset of node features that have the maximum influence on the prediction. It learns continuous masks for both adjacency matrix and features by optimizing cross entropy between the class label and model prediction on the masked subgraph. In a follow-up work, **PGExplainer** [56] extends the idea in GNNExplainer by assuming the graph to be a random Gilbert graph, where the probability distribution of edges is conditionally independent. The distribution of each edge is independently modeled as a Bernoulli distribution, i.e., each edge has a different parametric distribution. These parameters are modeled by a neural network (MLP), and the parameters of this MLP is computed by optimizing the mutual information between the explanation subgraph and the predictions of the underlying GNN model. Another masking-related method, **GraphMask** [81] provides an explanation by learning a parameterized edge mask that predicts the edge to drop at every layer. A single-layer MLP classifier is trained to predict the edges that can be dropped. To keep the topology unaffected, these edges are not dropped but are replaced by a learned baseline vector. The training objective is to minimize the L_0 norm i.e., the total number of edges not masked, such that the prediction output remains within a tolerance level. To make the objective differentiable, it uses sparse relaxations through the reparameterization trick and the hard concrete distribution [59, 36].

Another approach **Zorro** [23] finds explanations in the form of important nodes and features that maximizes *Fidelity* (see Sec. 8). It uses a greedy approach that selects the node and the feature at each step with the highest fidelity score. Fidelity is computed as the expected validity of the perturbed input. The approach uses a discrete mask for selecting a subgraph without any backpropagation. A two-staged approach, **ReFine** [99] consists of the edge attribution or pre-training and the edge selection or fine-tuning steps. During pre-training, a GNN and an MLP are trained to find the edge probabilities for the entire class by maximizing mutual information and contrastive loss between classes. During the fine-tuning step, the edge probabilities from the previous stage are used to sample edges and find an explanation that maximizes mutual information for a specific instance.

The next two approaches use cooperative game theoretic techniques. **SubgraphX** [121] applies the Monte Carlo Tree search technique for subgraph exploration and uses the Shapley value [85] to measure the importance of the subgraphs. For the search algorithm, the child nodes are obtained by pruning the parent graph. In computing the Shapley values, the Monte Carlo sampling helps to find a coalition set and the prediction from the GNN is used as the pay-off in the game. In a subsequent work, **GStarX** [122] uses a different technique from cooperative game theory known as HN value [27], to compute importance scores of a node for both graph and node classification tasks. In contrast to the Shapley value, the HN value is a structure-aware metric. Since computing the HN values is expensive, Monte Carlo sampling is used for large graphs. The nodes with the top-k highest HN values act as

Table 4: Key highlights of the *perturbation-based* methods.

Method	Subgraph Extraction Strategy	Scoring function	Constraints	Node Feature Explanation
GNNExplainer [114]	Continuous relaxation	Mutual Information	Size	Yes
SubgraphX [121]	Monte Carlo Tree Search	Shapley Value	Size, connectivity	No
GraphMask [81]	Layer-wise parameterized edge selection	L_0 norm	Prediction divergence	No
PGexplainer [56]	Parameterized edge selection	Mutual Information	Size and/or connectivity	No
Zorro [23]	Greedy selection	Fidelity	Threshold fidelity	Yes
ReFine [99]	Parameterized edge attribution	Mutual Information	Number of edges	No
GstarX [122]	Monte Carlo sampling	HN-value	Size	No

Table 5: Key highlights of the *generation-based* methods

Methods	Explanation Type	Optimization	Constraints	Task
XGNN [118]	Model level	RL-policy gradient	Domain specific rules	Graph classification
RG-Explainer [84]	Instance level	RL-policy gradient	Size, radius, similarity	Node & Graph classification
GFLOW Explainer [46]	Instance level	TD flow matching	Connectivity / cut vertex	Node & Graph classification
GNNinterpreter [100]	Model level	Continuous relaxation	Similarity to mean	Graph Classification
GEM [49]	Instance level	Autoencoder	Graph validity rules	Node & Graph Classification

an explanation.

3.1.5 Generation-based methods

Generation-based approaches either use generative models or graph generators to derive instance-level or model-level explanations. Furthermore, to ensure the validity of the generated graphs, different approaches have been proposed. Table 5 provides a summary of the generation-based methods.

XGNN [118] provides model-level explanations by generating key subgraph patterns to maximize prediction for a certain class. The subgraph is generated using a Reinforcement learning (RL) based graph generator which is optimized using policy gradient. In the setup for the RL agent, the previous graph is the state; adding an edge is an action; and the model prediction along with the validity rules acts as the reward. Unsurprisingly, the validity rules are specified based on domain knowledge. Another RL-based method, **RG-Explainer** [84] formulates the underlying problem as combinatorial optimization instead of using continuous relaxation or search methods to find the subgraph. A starting point is selected using an MLP which acts as an input to the graph generator. The graph generator is an RL agent that optimizes for the policy using policy gradient with subgraph as the state, adding neighboring nodes as the action, and the function of the cross entropy loss as the reward.

A non-RL method, **GNNinterpreter** [100] is a generative model-level explanation method for the graph classification task. Its objective is to maximize the likelihood of predicting the explanation graph correctly for a given class. The similarity between the explanation graph embedding and the mean embedding of all graphs act as an optimization constraint. Intuitively, this ensures that the explanation graph stays closer to the domain and is meaningful. Since the adjacency matrix and sometimes even the features can be categorical, GNNinterpreter uses the Grumbel softmax method [36] to enable backpropagation of gradients. Contrary to XGNN with domain-specific hand-crafted rules, GNNinterpreter uses numerical optimization and does not need any domain knowledge.

GFLOW Explainer [46] uses GFLOWNETs as the generative component. The objective is to construct a TD-like flow matching condition [6] to learn a policy to generate a subgraph by sequentially adding neighbors (nodes) such that the probability of the subgraph of a class is proportional to the mutual information between the label and the distribution of possible subgraphs. A *state* consists of several nodes with the initial state as the single most influential node and the end state that satisfies the stopping criteria. *Action* is adding a node and the *reward* is a function of the cross-entropy loss.

GEM [49] uses the principles of Granger causality to generate ground-truth explanations which are used to

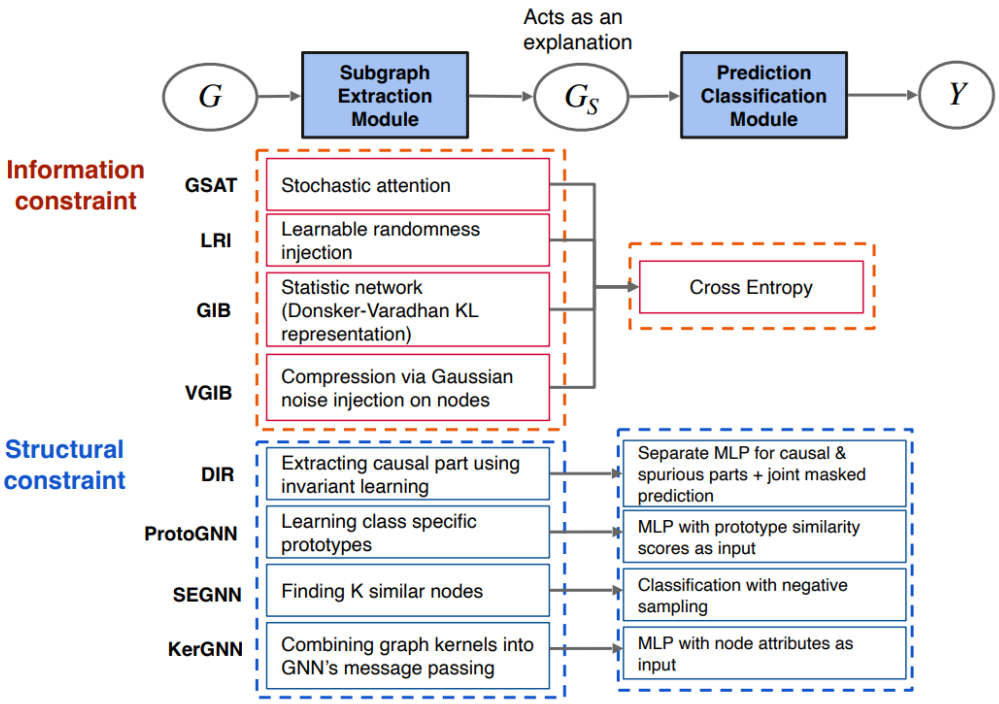


Figure 4: **Self-interpretable methods:** Every self-interpretable method has a *subgraph extraction* and a *prediction* module. The subgraph extraction module (the function g) uses constraints to find an informative subgraph G_s from input graph G. The prediction module uses G_s to predict label Y. This also shows the techniques used by each method to implement these individual modules. Self-interpretable Methods are categorized based on constraints: **(1) Information constraint:** GIB [117], VGIB [115], GSAT [68], LRI [69]; **(2) Structural constraint:** DIR [106], ProtGNN [124], SEGNN [12], KER-GNN [21].

train the explainer. It quantifies the causal contribution of each edge in the computational graph by the difference in the loss of the model with and without the edge. This distilled ground-truth for the computation graph is used to train the generative auto-encoder based explainer. This explainer provides an explanation for any instance in the form of the subgraph of the computation graph.

3.2 Self-interpretable

In self-interpretable methods, the explainable procedure is intrinsic to the model. Such methods derive explainability by incorporating interpretability constraints. These methods use either information constraints or cardinality (structural) constraints to derive an informative subgraph which is used for both the prediction and the explanation. Based on the design of the explainability, we further classify the self-interpretable methods into two types based on the imposed constraints (Fig. 4).

3.2.1 Methods with information constraints

One of the major challenges in constructing explanations via subgraphs is that the critical subgraphs may have different sizes and can be irregular. Thus, constraining the size of the explanation may not be appropriate for the underlying prediction task. To address this challenge, the methods based on information constraint use the principle of information bottleneck (IB) [93] to impose constraints on the information instead of the size. For a graph G, subgraph G_s and label Y, the graph information bottleneck (GIB) objective is:

$$\max_{G_s} I(Y, G_s) \;\text{ such that }\; I(G, G_s) \leq \gamma$$

Table 6: Key highlights of the methods with *information constraints*

Method	Process of calculating $I(G, G_s)$	Injection of Randomness	Subgraph Extractor Architecture
GSAT [68]	Stochastic attention	Bernoulli as Prior for KL divergence	GNN + MLP + Reparameterization
LRI [69]	Learnable Randomness injection	Bernoulli and Gaussian as prior	GNN + MLP + Reparameterization
GIB [117]	Donsker-Vardhan KL representation [16]	No randomness injection	Statistic Network: GNN + MLP
VGIB [115]	Compression via Noise injection	Gaussian noise on node features	GNN + MLP + Reparameterization

where I denotes the mutual information. Using Lagrangian multiplier β, we can write the equation as:

$$\min_{G_s} -I(Y, G_s) \; + \; \beta * I(G, G_s)$$

As seen from the above equations, GIB objective-based methods have two parts in the objective function and both are intractable. All methods approximate $I(Y, G_s)$ by calculating the cross-entropy loss. However, all methods vary in their approach in making $I(G, G_s)$ tractable i.e., all have different approaches to compressing the graph and finding the informative subgraph G_s. This subgraph is used for both prediction and interpretation. Table 6 provides the key highlights of all methods in this category.

GSAT [68] uses a stochastic attention mechanism to calculate the variational upper bound for $I(G, G_s)$. First, it encodes graph G using a GNN to find the representation for each node. Then, for each node pair (u, v), GSAT uses an MLP to calculate P_{uv}. This is used to sample stochastic attention from Bernoulli distribution $Bern(P_{uv})$ to extract a subgraph G_s. The variational upper bound is the KL divergence between $Bern(P_{uv})$ and $Bern(\alpha)$ where α is a hyper-parameter. Building on similar concepts, **LRI** [69] uses both Bernoulli and Gaussian distribution as the prior distribution. LRI-Bernoulli provides the existence importance of points and LRI-Gaussian provides the location importance of the points i.e., how perturbing the location of the point in different directions affects the prediction. Another method, **GIB** [117] assumes that there is no reasonable prior distribution to solve $I(G, G_s)$ via KL divergence in the graph space. Hence, it uses the Donsker-Vardhan KL representation [16] in the latent space. It employs a bi-level optimization wherein the statistic network of the Donsker-Varadhan representation is used to estimate $I(G, G_s)$ in the inner loop. This estimate with classification and connectivity loss is used to optimize the GIB objective in the outer loop. This bi-level training process is inefficient and unstable; and hence **VGIB** [115] uses a different compression technique. The information in the original graph is dampened by injecting noise into the node representations via a learned probability P_i for each node i. The classification loss will be higher if the informative substructure G_s^* is injected with noise. Hence, G_s^* is less likely to be injected with noise compared to label-irrelevant substructures.

3.2.2 Methods with structural constraints

Imposing structural constraints on the input to derive the most informative subgraph has also been a common approach. The obtained informative subgraph is used for both making predictions and generating explanations. The key difference across the methods is the set up of the structural constraints. In Table 7, we provide the key highlights of these methods.

One of the earlier methods, **DIR** [106] finds explanations in the form of invariant causal rationales by learning to split the input into causal (C) and non-causal (S) parts. The objective is to minimize the classification loss such that Y (the prediction) is independent of S given C. To achieve this, it first creates multiple interventional distributions by conducting interventions on the training distribution. The part that is invariant across these distributions is considered a causal part. Moreover, the implementation has three key stages. First, the architecture consists of a rationale generator (GNN) that splits the input graph into a causal part with top k edges and a non-causal part. Second, a distribution intervener, i.e., a random replacement from a set, creates perturbed distribution to infer the invariant causal parts. Finally, two classifiers are used to generate a joint prediction on causal and non-causal parts.

Table 7: Key highlights of explainability methods with *structural constraints*. Note that NC and GC denote node and graph classification respectively.

Method	Subgraph Extraction	Explanation form	Prediction/ classification module	Task
DIR [106]	Separating pattern that is invariant across interventional distribution	Invariant rationale	Seperate MLP for spurious and invariant parts	GC
ProtoGNN [124]	Computes similarity between Graph embedding and several learned diverse prototypes	Prototypes with high similarity	MLP with similarity scores as input	GC
SEGNN [12]	Finds K nodes that have similar structure and node features via contrastive loss	Similar nodes	classification via negative sampling of nodes	NC
KER-GNN [21]	Kernel filters integrated in message passing of GNNs	learned kernels and output node attributes	MLP with node attributes as input	NC, GC

ProtoGNN [124] combines prototype learning [82] with GNNs. Prototype learning is a form of case-based reasoning which makes predictions for new instances by comparing them with several learned exemplar cases also called prototypes. ProtoGNN computes the similarity scores between the graph embedding and multiple learned prototypes. Moreover, these prototypes are projected onto the nearest latent training subgraph during training using the Monte Carlo tree search [87, 7]. The similarity scores are used for the classification task where the subgraphs with high similarities can be used for explanation. In another work, for a given unlabeled node, **SEGNN** [12] finds k nearest labeled nodes that have structural and feature similarities and can be used for both generating predictions and explanations. It uses contrastive loss on node representations for feature similarity and also on edge representations of local neighborhood nodes for structural similarity. Moreover, the classification loss uses negative sampling with approximate k similar nodes. These k nearest nodes can be used to derive an explanation subgraph with threshold importance.

The method, **KER-GNN** [21] integrates graph kernels into the message-passing process of GNNs to increase the expressivity of GNNs beyond the 1-WL isomorphism test. In each layer, the node embeddings are updated by computing the similarity between the node's subgraph (the node with its ego-net) and trainable filters in the form of hidden graphs. The learned graph filters can provide important structural information about the data. Moreover, the output node attributes can be used to extract important substructures.

4 Counterfactual Explanation

Counterfactual methods provides an explanation by identifying the minimal alteration in the input graph that results in a change in the model's prediction. Recently, there have been several attempts to have explanations of graph neural networks (GNNs) via counterfactual reasoning. We classify these explainer methods that find counterfactuals into three major categories based on the type of methods: **(1) Perturbation-based**, **(2) Neural framework-based**, and **(3) Search-based**. We discuss the works in the individual categories below.

4.1 Perturbation-based methods

An intuitive way to generate counterfactuals for both the graph classification and the node classification task is to *alter the edges*, i.e., add or delete the edges in the graph such that it would change the prediction of the underlying GNN method. This alteration can be achieved by perturbing either the adjacency matrix or the computational graph of a node. The perturbation-based methods are summarized in Table 8.

One of the initial efforts, **CF-GNNExplainer** [55] aims to perturb the computational graph by using a binary mask matrix. It uses a binary matrix (all values are 0 or 1) P and modifies the computational graph matrix as $\tilde{A}_v = P \odot A_v$, where A_v is the original computational graph matrix and $\tilde{A}_v$ is computational graph matrix after the perturbation. The matrix P is computed by minimizing a combination of two different loss functions: L_{pred}, and L_{dist}. They are combined using a hyper-parameter in the final loss (L) as $L_{pred} + \beta L_{dist}$. The loss function,

Table 8: Key highlights of *perturbation-based* methods for counterfactuals

Method	Explanation Type	Downstream Task	Perturbation Target	Datasets Evaluated
CF-GNNExplainer [55]	Instance level	Node Classification	Computation graph	Tree-Cycles [114], Tree-Grids [114] BA-Shapes [114]
CF^2 [92]	Instance level	Graph Classification Node Classification	Original graph	BA-Shapes [114], Tree-Cycles [114] Mutag [14], NCI1 [98], CiteSeer [24]
GREASE [9]	Instance level	Node Ranking	Computation graph	LastFM, Yelp

L_{pred} quantifies the accuracy of the produced counterfactual, and L_{dist} captures the distance (or similarity) between the counterfactual graph and the original graph. In follow-up work, the method **CF^2** [92] extends the method in CF-GNNExplainer [55] by including a contrastive loss that jointly optimizes the quality of both the factual explanation and the counterfactual one. For an input graph, G, it aims to find an optimal subgraph G_s where G_s is a good factual explanation, and $G \backslash G_s$ is a good counterfactual. These objectives are formulated as a single optimization problem with the corresponding loss as $L_{overall} = \alpha L_{factual} + (1-\alpha) L_{counterfactual}$, where α is a hyperparameter.

Another method, **GREASE** [9] follows the standard technique of using a perturbation matrix to generate a counterfactual, but with two key modifications mainly to accommodate GNNs used for recommendation systems instead of classification tasks. In the recommendation task, GNNs rank the items (nodes) by assigning them a score instead of classifying them. GREASE uses a loss function based on the scores given by the GNN before and after the perturbation. This score helps to rank the items or nodes. The second modification is the perturbation matrix, which acts as the mask, and is used to perturb the computational graph (l-hop neighborhood of the node) instead of perturbing the entire graph. Here l denotes the number of layers in the GNN. Similar to CF^2 [92], GREASE also optimizes counterfactual and factual explanation losses, but not jointly.

In summary, all these techniques share similarities in computing the counterfactual similarity and constructing the search space. Similarity is measured by the number of edges removed from input instances and the search space is the set of all subgraphs obtained by edge deletions in the original graph. Because of the unrestricted nature of the search space, these methods might not be ideal for graphs such as molecules, where the validity of the subgraphs has valency restrictions. On the other hand, the mentioned methods differ mainly in the loss function formulations and the perturbation operations for the downstream tasks. For instance, CF-GNNExplainer [55] and GREASE [9] perform node classification and regression, they can use perturbations on the computation graph. However, CF^2 [92] considers both graph and node classification tasks, hence it uses perturbations on the entire graph, i.e., the adjacency matrix.

4.2 Neural framework-based methods

The approaches in this section use neural architectures to generate counterfactual graphs as opposed to the perturbation-based methods where the adjacency matrix of the input graph is minimally perturbed to generate counterfactuals. Table 9 summarizes these methods.

The objective of **RCExplainer** [4] is to identify a resilient subset of edges that, when removed, alter the prediction of the remaining graph. This is accomplished by modeling the implicit decision regions using graph embeddings. Even though the counterfactual graph generated by a neural architecture is used in conjunction with the adjacency matrix of the input graph, the counterfactual itself is not generated through perturbations on the adjacency matrix. RCExplainer addresses the issue of fragility where an interpretation is fragile (or non-robust) if systematic perturbations in the input graph can lead to dramatically different interpretations without changing the label. The standard explainers aim to generate good counterfactuals by choosing the closest counterfactual to the input instance and it might induce over-fitting. RCExplainer reduces this over-fitting by first clustering input graphs using polytopes, and finding good counterfactuals close to the cluster (polytope) instead of individual

Table 9: Key highlights of *neural framework-based* methods for counterfactuals

Method	Explanation Type	Downstream Task	Counterfactual Generator	Datasets Evaluated
RCExplainer [4]	Instance level	Graph classification Node classification	Edge prediction with Neural Network	Mutag [14], BA-2motifs [56], NCI1 [98] Tree-Cycles [114], Tree-Grids [114] BA-Shapes [56], BA-Community [114]
CLEAR [57]	Instance level	Graph classification Node classification	Graph generation with Variational Autoencoder	Community [18], Ogbg-molhiv, IMDB-M

instances. Another method, **CLEAR** [57] generates counterfactual graphs by leveraging a graph variational autoencoder. Two major issues often seen in other explainer methods, namely, generalization and causality are addressed in this paper.

Both methods use a generative neural model to find counterfactuals, but the generative model is different across the methods. While **RCExplainer** [4] uses a neural network that takes pairwise node embeddings and predict the existence of an edge between them, **CLEAR** [57] uses a variational autoencoder to generate a complete graph. This shows that while the former method cannot create nodes that are not present in the original graph, the latter can. In terms of the objective, the primary focus in **RCExplainer** [4] is the robustness of the generated counterfactual, but **CLEAR** [57] aims to generate counterfactuals that explain the underlying causality.

4.3 Search-based methods

These methods usually depend on search techniques over the counterfactual space for relevant tasks or applications (see the highlights in Table 10). For example, given an inactive molecule in a chemical reaction, the task is to find a similar but active molecule. Here, generative methods or perturbation methods might not be effective, and the perturbations might not even result in a valid molecule. In such cases, a good search technique through the space of counterfactuals could be more useful. An inherent challenge is that the search space of counterfactuals might be exponential in size. Hence, building efficient search algorithms is required.

The major application is finding counterfactual examples for molecules in related tasks. The method **MMACE** [101] finds counterfactuals for molecules. In the corresponding graph classification problem, it aims to classify a molecule based on a specific property. Examples include whether a molecule will permeate blood brain barrier and molecule's solubility. The search space can be generated by a method called *Superfast Traversal, Optimization, Novelty, Exploration and Discovery (STONED)* [71]. MMACE uses this method to generate the close neighbourhood and searches with a BFS-style algorithm to find an optimal set of counterfactuals.

Similarly, **MEG** [72] also aims to find a counterfactual and the search space consists of molecules. However, instead of searching the space with traditional graph search algorithms, MEG uses a reinforcement learning-based approach to navigate the search space more efficiently. The reward for finding a counterfactual is defined as the inverse of the probability that the candidate molecule found by the agent is not a counterfactual. This method is applied in a classification problem of predicting toxicity of a molecule as well as in a regression problem of predicting solubility of a molecule.

Another approach **GCFExplainer** [35] uses a random walk-based method to search the counterfactual space. The objective is not to find an individual counterfactual for each input sample but to find a small set of counterfactuals that explain all or a subset of the input samples. Hence, this is a global method (see Sec. 5.2). Here the counterfactual search space is obtained by applying graph edit operations on the training data. The method uses a random walk called **Vertex Reinforced Random Walk (VRRW)** [73], which is a modified version of a Markov chain where the state transition probabilities depend on the number of previous visits to that state.

Both **MMACE** [101] and **MEG** [72] are developed for GNNs that predict molecular properties while the objective of **GCFExplainer** [35] is to generate global explanations. However, the search algorithms and the generation mechanisms of the counterfactual space are quite different. For instance, MMACE employs a graph search algorithm to locate the nearest counterfactual instance. In contrast, MEG utilizes reinforcement learning,

Table 10: Key highlights of *search-based* methods for counterfactuals

Method	Explanation Type	Downstream Task	Counterfactual Similarity Metric	Datasets Evaluated
MMACE [101]	Instance level	Graph classification Node classification	Tanimoto similarity	Blood brain barrier dataset [62] Solubility data [88] HIV drug dataset [72]
GCFExplainer [35]	Global	Graph classification	Graph edit distance	Mutag [14], NCII [98]
MEG [72]	Instance level	Graph classification Node classification	Cosine Similarity Tanimoto similarity	Tox21 [40], ESOL [107]

and GCFExplainer employs random walks to achieve the same goal.

5 Others

In this section we describe the explainer methods in three special categories: **explainer for temporal GNNs, global explainers** and **causality-based explainers** in this section.

5.1 Explainers for Temporal GNNs

In temporal or dynamic graphs, the graph topology and node attributes evolve over time. For instance, in social networks the relationships can be dynamic, or in the citation networks co-authorships change over time. There has been effort towards explaining the GNN models that are specifically designed for such structures.

One of the earlier explainer methods on dynamic graphs is GCN-SE [20]. GCN-SE learns the attention weights in the form of linear combination of the representations of the nodes over multiple snapshots (i.e., over time). To quantify the explanatory power of the proposed method, importance of different snapshots are evaluated via these learned weights. Another method [31] designs a two-step process. It uses static explainer such as PGM-explaine [96] to explain the underlying temporal GNN (TGNN) model (such as TGCN [125]) for each time step separately and then it aims to discover the dominant explanations from the explanations identified by the static one. DGExplainer [109] also generates explanations for dynamic GNNs by computing the relevance scores that capture the contributions of each component for a dynamic graph. More specifically, it redistributes the output activation score to the relevance of the neurons of its previous layer in the model. This process iterates until the relevance scores of the input neuron are obtained. Recently, T-GNNExplainer [108] has been proposed for temporal graph explanation where a temporal graph constituted by a sequence of temporal events. T-GNNExplainer solves the problem of finding a small set of previous events that are responsible for the model's prediction of the target event. In [51], the approach involves a smooth parameterization of the GNN predicted distributions using axiomatic attribution. These distributions are assumed to be on a low-dimensional manifold. The approach models the distributional evolution as smooth curves on the manifold and reparameterize families of curves by designing a convex optimization problem. The aim is to find a unique curve that approximates the distributional evolution and will be useful for human interpretation.

The following ones also design explainers of temporal GNNs but with specific objectives or applications. [97] studies the limit of perturbation-based explanation methods. The approach constructs some specific instances of TGNNs and evaluate how reliably node-perturbation, edge-perturbation or both can reliably identify specific graph components carrying out the temporal aggregation in temporal GNNs. In [113], a novel interpretable model on temporal heterogeneous graphs has been proposed. The method constructs temporal heterogeneous graphs which represent the research interests of the target authors. After the detection task, a deep neural network has been used for the generation process of interpretation on the predicted results. This method has been applied to research interest shift detection of researchers. Another related work [29] is on explaining GraphRC which is a special type of GNN and popular because of its training efficiency. The proposed method explores the specific role played by each reservoir node (neuron) of GraphRC by using attention mechanism on the distinct temporal patterns in the reservoir nodes.

5.2 Global Explainers

Majority of the explainers provide explanation for specific instances and can be seen as *local explainers*. However, global explainers aim to explain the overall behavior of the model by finding common input patterns that explain certain predictions [118]. Global explainers provide a high-level and generic explanation compared to local explainers. However, local explainers can be more accurate compared to global explainers [118] especially for individual instances. We categorize global explainers into following three types.

(1) Generation-based: These post-hoc methods use either a generator or generative modeling to find explanations. For instance, **XGNN** [118] uses a reinforcement learning (RL) based graph generator optimized using policy gradient. In contrast, **GNNInterpreter** [100] is a generative global explainer that maximizes the likelihood of explanation graph being predicted as the target class by the model (the details are in Sec. 3.1.5).

(2) Concept-based: These methods provide concept based explanations. Concepts are small higher level units of information that can be interpreted by humans [25]. The methods differ in the approaches to find concepts. **GCExplainer** [60] adapts an image explanation framework known as automated concept based explanation (ACE) [25] to find global explanation for graphs. It finds concepts by clustering the embeddings from the last layer of the GNN. A concept and its importance are represented by a cluster and the number of nodes in it respectively. Another method **GCneuron** [112], which is inspired by Compositional Explanations of Neurons [70], finds global explanation for GNNs by finding compositional concepts aligned with neurons. A base concept is a function C on Graph G that produces a binary mask over all the input nodes V. A compositional concept is logical combination of base concepts. This method uses beam search to find compositional concept that minimizes the divergence between the concept and the neuron activation. Lastly, **GLGExplainer** [2] uses local explanations from PGExplainer [56] and projects them to a set of learned prototype or concepts (similar to ProtGNN [124]) to derive a concept vector. A concept vector is vector of distances between graph explanation and each prototype. This concept vector is then used to train an Entropy based logic explainable network (E-LEN) [10] to match the prediction of the class. The logic formula from the entropy layer for each class acts as explanations.

(3) Counterfactual: **Global counterfactual explainer** [35] finds a candidate set of counterfactuals using vertex re-inforced random walk. It then uses a greedy strategy to select the top k counterfactuals from the candidate set as global explanations. We explain it in more detail in Sec. 4.3.

5.3 Causality-based Explainers

Most of the GNN classifiers learn all statistical correlation between the label and input features. As a result, these may not distinguish between causal and non-causal features and may make prediction using shortcut features [89]. Shortcut features serve as confounders between the causal features and the prediction. Methods in this category attempt to reduce the confounding effect so that the model exploits causal substructure for prediction and these substructures also act as explanations. They can be categorized into the followings.

Self-interpretable methods. Methods in this category have the explainer architecture inbuilt into the model. One of the methods, **DIR** [106] creates multiple interventional distribution by conducting intervention on the training distribution. The invariant part across these distributions is considered as the causal part (see details in Sec. 3.2.2). **CAL** [89] uses edge and node attention to estimate causal and shortcut features of the graph. Two classifiers are used to make prediction on causal and shortcut features respectively. Loss on causal features is used as classification loss. Moreover, KL divergence is used to push the prediction based on shortcut features to have uniform distribution across classes. Finally, CAL creates an intervention graph in the representation space via random additions. The loss on this intervened graph classification is considered as the causal loss. These three loss terms are used to reduce the confounding effect and find the causal substructure that acts as explanation. **DisC** [19] uses a disentangled GNN framework to separate causal and shortcut substructures. It first learns a edge mask generator that divides the input into causal and shortcut substructures. Two separate GNNs are trained to produce disentangled representation of these substructures. Finally, these representations are used

to generate unbiased counterfactual samples by randomly permuting the shortcut representation with the causal representation.

Generation-based methods. These methods use generative modeling to find explanations and are post-hoc. **GEM [49]** trains a generative auto-encoder by finding the causal contribution of each edge in the computation graph (details are in Sec. 3.1.5). While GEM focuses on the graph space, **OrphicX** [50] identifies the causal factors in the embedding space. It trains a variational graph autoencoder (VGAE) that has an encoder and a generator. The encoder outputs latent representations of causal and shortcut substructures of input. Generator uses both of these representations to generate the original graph and the causal representation to produce a causal mask on original graph. The information flow between the latent representation of the causal substructure and the prediction is maximized to train the explainer. The causal substructure also acts as an explanation.

6 Applications

We describe the explainers methods that are relevant for specific applications in different domains such as in social networks, biology, and computer security.

Computer Security. This work [30] focuses on designing an explanation framework for cybersecurity applications using GNN models by identifying the important nodes, edges, and attributes that are contributing to the prediction. The applications include code vulnerability detection and smart contract vulnerability detection. Another work [33] proposes CFGExplainer for GNN oriented malware classification and identifies a subgraph of the malware control flow graph that is most important for the classification. Some other work focuses on the problem of botnet detection. The first method BD-GNNExplainer [127] extracts the explainer subgraph by reducing the loss between the classification results generated by the input subgraph and the entire input graph. The XG-BoT detector proposed in [53] detects malicious botnet nodes in botnet communication graphs. The explainer is based on the GNNExplainer and saliency map in the XG-BoT.

Social Networks. A recent work [79] studies the problem of detecting fake news spreaders in social networks. The proposed method SCARLET is a user-centric model that uses a GNN with attention mechanism. The attention scores help in computing the importance of the neighbors. The findings include that a person's decision to spread false information is dependent on its perception (or trust dynamics) of neighbor's credibility. On the other hand, **GCAN** [54] uses sequence models. The aim is to find a fake tweet based on the user profile and the sequence of its retweets. The sequence models and GNNs help to learn representation of retweet propagation and representation of user interactions respectively. A co-attention mechanism is further used to learn the correlation between source tweet and retweet propagation and make prediction. In [58], a GNN model has been proposed along with the explanation of its prediction for the problem on drug abuse in social networks.

Computational Biology. One of the long standing problems in neuroscience is the understanding of Brain networks, especially understanding the Regions of Interests (ROIs) and the connectivity between them. These regions and their connectivity can be modelled as a graph. A recent work on explainability, IBGNN [11] explores the explainable GNN methods to solve the task of identifying ROIs and their connectivity that are indicative of brain disorders. It uses a perturbation matrix to create an edge mask, and extracts important edges and nodes. A few more works also focus on the same task of identifying ROIs, but use different explanation techniques. In [63], the method uses a perturbation matrix with feature masks and optimizes mutual information to find the explanations. This work [126] uses Grad-CAM [76] to find important ROIs. [1] uses a search-based method to extract counterfactuals, which can serve as good candidates for important ROIs. The method uses graph edit operations to navigate from input graph to a counterfactual, but it optimizes this by using a lookup database to select edges that are the most effective in discriminating between different predicted classes. As

another interesting application, this work [75] explores is related to the extraction of subgraphs in protein-protein interaction (PPI) network, where the downstream task is to detect the relevance of a protein to cancer.

Chemistry. GNNs are being used to study molecular properties extensively and often requires explanations to better understand the model's predictions. A recent work [32] focuses on improving self-interpretability of GCNs by imposing orthogonality of node features and sparsity of the GCN's weights using Gini regularization. The intuition behind the orthogonality of features is driven by the assumption that atoms in a molecule can be represented by a linear combination of orthonormal basis of wavefunctions. Another method, APRILE [111] aims at finding the parts of a drug molecule responsible for side effects. It uses perturbation techniques to extract an explanation. In drug design, the method in [38] uses integrated gradients [91] to assign importance to atoms (nodes) and the atomic properties (node features) to understand the properties of a drug.

Pathology. In medical diagnosis a challenging task is to understand the reason behind a particular diagnosis, whether it is made by a human or a machine learning system. To this end, explainer frameworks for the machine learning models become useful. In many cases the diagnosis data can be represented by graphs. This work [105] builds a graph using words, entities, clauses and sentences extracted from a patient's electronic medical record (EMR). The objective is to extract the entities most relevant for the diagnosis by training an edge mask, and is achieved by minimizing the sum of the elements in the mask matrix. Another method [37] focuses on generating explanations for histology (micro-anatomy) images. It first converts the image into a graph of biological entities, where the nodes could be cells, tissues or some other task specific biological features. Afterwards the standard explainer techniques described in gradient 3.1.2 or perturbation 3.1.4 based methods are used to generate the explanations. Another work [116] in this field modifies the objective to optimise for both necessity and sufficiency (Sec. 8). The explanation is generated in such a way that the mutual information between explanation subgraph and the prediction is maximized. Additionally, the mutual information between the remaining graph after removing the explanation subgraph and the prediction is minimized.

7 Datasets

A set of synthetic as well as real-world datasets have been used for evaluating the proposed explainers in several tasks such as node classification and graph classification. Table 11 lists down the set of datasets and the corresponding explanation types and tasks used in the literature.

7.1 Synthetic datasets

Annotating ground truth explanations in graph data is laborious and requires domain expertise. To overcome this challenge, several explainers have been evaluated using synthetic datasets that are created using certain motifs as ground truth values. We highlight *six* popular synthetic datasets:
BA-Shapes [114]: This graph is formed by randomly connecting a base graph to a set of motifs. The base graph is a Barabasi-Albert (BA) graph with 300 nodes. It includes 80 house-structured motifs with five nodes each, formed by a top, a middle, and a bottom node type.
BA-Community [114]: The BA-community graph is a combination of two BA-Shapes graphs. The features of each node are assigned based on two Gaussian distributions. Also, nodes are assigned a class out of eight classes based on the community they belong to.
Tree Cycle [114]: This consists of a 8-level balanced binary tree as a base graph. To this base graph, 80 cycle motifs with six nodes each are randomly connected. It just has two classes; one for the nodes in the base graph and another for nodes in the defined motif.

Table 11: It shows the datasets for different categories, explanation types and tasks.

Dataset	References	Nature	Explanation Type	Task
BA-Shapes	[114, 96, 123, 56, 49, 55, 4]	Synthetic	Compared to Motif	Node classification
BA-Community	[84, 114, 56, 4, 57]	Synthetic	Compared to Motif	Node classification
Tree Cycle	[49, 56, 123, 4, 55]	Synthetic	Compared to Motif	Node classification
Tree Grids	[114, 56, 123, 49, 4, 55]	Synthetic	Compared to Motif	Node classification
BA-2Motif	[56, 121, 68, 4]	Synthetic	Compared to Motif	Graph classification
Spurious Motifs	[68, 106]	Synthetic	Compared to Motif	Graph classification
Mutagenicity	[114, 56, 49, 124, 121, 4, 118]	Real-World	Compared to Chemical property	Graph classification
NCI1	[49, 4]	Real-World	Compared to Chemical property	Graph classification
BBBP	[124, 92, 107]	Real-World	Compared to Chemical property	Graph classification
Tox21	[72, 107]	Real-World	Compared to Chemical property	Graph classification
MNIST-75sp	[96, 68, 43]	Real-World	Visual	Graph classification
Sentiment Graphs	[121, 68, 124, 120]	Real-World	Visual	Graph classification

Tree Grids [114]: This graph uses same base graph but a different motif set compared to the tree cycle graph. It uses 3 by 3 grid motifs instead of the cycle motifs.
BA-2Motifs [56]: This is used for graph classification and has two classes. The base graph is BA graph for both the classes. However, one class has a house-structure motif and another has a 5-node cycle motif.
Spurious Motifs [106]: With 18000 graphs in the dataset, each graph is a combination of one base S (Tree, Ladder or Wheel) and one motif C (Cycle, House, Crane). Ground-truth Y is determined by the motif. A spurious relation between S and Y is manually induced. This spurious correlation can be varied based on a parameter that ranges from 0 to 1.

7.2 Real-world datasets

Due to the known chemical properties of the molecules, molecular graph datasets become a good choice for evaluating the generated explanation structure. We highlight some widely used molecular datasets for evaluating explainers in the *graph classification task*.
Mutag [14]: This consists of 4337 molecules (graphs) with two classes based on the mutagenic effect. Using domain knowledge, specific chemical groups are assigned as ground truth explanations.
NCI1 [98]: It is a graph classification dataset with 4110 instances. Each graph is a chemical compound where a node represents an atom and an edge represents a bond between atoms. Each molecule is screened for activity against non-small cell lung cancer or ovarian cancer cell lines.
BBBP [107]: Similar to Mutag, Blood-brain barrier penetration (BBBP) is also a molecule classification dataset with two classes with 2039 compounds. Classification is based on their permeability properties.
Tox21 [107]: This dataset consists of 7831 molecules with 12 different categories of chemical compounds. The categorization is based on the chemical structures and properties of those compounds.

Visual explanation can be an important component of comparing explainers. Hence, researchers also use datasets that do not have ground truth explanations but can be visually evaluated through generated examples. Below are some of the datasets used for visual analysis:
MNIST-75sp [43]: An MNIST image is converted to a super-pixel graph with at most 75 nodes, where each node denotes a "super pixel". Pixel intensity and coordinates of their centers of masses are used as the node attributes. Edges are formed based on the spatial distance between the super-pixel centers. Each graph is assigned one of the 10 MNIST classes, i.e., numerical digits.
Sentiment Graphs [120]: Graph SST2, Graph SST5, and Graph Twitter are based on text sentiment analysis data of SST2, SST5, and Twitter datasets. A graph is constructed by considering tokens as nodes, relations as edges, and sentence sentiment as its label. The BERT architecture is used to obtain 768-dimensional word embeddings for the dataset. The generated explanation graph can be evaluated for its textual meaning.

8 Evaluation

The evaluation of the explainer methods is based on the quality of the explainer's ability to generate human-intelligible explanations about the model prediction. As this might be subjective depending on the applications in hand, the evaluation measures consider both quantitative and qualitative metrics.

8.1 Quantitative Evaluation

Quantitative evaluation metrics help in having a standardized evaluation that is free of human bias. For this, explainability is posed as a binary classification problem. The explainers assign a score to the *node features*, *edges*, and *motifs*, which are the most responsible for the prediction according to the explainer. We are also provided with the ground-truth binary labels for the features and structures based, denoting whether they are responsible for the prediction or not. The explainer is then evaluated by comparing these scores to the ground-truth explanation labels using different methods:

Accuracy [55, 92]: To find the accuracy, the top-k edges produced by the explainer are set to be positive, and the rest are negative. These top-k edges and the ground-truth labels are compared to compute the accuracy.

Area Under Curve (AUC) [123, 101, 4]: We compare the top-k raw scores directly against the ground-truth labels by computing the area under the ROC curve.

Fidelity [115, 55, 4]: This is used for explainers that generate a subgraph as the explanation. It compares the performances of the base GNN model on the input graph and the explainer subgraph. Let N be the number of samples, y_i is the true label of sample i, $\hat{y}_i$ is the predicted label of sample i, $\hat{y}^k$ is the predicted label after choosing the subgraph formed by nodes with top-$k\%$ nodes, and $\mathbb{1}[\cdot]$ is the indicator function. Fidelity measures how close the predictions of the explanation sub-graph are to the input graph. For factual explainers, the lower this value, the better is the explanation. It is formally defined as follows:

$$\text{Fidelity} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}[y_i = \hat{y}_i] - \mathbb{1}[y_i = \hat{y}_i^k]$$

Sparsity [115, 55]: It measures the conciseness of explanations (e.g., the sub-graphs) that are responsible for the final prediction. Let $|p_i|$ and $|G_i|$ denote the number of edges in the explanation, and the same in the original input graph, respectively. The sparsity is then defined as follows:

$$\text{Sparsity} = 1 - \frac{1}{N}\sum_{i=1}^{N} \frac{|p_i|}{|G_i|}$$

Robustness [4]: It quantifies how resistant an explainer is to perturbations on input graph. Here perturbations are addition or deletion of edges randomly such that it does not change the prediction of the underlying GNN. The robustness is the percentage of graphs for which these perturbations do not change the explanation.

Probability of Sufficiency (PS) [92, 9]: It is the percentage of graphs for which the explanation graph is sufficient to generate the same prediction as the original input graph.

Probability of Necessity (PN) [92, 9]: It is the percentage of graphs for which the explanation graph when removed from the original input graph will alter the prediction made by the GNN.

Generalization [84]: This measures the capability of generalization of the explainer method in an inductive setting. To measure this, the training dataset size is usually varied and the AUC scores are computed for these tests. Generalisation plays an important role in explanability as the good generalizable models are generally sparse in terms of inputs. This metric is highly relevant for the self-interpretable models.

8.2 Qualitative Evaluation

Explanations can also be evaluated qualitatively using expert domain knowledge. This mode of evaluation is crucial especially while working with real-world datasets that do not have ground truth labels.
Domain Knowledge [114]: Generated explanations can be evaluated for their meaning in the application domain. For example, GNNExplainer [114] correctly identifies the carbon ring as well as chemical groups NH2 and NO2, which are known to be mutagenic.
Manual Scoring [96]: Another method of evaluating the explanations is by asking users (e.g., domain experts) to score, say on a scale of 1-10, the explanations generated by various explainers and compare them. One can also use RMSE scores to quantitatively compare these explainers based on the scores.

9 Future Directions

Combinatorial problems: Most of the existing explanation frameworks are for prediction tasks such as node and graph classification. However, graphs are prevalent in various domains, such as in social networks [39, 67], healthcare [102], and infrastructure development [65, 64]. Solving combinatorial optimization problems on graphs is a common requirement in these domains. Several architectures based on Graph Neural Networks (GNNs) [41, 61, 77] have been proposed to tackle these problems that are usually computationally hard. However, the explainability of these methods for such combinatorial problems is largely missing. One potential direction is to build frameworks that can explain the behavior of the solution set in such problems.
Global methods: Most explainers primarily adopt a local perspective by generating examples specific to individual input graphs. From global explanations, we can extract higher-level insights that complement the understanding gained from local explanations (see details on global methods in Sec. 5.2). Moreover, global explanations can be easily understood by humans even for large datasets. Real-world graph datasets often consist of millions of nodes. When generating explanations specific to each instance, the number of explanations increases proportionally with the size of the dataset. As a result, the sheer volume of explanations becomes overwhelming for human cognitive capabilities to process effectively. Global approaches can immensely help in these scenarios.
Visualization and HCI tools: Graph data, unlike textual and visual data, cannot be perceived by human senses. Thus, qualitative evaluation of explanation becomes a non-trivial problem and often requires expert guidance [114, 96]. This makes crowdsourcing evaluations difficult and not scalable. Other ways to qualitatively assess graph structures for explanation of a certain prediction can be explored. Additionally, since explainability is human-centric, it is crucial that explainers are influenced by human cognition and behavior, particularly those of domain experts [48] while using GNNs in making important decisions [66]. HCI research can help in designing the interface for the experts to assess the generated explanation graphs [55].
Temporal GNNs: Temporal graph models are designed to predict the graph structure and labels in the future by exploiting how the graph has evolved in the past. This increases the complexity of explanations significantly as they now involve combinations of graph structures at different time intervals. Existing methods [20, 109, 31, 44] mostly focus on discrete-time models where graphs are provided at different points in time. Future works can explore ways to explain the prediction of a continuous-time dynamic graph model, where interactions happen in real time [108]. One direction could be to optimize over a parameterized temporal point process [94].

10 Conclusions

In this survey, we have provided a comprehensive overview of explanation methods for Graph Neural Networks (GNNs). Besides outlining some background on GNNs and explainability, we have presented a detailed taxonomy of the papers from the literature. By categorizing and discussing these methods, we have highlighted their

strengths, limitations, and applications in understanding GNN predictions. Moreover, we have highlighted some widely used datasets and evaluation metrics in assessing the explainability of GNNs. As GNNs continue to play a significant role in various fields, such as healthcare, recommendation systems, and natural language processing, the need for interpretable and transparent models becomes increasingly important. Overall, we believe this survey serves as a valuable resource for researchers and practitioners interested in the explainability of GNNs and provides a foundation for further advancements in interpretable graph representation learning.

References

[1] Carlo Abrate and Francesco Bonchi. Counterfactual graphs for explainable classification of brain networks. In ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pages 2495–2504, 2021.

[2] Steve Azzolin, Antonio Longa, Pietro Barbiero, Pietro Liò, and Andrea Passerini. Global explainability of gnns via logic combination of learned concepts. arXiv preprint arXiv:2210.07147, 2022.

[3] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PloS one, 10(7):e0130140, 2015.

[4] Mohit Bajaj, Lingyang Chu, Zi Yu Xue, Jian Pei, Lanjun Wang, Peter Cho-Ho Lam, and Yong Zhang. Robust counterfactual explanations on graph neural networks. Advances in Neural Information Processing Systems, 34:5644–5655, 2021.

[5] Federico Baldassarre and Hossein Azizpour. Explainability techniques for graph convolutional networks. arXiv preprint arXiv:1905.13686, 2019.

[6] Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J Hu, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations. arXiv preprint arXiv:2111.09266, 2021.

[7] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games, 4(1):1–43, 2012.

[8] Nadia Burkart and Marco F Huber. A survey on the explainability of supervised machine learning. Journal of Artificial Intelligence Research, 70:245–317, 2021.

[9] Ziheng Chen, Fabrizio Silvestri, Jia Wang, Yongfeng Zhang, Zhenhua Huang, Hongshik Ahn, and Gabriele Tolomei. Grease: Generate factual and counterfactual explanations for gnn-based recommendations. arXiv preprint arXiv:2208.04222, 2022.

[10] Gabriele Ciravegna, Pietro Barbiero, Francesco Giannini, Marco Gori, Pietro Lió, Marco Maggini, and Stefano Melacci. Logic explained networks. Artificial Intelligence, 314:103822, 2023.

[11] Hejie Cui, Wei Dai, Yanqiao Zhu, Xiaoxiao Li, Lifang He, and Carl Yang. Interpretable graph neural networks for connectome-based brain disorder analysis. In Medical Image Computing and Computer Assisted Intervention–MICCAI 2022: 25th International Conference, Singapore, September 18–22, 2022, Proceedings, Part VIII, pages 375–385. Springer, 2022.

[12] Enyan Dai and Suhang Wang. Towards self-explainable graph neural network. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pages 302–311, 2021.

[13] Enyan Dai, Tianxiang Zhao, Huaisheng Zhu, Junjie Xu, Zhimeng Guo, Hui Liu, Jiliang Tang, and Suhang Wang. A comprehensive survey on trustworthy graph neural networks: Privacy, robustness, fairness, and explainability. arXiv preprint arXiv:2204.08570, 2022.

[14] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. Journal of medicinal chemistry, 34(2):786–797, 1991.

[15] John S. Delaney. Esol: Estimating aqueous solubility directly from molecular structure. Journal of Chemical Information and Computer Sciences, 44(3):1000–1005, 2004. PMID: 15154768.

[16] Monroe D Donsker and SR Srinivasa Varadhan. Asymptotic evaluation of certain markov process expectations for large time, i. Communications on Pure and Applied Mathematics, 28(1):1–47, 1975.

[17] Alexandre Duval and Fragkiskos D Malliaros. Graphsvx: Shapley value explanations for graph neural networks. In Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part II 21, pages 302–318. Springer, 2021.

[18] P Erd. os and a. r enyi,\on random graphs i. Publ. Math. Debrecen, 6:290–297, 1959.

[19] Shaohua Fan, Xiao Wang, Yanhu Mo, Chuan Shi, and Jian Tang. Debiasing graph neural networks via learning disentangled causal substructure. arXiv preprint arXiv:2209.14107, 2022.

[20] Yucai Fan, Yuhang Yao, and Carlee Joe-Wong. Gcn-se: Attention as explainability for node classification in dynamic graphs. In 2021 IEEE International Conference on Data Mining (ICDM), pages 1060–1065. IEEE, 2021.

[21] Aosong Feng, Chenyu You, Shiqiang Wang, and Leandros Tassiulas. Kergnns: Interpretable graph neural networks with graph kernels. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 36, pages 6614–6622, 2022.

[22] Qizhang Feng, Ninghao Liu, Fan Yang, Ruixiang Tang, Mengnan Du, and Xia Hu. Degree: Decomposition based explanation for graph neural networks. In International Conference on Learning Representations, 2022.

[23] Thorben Funke, Megha Khosla, Mandeep Rathee, and Avishek Anand. Z orro: Valid, sparse, and stable explanations in graph neural networks. IEEE Transactions on Knowledge and Data Engineering, 2022.

[24] L Getoor. Advanced methods for knowledge discovery from complex data. Link-based Classification, 189:207, 2005.

[25] Amirata Ghorbani, James Wexler, James Y Zou, and Been Kim. Towards automatic concept-based explanations. Advances in Neural Information Processing Systems, 32, 2019.

[26] Zhimeng Guo, Teng Xiao, Charu Aggarwal, Hui Liu, and Suhang Wang. Counterfactual learning on graphs: A survey. arXiv preprint arXiv:2304.01391, 2023.

[27] Gérard Hamiache and Florian Navarro. Associated consistency, value and graphs. International Journal of Game Theory, 49:227–249, 2020.

[28] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. Advances in neural information processing systems, 30, 2017.

[29] Xinyu Han and Yi Zhao. Interpretable graph reservoir computing with the temporal pattern attention. IEEE Transactions on Neural Networks and Learning Systems, 2022.

[30] Haoyu He, Yuede Ji, and H Howie Huang. Illuminati: Towards explaining graph neural networks for cybersecurity analysis. In 2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P), pages 74–89. IEEE, 2022.

[31] Wenchong He, Minh N Vu, Zhe Jiang, and My T Thai. An explainer for temporal graph neural networks. In GLOBECOM 2022-2022 IEEE Global Communications Conference, pages 6384–6389. IEEE, 2022.

[32] Ryan Henderson, Djork-Arné Clevert, and Floriane Montanari. Improving molecular graph neural network explainability with orthonormalization and induced sparsity. In International Conference on Machine Learning, pages 4203–4213. PMLR, 2021.

[33] Jerome Dinal Herath, Priti Prabhakar Wakodikar, Ping Yang, and Guanhua Yan. Cfgexplainer: Explaining graph neural network-based malware classification from control flow graphs. In 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pages 172–184. IEEE, 2022.

[34] Qiang Huang, Makoto Yamada, Yuan Tian, Dinesh Singh, and Yi Chang. Graphlime: Local interpretable model explanations for graph neural networks. IEEE Transactions on Knowledge and Data Engineering, 2022.

[35] Zexi Huang, Mert Kosan, Sourav Medya, Sayan Ranu, and Ambuj Singh. Global counterfactual explainer for graph neural networks. In Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining, pages 141–149, 2023.

[36] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. arXiv preprint arXiv:1611.01144, 2016.

[37] Guillaume Jaume, Pushpak Pati, Behzad Bozorgtabar, Antonio Foncubierta, Anna Maria Anniciello, Florinda Feroce, Tilman Rau, Jean-Philippe Thiran, Maria Gabrani, and Orcun Goksel. Quantifying explainers of graph neural networks in computational pathology. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 8106–8116, 2021.

[38] José Jiménez-Luna, Miha Skalic, Nils Weskamp, and Gisbert Schneider. Coloring molecules with explainable artificial intelligence for preclinical relevance assessment. Journal of Chemical Information and Modeling, 61(3):1083–1094, 2021.

[39] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 137–146, 2003.

[40] Kristian Kersting, Nils M Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels. 2016.

[41] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. Advances in neural information processing systems, 30, 2017.

[42] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.

[43] Boris Knyazev, Graham W Taylor, and Mohamed Amer. Understanding attention and generalization in graph neural networks. Advances in neural information processing systems, 32, 2019.

[44] Mert Kosan, Arlei Silva, Sourav Medya, Brian Uzzi, and Ambuj Singh. Event detection on dynamic graphs. arXiv preprint arXiv:2110.12148, 2021.

[45] Mert Kosan, Arlei Silva, and Ambuj Singh. Robust ante-hoc graph explainer using bilevel optimization. arXiv preprint arXiv:2305.15745, 2023.

[46] Wenqian Li, Yinchuan Li, Zhigang Li, Jianye Hao, and Yan Pang. Dag matters! gflownets enhanced explainer for graph neural networks. arXiv preprint arXiv:2303.02448, 2023.

[47] Yiqiao Li, Jianlong Zhou, Sunny Verma, and Fang Chen. A survey of explainable graph neural networks: Taxonomy and evaluation metrics. arXiv preprint arXiv:2207.12599, 2022.

[48] Q Vera Liao and Kush R Varshney. Human-centered explainable ai (xai): From algorithms to user experiences. arXiv preprint arXiv:2110.10790, 2021.

[49] Wanyu Lin, Hao Lan, and Baochun Li. Generative causal explanations for graph neural networks. In International Conference on Machine Learning, pages 6666–6679. PMLR, 2021.

[50] Wanyu Lin, Hao Lan, Hao Wang, and Baochun Li. Orphicx: A causality-inspired latent variable model for interpreting graph neural networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 13729–13738, 2022.

[51] Yazheng Liu, Xi Zhang, and Sihong Xie. A differential geometric view and explainability of gnn on evolving graphs. In The Eleventh International Conference on Learning Representations, 2023.

[52] Yunchao Liu, Yu Wang, Oanh T Vu, Rocco Moretti, Bobby Bodenheimer, Jens Meiler, and Tyler Derr. Interpretable chirality-aware graph neural network for quantitative structure activity relationship modeling in drug discovery. bioRxiv, pages 2022–08, 2022.

[53] Wai Weng Lo, Gayan Kulatilleke, Mohanad Sarhan, Siamak Layeghy, and Marius Portmann. Xg-bot: An explainable deep graph neural network for botnet detection and forensics. Internet of Things, 22:100747, 2023.

[54] Yi-Ju Lu and Cheng-Te Li. Gcan: Graph-aware co-attention networks for explainable fake news detection on social media. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 505–514, 2020.

[55] Ana Lucic, Maartje A Ter Hoeve, Gabriele Tolomei, Maarten De Rijke, and Fabrizio Silvestri. Cf-gnnexplainer: Counterfactual explanations for graph neural networks. In International Conference on Artificial Intelligence and Statistics, pages 4499–4511. PMLR, 2022.

[56] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. Parameterized explainer for graph neural network. Advances in neural information processing systems, 33:19620–19631, 2020.

[57] Jing Ma, Ruocheng Guo, Saumitra Mishra, Aidong Zhang, and Jundong Li. Clear: Generative counterfactual explanations on graphs. arXiv preprint arXiv:2210.08443, 2022.

[58] Zuanjie Ma, Hongming Gu, and Zhenhua Liu. Understanding drug abuse social network using weighted graph neural networks explainer. In Computational Science and Its Applications–ICCSA 2021: 21st International Conference, Cagliari, Italy, September 13–16, 2021, Proceedings, Part III 21, pages 52–61. Springer, 2021.

[59] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. arXiv preprint arXiv:1611.00712, 2016.

[60] Lucie Charlotte Magister, Dmitry Kazhdan, Vikash Singh, and Pietro Liò. Gcexplainer: Human-in-the-loop concept-based explanations for graph neural networks. arXiv preprint arXiv:2107.11889, 2021.

[61] Sahil Manchanda, Akash Mittal, Anuj Dhawan, Sourav Medya, Sayan Ranu, and Ambuj Singh. Gcomb: Learning budget-constrained combinatorial algorithms over billion-sized graphs. Advances in Neural Information Processing Systems, 33:20000–20011, 2020.

[62] Ines Filipa Martins, Ana L Teixeira, Luis Pinheiro, and Andre O Falcao. A bayesian approach to in silico blood-brain barrier penetration modeling. Journal of chemical information and modeling, 52(6):1686–1697, 2012.

[63] Chiara Mauri, Stefano Cerri, Oula Puonti, Mark Mühlau, and Koen Van Leemput. Accurate and explainable image-based prediction using a lightweight generative model. In Medical Image Computing and Computer Assisted Intervention–MICCAI 2022: 25th International Conference, Singapore, September 18–22, 2022, Proceedings, Part VIII, pages 448–458. Springer, 2022.

[64] Sourav Medya, Petko Bogdanov, and Ambuj Singh. Towards scalable network delay minimization. In 2016 IEEE 16th International Conference on Data Mining (ICDM), pages 1083–1088. IEEE, 2016.

[65] Sourav Medya, Sayan Ranu, Jithin Vachery, and Ambuj Singh. Noticeable network delay minimization via node upgrades. Proceedings of the VLDB Endowment, 11(9):988–1001, 2018.

[66] Sourav Medya, Mohammad Rasoolinejad, Yang Yang, and Brian Uzzi. An exploratory study of stock price movements from earnings calls. In Companion Proceedings of the Web Conference 2022, pages 20–31, 2022.

[67] Sourav Medya, Arlei Silva, and Ambuj Singh. Approximate algorithms for data-driven influence limitation. IEEE Transactions on Knowledge and Data Engineering, 34(6):2641–2652, 2020.

[68] Siqi Miao, Mia Liu, and Pan Li. Interpretable and generalizable graph learning via stochastic attention mechanism. In International Conference on Machine Learning, pages 15524–15543. PMLR, 2022.

[69] Siqi Miao, Yunan Luo, Mia Liu, and Pan Li. Interpretable geometric deep learning via learnable randomness injection. arXiv preprint arXiv:2210.16966, 2022.

[70] Jesse Mu and Jacob Andreas. Compositional explanations of neurons. Advances in Neural Information Processing Systems, 33:17153–17163, 2020.

[71] AkshatKumar Nigam, Robert Pollice, Mario Krenn, Gabriel dos Passos Gomes, and Alan Aspuru-Guzik. Beyond generative models: superfast traversal, optimization, novelty, exploration and discovery (stoned) algorithm for molecules using selfies. Chemical science, 12(20):7079–7090, 2021.

[72] Danilo Numeroso and Davide Bacciu. Meg: Generating molecular counterfactual explanations for deep graph networks. In 2021 International Joint Conference on Neural Networks (IJCNN), pages 1–8. IEEE, 2021.

[73] Robin Pemantle. Vertex-reinforced random walk. Probability Theory and Related Fields, 92(1):117–136, 1992.

[74] Tamara Pereira, Erik Nascimento, Lucas E Resck, Diego Mesquita, and Amauri Souza. Distill n'explain: explaining graph neural networks using simple surrogates. In International Conference on Artificial Intelligence and Statistics, pages 6199–6214. PMLR, 2023.

[75] Bastian Pfeifer, Anna Saranti, and Andreas Holzinger. Gnn-subnet: disease subnetwork detection with explainable graph neural networks. Bioinformatics, 38(Supplement_2):ii120–ii126, 2022.

[76] Phillip E Pope, Soheil Kolouri, Mohammad Rostami, Charles E Martin, and Heiko Hoffmann. Explainability methods for graph convolutional neural networks. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 10772–10781, 2019.

[77] Rishabh Ranjan, Siddharth Grover, Sourav Medya, Venkatesan Chakaravarthy, Yogish Sabharwal, and Sayan Ranu. Greed: A neural framework for learning graph distance functions. In Advances in Neural Information Processing Systems, 2022.

[78] Susie Xi Rao, Shuai Zhang, Zhichao Han, Zitao Zhang, Wei Min, Zhiyao Chen, Yinan Shan, Yang Zhao, and Ce Zhang. xfraud: explainable fraud transaction detection. Proceedings of the VLDB Endowment, (3):427–436, 2021.

[79] Bhavtosh Rath, Xavier Morales, and Jaideep Srivastava. Scarlet: explainable attention based graph neural network for fake news spreader prediction. In Advances in Knowledge Discovery and Data Mining: 25th Pacific-Asia Conference, PAKDD 2021, Virtual Event, May 11–14, 2021, Proceedings, Part I, pages 714–727. Springer, 2021.

[80] Cynthia Rudin. Please stop explaining black box models for high stakes decisions. stat, 1050:26, 2018.

[81] Michael Sejr Schlichtkrull, Nicola De Cao, and Ivan Titov. Interpreting graph neural networks for nlp with differentiable edge masking. International Conference on Learning Representations, 2021.

[82] Rainer Schmidt, Stefania Montani, Riccardo Bellazzi, Luigi Portinale, and Lothar Gierl. Cased-based reasoning for medical knowledge-based systems. International Journal of Medical Informatics, 64(2-3):355–367, 2001.

[83] Thomas Schnake, Oliver Eberle, Jonas Lederer, Shinichi Nakajima, Kristof T Schütt, Klaus-Robert Müller, and Grégoire Montavon. Higher-order explanations of graph neural networks via relevant walks. IEEE transactions on pattern analysis and machine intelligence, 44(11):7581–7596, 2021.

[84] Caihua Shan, Yifei Shen, Yao Zhang, Xiang Li, and Dongsheng Li. Reinforcement learning enhanced explainer for graph neural networks. Advances in Neural Information Processing Systems, 34:22523–22533, 2021.

[85] Lloyd S Shapley et al. A value for n-person games. 1953.

[86] Ben Shneiderman. Bridging the gap between ethics and practice: guidelines for reliable, safe, and trustworthy human-centered ai systems. ACM Transactions on Interactive Intelligent Systems (TiiS), 10(4):1–31, 2020.

[87] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. nature, 550(7676):354–359, 2017.

[88] Murat Cihan Sorkun, Abhishek Khetan, and Süleyman Er. Aqsoldb, a curated reference set of aqueous solubility and 2d descriptors for a diverse set of compounds. Scientific data, 6(1):143, 2019.

[89] Yongduo Sui, Xiang Wang, Jiancan Wu, Min Lin, Xiangnan He, and Tat-Seng Chua. Causal attention for interpretable and generalizable graph classification. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pages 1696–1705, 2022.

[90] Mengying Sun, Sendong Zhao, Coryandar Gilvary, Olivier Elemento, Jiayu Zhou, and Fei Wang. Graph convolutional networks for computational drug development and discovery. Briefings in bioinformatics, 21(3):919–935, 2020.

[91] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In International conference on machine learning, pages 3319–3328. PMLR, 2017.

[92] Juntao Tan, Shijie Geng, Zuohui Fu, Yingqiang Ge, Shuyuan Xu, Yunqi Li, and Yongfeng Zhang. Learning and evaluating graph neural network explanations based on counterfactual and factual reasoning. In Proceedings of the ACM Web Conference 2022, pages 1018–1027, 2022.

[93] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In 2015 ieee information theory workshop (itw), pages 1–5. IEEE, 2015.

[94] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In International conference on learning representations, 2019.

[95] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. arXiv preprint arXiv:1710.10903, 2017.

[96] Minh Vu and My T Thai. Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. Advances in neural information processing systems, 33:12225–12235, 2020.

[97] Minh N Vu and My T Thai. On the limit of explaining black-box temporal graph neural networks. In AAAI, 2022.

[98] Nikil Wale, Ian A Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. Knowledge and Information Systems, 14:347–375, 2008.

[99] Xiang Wang, Yingxin Wu, An Zhang, Xiangnan He, and Tat-Seng Chua. Towards multi-grained explainability for graph neural networks. Advances in Neural Information Processing Systems, 34:18446–18458, 2021.

[100] Xiaoqi Wang and Han-Wei Shen. Gnninterpreter: A probabilistic generative model-level explanation for graph neural networks. arXiv preprint arXiv:2209.07924, 2022.

[101] Geemi P Wellawatte, Aditi Seshadri, and Andrew D White. Model agnostic generation of counterfactual explanations for molecules. Chemical science, 13(13):3697–3705, 2022.

[102] Bryan Wilder, Han-Ching Ou, Kayla de la Haye, and Milind Tambe. Optimizing network structure for preventative health. In AAMAS, pages 841–849, 2018.

[103] Bingzhe Wu, Jintang Li, Junchi Yu, Yatao Bian, Hengtong Zhang, CHaochao Chen, Chengbin Hou, Guoji Fu, Liang Chen, Tingyang Xu, et al. A survey of trustworthy graph learning: Reliability, explainability, and privacy protection. arXiv preprint arXiv:2205.10014, 2022.

[104] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In International conference on machine learning, pages 6861–6871. PMLR, 2019.

[105] Haoran Wu, Wei Chen, Shuang Xu, and Bo Xu. Counterfactual supporting facts extraction for explainable medical record based diagnosis with graph network. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 1942–1955, 2021.

[106] Ying-Xin Wu, Xiang Wang, An Zhang, Xiangnan He, and Tat-Seng Chua. Discovering invariant rationales for graph neural networks. International Conference on Learning Representations, 2022.

[107] 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.

[108] Wenwen Xia, Mincai Lai, Caihua Shan, Yao Zhang, Xinnan Dai, Xiang Li, and Dongsheng Li. Explaining temporal graph models through an explorer-navigator framework. In The Eleventh International Conference on Learning Representations, 2023.

[109] Jiaxuan Xie, Yezi Liu, and Yanning Shen. Explaining dynamic graph neural networks via relevance back-propagation. arXiv preprint arXiv:2207.11175, 2022.

[110] Jiacheng Xiong, Zhaoping Xiong, Kaixian Chen, Hualiang Jiang, and Mingyue Zheng. Graph neural networks for automated de novo drug design. Drug Discovery Today, 26(6):1382–1393, 2021.

[111] Hao Xu, Shengqi Sang, Herbert Yao, Alexandra I Herghelegiu, Haiping Lu, James T Yurkovich, and Laurence Yang. Aprile: Exploring the molecular mechanisms of drug side effects with explainable graph neural networks. bioRxiv, pages 2021–07, 2021.

[112] Han Xuanyuan, Pietro Barbiero, Dobrik Georgiev, Lucie Charlotte Magister, and Pietro Lió. Global concept-based interpretability for graph neural networks via neuron analysis. arXiv preprint arXiv:2208.10609, 2022.

[113] Qiang Yang, Changsheng Ma, Qiannan Zhang, Xin Gao, Chuxu Zhang, and Xiangliang Zhang. Interpretable research interest shift detection with temporal heterogeneous graphs. In Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining, pages 321–329, 2023.

[114] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. Advances in neural information processing systems, 32, 2019.

[115] Junchi Yu, Jie Cao, and Ran He. Improving subgraph recognition with variational graph information bottleneck. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 19396–19405, 2022.

[116] Junchi Yu, Tingyang Xu, and Ran He. Towards the explanation of graph neural networks in digital pathology with information flows. arXiv preprint arXiv:2112.09895, 2021.

[117] Junchi Yu, Tingyang Xu, Yu Rong, Yatao Bian, Junzhou Huang, and Ran He. Graph information bottleneck for subgraph recognition. International Conference on Learning Representations, 2021.

[118] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. Xgnn: Towards model-level explanations of graph neural networks. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 430–438, 2020.

[119] Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. Explainability in graph neural networks: A taxonomic survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2022.

[120] Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. Explainability in graph neural networks: A taxonomic survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2022.

[121] Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. On explainability of graph neural networks via subgraph explorations. In International Conference on Machine Learning, pages 12241–12252. PMLR, 2021.

[122] Shichang Zhang, Yozen Liu, Neil Shah, and Yizhou Sun. Gstarx: Explaining graph neural networks with structure-aware cooperative games. In Advances in Neural Information Processing Systems, 2022.

[123] Yue Zhang, David Defazio, and Arti Ramesh. Relex: A model-agnostic relational model explainer. In Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society, pages 1042–1049, 2021.

[124] Zaixi Zhang, Qi Liu, Hao Wang, Chengqiang Lu, and Cheekong Lee. Protgnn: Towards self-explaining graph neural networks. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 36, pages 9127–9135, 2022.

[125] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. IEEE transactions on intelligent transportation systems, 21(9):3848–3858, 2019.

[126] Houliang Zhou, Lifang He, Yu Zhang, Li Shen, and Brian Chen. Interpretable graph convolutional network of multi-modality brain imaging for alzheimer's disease diagnosis. In 2022 IEEE 19th International Symposium on Biomedical Imaging (ISBI), pages 1–5. IEEE, 2022.

[127] Xiaolin Zhu, Yong Zhang, Zhao Zhang, Da Guo, Qi Li, and Zhao Li. Interpretability evaluation of botnet detection model based on graph neural network. In IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pages 1–6. IEEE, 2022.

[128] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. Bioinformatics, 34(13):i457–i466, 2018.

[129] Marinka Zitnik, Francis Nguyen, Bo Wang, Jure Leskovec, Anna Goldenberg, and Michael M Hoffman. Machine learning for integrating data in biology and medicine: Principles, practice, and opportunities. Information Fusion, 50:71–91, 2019.

Generative Explanation for Graph Neural Network: Methods and Evaluation

Jialin Chen[1], Kenza Amara[2], Junchi Yu[3], Rex Ying[1],
[1]Department of Computer Science, Yale University
[2]Department of Computer Science, ETH Zurich
[3]Institute of Automation, Chinese Academy of Sciences
[1]{jialin.chen,rex.ying}@yale.edu,[2]kenza.amara@ai.ethz.ch,[3]yujunchi2019@ia.ac.cn

Abstract

Graph Neural Networks (GNNs) achieve state-of-the-art performance in various graph-related tasks. However the black-box nature often limits their interpretability and trustworthiness. Numerous explanation methods have been proposed to uncover the decision-making logic of GNNs, by generating underlying explanatory substructures. In this paper, we conduct a comprehensive review of the existing explanation methods for GNNs from the perspective of graph generation. Specifically, we propose a unified optimization objective for current generative explanation methods, comprising two sub-objectives: Attribution and Information constraints. We further demonstrate their specific manifestations in different generative model architectures and explanation scenarios. With the unified objective of the explanation problem, we reveal the shared characteristics and distinctions among current methods, laying the foundation for future methodological advancements. Empirical results demonstrate the advantages and limitations of different approaches in terms of explanation performance, efficiency, and generalizability.

1 Introduction

Graph Neural Networks (GNNs) have emerged as a powerful tool for studying graph-structured data in various applications, such as social networks, drug discovery, and recommendation systems [55, 10, 40, 11, 9, 46, 13]. The explainability and trustworthiness of GNNs are crucial for their successful deployment in real-world scenarios, especially in high-stake applications, such as anti-money laundering, fraud detection, and healthcare forecasting [51, 31, 1]. Explanations for GNNs aim to discover the reasoning logic behind their predictions, making them more understandable and transparent to users. Explanations also help identify potential biases and build trust in the decision-making process of the model. Furthermore, they aid users in understanding complex graph-structured data, leading to improved outcomes in various applications through better feature extraction [54, 12, 47].

Numerous explanation methods have been extensively studied for GNNs, including gradient-based attribution methods [32, 6, 36], perturbation-based methods [48, 41, 52, 34, 16], *etc.* However, most of these methods optimize individual explanations for a specific instance, lacking global attention to the overall dataset and the ability to generalize to unseen instances. To tackle this challenge, generative explainability methods have emerged recently, which instead formulate the explanation task as a distribution learning problem. Generative explainability methods aim to learn the underlying distributions of the explanatory graphs across the entire graph dataset [42, 22, 28, 50], providing a more holistic approach to GNN explanations.

Current surveys in the field of Graph Neural Networks (GNNs) explainability primarily focus on the taxonomy and evaluation of explanation methods [51, 1, 33], as well as broader trustworthy aspects such as robustness, privacy, and fairness [44, 54, 23, 47]. The emerging generative explainability methods prompt us to consider

the potential advantages of incorporating distribution learning into the optimization objective, such as better explanation efficiency and generalizability.

Our work stands apart from previous works by thoroughly investigating the different mechanisms for generating explanations. We explore a comprehensive range of graph generation techniques that have been employed in GNN explanation tasks, including cutting-edge techniques such as the Variational Graph Autoencoder (VGAE) and denoising diffusion models. Our study begins by elucidating the core design considerations of different generative models and employs a novel generative perspective to unify a group of effective GNN explanation approaches. The key insight lies in a unified optimization objective, which includes an *Attribution* constraint and an *Information* constraint to ensure that the generated explanations are sufficiently succinct and relevant to the predictions. We subsequently delve into the details of the practical designs of the *Attribution* and *Information* constraints to facilitate the analysis of the connections and potential extensions of current generative explainability methods. The proposed unified optimization objective also empowers GNN users to efficiently design effective generative explainability methods.

Comprehensive experiments on synthetic and real-world datasets demonstrate the advantages and drawbacks of these existing methods. Specifically, our results show that generative approaches are empirically more efficient during the inference stage. Meanwhile, generative approaches achieve the best generalization capacity compared to other non-generative approaches.

This paper is structured as follows. First, we introduce the notations and problem setting in Sec. 2. Then, we propose a standard optimization objective with *Attribution* constraint and *Information* constraint to unify generative explanation methods in Sec. 3.1. Detailed expressions of these two constraints are elaborated in Sec. 3.2 and Sec. 3.3, respectively. In Sec. 3.4, we discuss how to generalize the proposed framework to extensive explanation scenarios, e.g.counterfactual and model-level explanations. Additionally, we present a taxonomy of representative works with different generative backbones in Sec. 4. Finally, we conduct comprehensive evaluations and demonstrate the potential of deep generative methods for GNN explanation in Section 5.

2 Preliminaries

2.1 Notations and Definitions

Given a well-trained GNN model f (base model) and an instance (i.e.a node or a graph) of the dataset, the objective of the explanation task is to identify concise graph substructures that contribute the most to the model's predictions. The given graph (or N-hop neighboring subgraph of the given node) can be represented as a quadruplet $G(\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{E})$, where $\mathcal{V}$ is the node set, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the edge set. $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d_n}$ and $\mathbf{V} \in \mathbb{R}^{|\mathcal{E}| \times d_e}$ denote the feature matrices for nodes and edges, respectively, where d_n and d_e are the dimensions of node features and edge features. In this work, we focus on structural explanation, i.e.we keep the dimensions of node and edge features unchanged. The notations used throughout this paper are summarized in Table 1. Depending on the specific explanation scenario, we define the explanation graphs with different target labels as follows.

Definition 2.1 (Explanation Graph) *Given a well-trained GNN f and an instance represented as $G(\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{V})$, an explanation graph $G_e(\mathcal{V}_e, \mathcal{E}_e, \mathbf{X_e}, \mathbf{E_e})$ for the instance is a compact subgraph of G, such that $\mathcal{V}_e \subseteq \mathcal{V}$, $\mathcal{E}_e \subseteq \mathcal{E}$, $\mathbf{X_e} = \{X_j | v_j \in \mathcal{V}_e\}$ and $\mathbf{E_e} = \{E_k | e_k \in \mathcal{E}_e\}$, where v_j and X_j denote the graph node and the corresponding node feature, e_k and E_k denote the graph edge and the corresponding edge feature. Explanation graph G_e is expected to be compact and result in the same predicted label Y^* as the label of G made by f, i.e.$Y^* = Y_f(G_e) = Y_f(G)$, where $Y_f(\cdot)$ denotes the predicted label made by the model f.*

Definition 2.2 (Counterfactual Explanation Graph) *Given a well-trained GNN f and an instance G, a counterfactual explanation graph G_{ce} is as close as possible to the original graph G, while it results in a different predicted label Y^* with the label of G predicted by f, i.e.$Y^* = Y_f(G_{ce}) \neq Y_f(G)$.*

Notation	Description
$G(\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{E})$	a graph with nodes $\mathcal{V}$, edges $\mathcal{E}$, node features $\mathbf{X}$ and edge features $\mathbf{E}$
$v_j \in \mathcal{V}$	a graph node
$e_k \in \mathcal{E}$	a graph edge
G_e	an explanation graph
G_{ce}	a counterfactual explanation graph
G_m^c	a model-level explanation graph for the target class c
$\mathcal{G} = \{G^1, \cdots, G^M\}$	the graph set of M input instances
$\mathcal{G}_e = \{G_e^1, \cdots, G_e^M\}$	the set of M generated explanation graphs
$\mathbf{X} = \{X_1, \cdots, X_{\lvert\mathcal{V}\rvert}\} \in \mathbb{R}^{\lvert\mathcal{V}\rvert \times d_n}$	the node feature matrix with d_n feature dimensions
$\mathbf{E} = \{E_1, \cdots, E_{\lvert\mathcal{E}\rvert}\} \in \mathbb{R}^{\lvert\mathcal{E}\rvert \times d_e}$	the edge feature matrix with d_e feature dimensions
$A \in \{0,1\}^{\lvert\mathcal{V}\rvert \times \lvert\mathcal{V}\rvert}$	the unweighted adjacency matrix
$\mathcal{Y}$	the label space
f	the well-trained GNN to be explained (base model)
$Y^s \in \{0, 1, \cdots, \lvert\mathcal{Y}\rvert\}$	the original label of s, where s can be a node or a graph
$Y_f(s) \in \{0, 1, \cdots, \lvert\mathcal{Y}\rvert\}$	the predicted label of s by f
$Y^* \in \{0, 1, \cdots, \lvert\mathcal{Y}\rvert\}$	the predicted label during the explanation stage
$P_f(s) \in [0,1]^{\lvert\mathcal{Y}\rvert}$	the output probability vector of s by f
$f(s) \in \mathbb{R}^{\lvert\mathcal{Y}\rvert}$	the output logit vector of s by f
$g_\theta(\cdot) : \mathcal{G} \to \mathcal{G}_e$	an explanation generator with parameters θ
$p(\cdot \mid G)$	the distribution of the explanation graphs for a given G

Table 1: Summary of the notations

Definition 2.3 (Model-level Explanation Graph) *Given a set of graph $\mathcal{G} = \{G^1, \cdots, G^M\}$, where each $G^j \in \mathcal{G}$ has the same label c predicted by the well-trained GNN f, a model-level explanation graph G_m^c is a distinctive subgraph pattern that commonly appears in $\mathcal{G}$, and is predicted as the same label c, i.e. $Y^* = Y_f(G_m^c) = c$.*

2.2 General Explanation for Graph Neural Network

Given a graph G and the corresponding label Y^* in specific explanation scenarios, generating explanation graphs can be formulated as an optimization problem that maximizes the mutual information between the generated graph and the target label Y^* with the following objective:

$$\begin{aligned} G_e^* &= \underset{G_e}{\operatorname{argmax}}\, MI(Y^*, G_e) = \underset{G_e}{\operatorname{argmax}}\, H(Y^*) - H(Y^*|G_e) \\ &= \underset{G_e}{\operatorname{argmin}}\, H(Y^*|G_e) = \underset{G_e}{\operatorname{argmin}} -\mathbb{E}_{Y^*|G_e} \log P(Y^*|G_e), \end{aligned} \tag{8}$$

where $MI(\cdot,\cdot)$ denotes the mutual information function, $H(\cdot)$ denotes the entropy function, $P(Y^*|G_e)$ measures the probability that G_e is predicted as the label Y^*.

Instance-dependent Explainers. Early efforts develop explanation frameworks for GNNs that optimize an explanation for each individual instance. For example, the Gradient-based methods [6, 32] evaluate the node and edge importance with the gradient norm of prediction node and edge features. Nodes and edges with higher gradient norms are considered more important and are included in the explanation subgraph of the final prediction. Other methods utilize more advanced frameworks such as mask optimization [48], surrogate model [39], and Monte Carlo Tree Search [52] to search the explanation subgraphs for each individual instance.

Although instance-dependent explainers partly reveal the behavior of GNNs, there are several limitations. Since these methods optimize explanations for individual graphs, they require significant computation and lack holistic knowledge about how the GNN model behaves across the entire dataset. Furthermore, the learning modules in instance-dependent explainers cannot be generalized to explain the predictions for unseen instances, since the parameters are specific for individual instances.

3 Generative Framework for Graph Explanations

3.1 Unified Optimization Objective

To overcome the aforementioned limitations, recent research has develop approaches that leverage deep generative methods to explain GNNs. Instead of optimizing an explanation for individual instances, the generative methods aim to generate explanations for new graphs by learning a strategy to search for the most explanatory subgraphs across the whole dataset. Formally, given a set of input graphs $\mathcal{G}$, the generative explainer learns the distribution of the underlying explanation graphs $p(G_e|G)$ using a parameterized subgraph generator $g_\theta : \mathcal{G} \to \mathcal{G}_e$. After training, the subgraph generator is capable of identifying the explanation subgraphs that are most important to the desired graph labels:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \log P_{Y^*}(G_e|\theta, G), \tag{9}$$

where $P_{Y^*}(G_e|\theta, G)$ is the probability that the generated $G_e = g_\theta(G)$ is a valid explanation for the desired label Y^*. In addition to the validity requirement, an ideal explanation graph should be sparse and compact compared with the given graph. Directly optimizing Eq. 9 leads to a trivial solution where $G_e = G$, as the input graph is most informative for the graph label. To obtain a compact explanation, we impose an information constraint $\mathcal{L}_{\text{INFO}}(G_e, G)$ that restricts the amount of information contained in the generated explanation subgraph, thereby ensuring the conciseness and brevity of the explanations. The overall objective of generative explanation is

$$\min_\theta - \log P_{Y^*}(G_e|\theta, G) + \mathcal{L}_{\text{INFO}}(G_e, G) := \mathcal{L}_{\text{ATTR}}(G_e, Y^*) + \mathcal{L}_{\text{INFO}}(G_e, G). \tag{10}$$

We name the first term in Eq. 10 the attribution loss $\mathcal{L}_{\text{ATTR}}(G_e, Y^*)$, which measures whether G_e captures the most important substructures for the desired label Y^*. $\mathcal{L}_{\text{ATTR}}$ is typically the cross-entropy loss for categorical Y^* and mean squared loss for continuous Y^*.

Connection With Variational Auto-encoder. $\mathcal{L}_{\text{INFO}}$ in Eq. 10 can be set as the variational constraint, i.e.$\mathcal{L}_{\text{INFO}}(G, G_e) := \mathrm{D}_{\text{KL}}\left(q_\theta(G_e|G) \| Q(G_e)\right)$, where D_{KL} denotes Kullback–Leibler divergence, $Q(G_e)$ is the prior distribution of the generated explanation graph G_e, and $q_\theta(G_e|G)$ is the variational approximation to $g_\theta(G_e|G)$, variational constraint drives the posterior distribution of G_e generated by $g_\theta(\cdot|G)$ to its prior distribution, thus restricting the information contained in G_e in the process. The overall objective is

$$\mathcal{L} = \mathbb{E}_G - \log P_{Y^*}(G_e|\theta, G) + \mathrm{D}_{\text{KL}}\left(q_\theta(G_e|G) \| Q(G_e)\right), \tag{11}$$

In this case, the objective of generative explanation shares similar spirits with the Variational Auto-encoder (VAE) [20]. Recall the optimization objective of VAE is

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_{z\sim q_\phi(z|G)} - \log\left(p_\varphi(G|z)\right) + \mathrm{D}_{\text{KL}}\left(q_\phi(z|G) \| q(z)\right), \tag{12}$$

where q_ϕ is the encoder that maps graph G into a latent space, then the decoder p_φ recovers the original graph G based on the latent representation z. $q(z)$ is the prior distribution of the latent representation, which is usually a Gaussian distribution. Notably, the generative explanation approach with the variational constraint as $\mathcal{L}_{\text{INFO}}$ (Eq. 11) is a variant of Variational Auto-encoder (VAE) (Eq. 12), albeit with two fundamental distinctions. Firstly, VAE aims to learn the distribution of the original graph, whereas generative explanation focuses on

learning the underlying distribution of explanatory structures. Secondly, VAE constrains the distribution of latent representation z, while the generative explanation constrains the posterior distribution of G_e. These distinctions highlight the methodologies for generalizing generative models to the task of GNN explainability.

3.2 Taxonomies of Generative Models

In this section, we will discuss several taxonomies of generative models that have been employed in the field of GNN explainability. These models aim to learn the probability distribution of the underlying explanatory substructures by training across the entire graph dataset.

Mask Generation (MG) [27, 49, 29, 43] The mask generation method is to optimize a mask generator g_θ to generate the edge mask M for the input graph G. The elements of the mask represent the importance score of the corresponding edges, which is further employed to select the important substructures of the input graph as explanations. The mask generator is usually a graph encoder followed by a multi-layer perception (MLP), which first embeds edge representations h_{e_i} for each edge and then generates the sampling probability p_i for edge e_i. The mask $m_i \in \{0, 1\}$ of e_i is sampled from the Bernoulli distribution $\mathrm{Bern}(p_i)$. Since the sampling process is non-differentiable, the Gumbel-Softmax trick is usually employed for continuous relaxation as follows:

$$m_i = \sigma((\log \epsilon - \log(1-\epsilon) + \log(p_i) - \log(1-p_i))/\tau), \epsilon \sim \mathrm{Uniform}(0, 1) \tag{13}$$

where τ is the temperature, σ is the sigmoid function. When τ goes to zero, m_i is close to the discrete Bernoulli distribution. The explanation G_e is obtained by applying the edge mask M to the input graph G, i.e. $G_e = M \odot G = g_\theta(G) \odot G$, where $\odot$ is element-wise multiplication. Given an input graph G and the desired label Y^*, the parameter θ of the mask generator g_θ is optimized by minimizing the following attribution loss:

$$\mathcal{L}_{\text{ATTR}} = -\mathbb{E}_G \log P_{Y^*}(G_e|G, \theta) \;\text{ with }\; G_e = g_\theta(G) \odot G, \tag{14}$$

which is equivalent to the cross entropy between the output probability $P_f(G_e)$ made by the base GNN f and the desired label Y^*.

Variational Graph Autoencoder (VGAE) [25, 28, 26] Variational Graph Autoencoder (VGAE) [20] is a variational autoencoder for graph-structured data, where the encoder $q_\phi(\cdot)$ and the decoder $p_\theta(\cdot)$ are typically parameterized by graph neural networks. VGAE can be used to learn the distribution of the underlying explanations and thus generate explanation graphs for unseen instances. The encoder maps an input graph G to a probability distribution over a latent space. The decoder then samples from the latent space and recovers an explanation graph by $G_e = p_\theta(z)$. The attribution loss of VGAE for generating explanation graphs is

$$\mathcal{L}_{\text{ATTR}} = \mathbb{E}_{z\sim q_\phi(z|G)} - \log P_{Y^*}(G_e|\theta, z, G) + \mathrm{D}_{\mathrm{KL}}(q_\phi(z|G)\|q(z)). \tag{15}$$

The standard VGAE shown in Eq. 12 aims to generate realistic graphs. On the contrary, the VGAE-based explainer maximizes the likelihood of the valid explanation graph G_e for the desired label Y^*. The former term in Eq. 15 evaluates whether the explanation graph G_e captures the most important structures for Y^*. It can be replaced by the cross entropy between the output probabilities $P_f(G_e)$ and Y^* as CLEAR [28], or the cross entropy between the generated graph G_e and ground-truth explanations as GEM [25]. The second term of KL divergence is a model-specific constraint that drives the posterior distribution $q_\phi(z|G)$ to the prior distribution $q(z)$, which is usually a Gaussian distribution.

Generative Adversarial Networks (GAN) [24] Generative Adversarial Network (GAN) is a type of generative model that does not include an explicit encoder component. Instead, GANs consist of a generator g_θ that creates an explanation graph $G_e = g_\theta(z)$ for z sampled from a prior distribution $q(z)$, and a discriminator d_ϕ that distinguishes between the input graph G and the generated explanation graph G_e. The objective function of a GAN is a min-max game in which the generator g_θ tries to minimize the function while the discriminator d_ϕ tries to maximize it.

$$\mathcal{L}_{\text{ATTR}} = -\mathbb{E}_{z\sim q(z)} \log P_{Y^*}(G_e|\theta, z, G) + \log d_\phi(G) + \mathbb{E}_{z\sim q(z)} \log(1 - d_\phi(g_\theta(z))), \tag{16}$$

where the first term can be the cross entropy between $P_f(g_\theta(z))$ and the desired label Y^*. $d_\theta(\cdot)$ denotes the probability that the discriminator predicts that the input is an explanation. GAN-based Explainer [24] was recently proposed with the first term replaced with the square of the difference between the output logit of f for G and G_e, i.e.$\mathbb{E}_{z\sim q(z)} \left(f(G) - f(g_\theta(z))\right)^2$. Once the GAN is well-trained, the generator g_θ can be employed to generate valid explanation graphs for any unseen instances, given a point in the prior distribution $q(z)$.

Diffusion [17, 4, 38] The Diffusion model is a class of generative models that have been used in graph generation tasks to generate realistic graphs, which contains two key components: the forward diffusion process, and the reverse denoising network. Given an original graph G_0, the forward diffusion process progressively generates a sequence of noisy graphs $\{G_0, G_1, \cdots, G_T\}$ with increasing levels of noise, and G_T becomes pure noise. Let $\boldsymbol{A}_t = [\boldsymbol{a}_t^{ij}]_{ij}$ denote the one-hot version of the adjacency matrix of G_t at timestep t, where $\boldsymbol{a}_t^{ij} \in \{0, 1\}^2$ is a 2-dimensional one-hot encoding of the ij-element. The discrete forward diffusion process is defined as $q(\boldsymbol{a}_t^{ij}|\boldsymbol{a}_{t-1}^{ij}) = \text{Cat}(\boldsymbol{a}_t^{ij}; \boldsymbol{P} = \boldsymbol{a}_{t-1}^{ij}\boldsymbol{Q}_t)$, where $\text{Cat}(\boldsymbol{x}, \boldsymbol{P})$ is a categorical distribution over the one-hot vector $\boldsymbol{x}$ with probability vector $\boldsymbol{P}$ and $\boldsymbol{Q}_t \in [0, 1]^{2\times 2}$ is a symmetric transition matrix at timestamp t. The forward diffusion is a Markov process that independently performs over all edges in the full adjacency matrix. Therefore, the graph-level diffusion process is $q(G_t|G_{t-1}) = \prod_{ij} q(\boldsymbol{a}_t^{ij}|\boldsymbol{a}_{t-1}^{ij})$ and $q(G_T|G_0) = \prod_{t=1}^{T} q(G_t|G_{t-1})$. The reverse denoising network g_θ learns to remove the noise and recover the target explanation graph by $G_e = g_\theta(G_t)$ for $t = 1, 2, \cdots, T$. Since an ideal G_e is a compact subgraph of the original graph G, it is equivalent to ensuring that the complementary subgraph $G - G_e$ is close to G. Therefore, instead of using the generated graph to reconstruct the original graph in the standard diffusion model, we make $G - G_e$ approximate G and take G_e as the output explanation graph for G. The loss function is as follows,

$$\mathcal{L} = -\mathbb{E}_{t\in[0,T]}\mathbb{E}_{G_t\sim q(G_t|G_0)} \log P_{Y^*}(G_e|\theta, G_t, G_0) + \mathbb{E}_{t\in[0,T]}\mathbb{E}_{G_t\sim q(G_t|G_0)}\mathcal{L}_{\text{CE}}(G_0 - g_\theta(G_t), G_0). \tag{17}$$

The second term denotes the binary cross entropy loss across all elements in the adjacency matrices of $(G_0 - g_\theta(G_t)$ and G_0. Notably, the diffusion-based explainer naturally involves the *Information* constraint into the optimization objective, as $\mathcal{L}_{\text{CE}}$ plays a role of $\mathcal{L}_{\text{INFO}}$ that restricts the size of generated explanation graph G_e.

Reinforcement Learning Approaches Reinforcement Learning (RL) can be used to learn the distribution of underlying explanation graphs by framing the process of generating an explanation graph as a trajectory of step-wise states. Let $\tau = (s_0, \cdots, s_K) \in \mathcal{T}$ denote a trajectory τ that consists of states $s_0, \cdots, s_K$ and $\mathcal{T}$ is a set of all possible trajectories. At the k-th step, the state s_k refers to a subgraph of the given graph, denoted as G_k. G_0 is a starting node from the given graph and G_K is the terminal explanation graph. Let a_k denote the action from s_{k-1} to s_k, which is usually adding a neighboring edge to the current subgraph G_{k-1}. Instead of learning the distribution of the holistic explanation graphs G_e, reinforcement learning approaches learn the distribution of the state transition, i.e.the distribution of the selected edge to be added given the current state. The objective of these approaches is to learn a generative agent (policy network) $g_\theta(G_{k-1})$ with parameters θ that determines the next action by $a_k \sim g_\theta(G_{k-1})$. The reward function is a crucial component of reinforcement learning to address the non-differentiability issue of the sampling process within the generative agent. In the explanation task, the

reward function measures the quality of the subgraph G_k for the desired label Y^* given the current subgraph G_{k-1}. RCExplainer [42] proposes to take the individual causal effect (ICE) [14] of the action a_k as the reward. GFlowExplainer [22] involves the output probability $P_f(G_k)$ over the desired label Y^* into the reward design. Reinforcement learning is used in conjunction with another probabilistic model to represent the distribution over the states, e.g.Markov Decision Process (MDP), Direct Acyclic Graph (DAG), etc.

- **Markov Decision Process (MDP) [50, 42].** The trajectories of states can be framed as a Markov Decision Process. The generative agent $g_\theta(G_{k-1})$ captures the sequential effect of each edge in the generating process toward a target explanation graph. The attribution loss function is as follows,

$$\mathcal{L}_{\text{ATTR}} = -\mathbb{E}_k[R(G_{k-1}, a_k)] \log P(a_k|\theta, G_{k-1}), \tag{18}$$

  where $R(G_{k-1}, a_k)$ is the reward for the action a_k at state G_{k-1} and $P(a_k|\theta, G_{k-1})$ is the probability of yielding a_k from the distribution $g_\theta(G_{k-1})$. This loss function encourages the generative agent to attach higher probabilities to the edges that bring larger rewards, thus leading to an ideal explanation.

- **Direct Acyclic Graph (DAG) [22].** GflowNet [8] frames the trajectories of the states as a direct acyclic graph and aims to train a generative policy network where the distribution over the states is proportional to a pre-defined reward function. A concept of *flow* is introduced to measure the probability flow along the trajectories. Let $F(\tau)$ denote the flow of the trajectory τ and $F(s)$ denote the flow of the state s, which is the sum of all trajectory flows passing through that state. It satisfies that the inflows of a state s_k equals the outflows of s_k. The attribution loss is as follows,

$$\mathcal{L}_{\text{ATTR}}(\tau) = \sum_{s_{k+1} \in \tau} \left(\sum_{(s_k, a_k) \to s_{k+1}} F(s_k, a_k) - \mathbb{1}_{s_{k+1}=s_K} R(s_K) - \mathbb{1}_{s_{k+1} \neq s_K} \sum_{a_{k+1}} F(s_{k+1}, a_{k+1}) \right)^2, \tag{19}$$

  where $\sum_{(s_k,a_k)\to s_{k+1}} F(s_k, a_k)$ and $\sum_{a_{k+1}} F(s_{k+1}, a_{k+1})$ denote the inflows and outflows of a state s_{k+1}, respectively. $\mathbb{1}$ is used to check whether s_{k+1} is the terminal state s_K. $R(s_K)$ is the reward of the graph G_K corresponding to the terminal state s_K. It is provable that the distribution of the terminal states generated by the agent $P(s_K|\theta)$ trained with Eq. 19 is proportional to their rewards.

Typically, reinforcement learning approaches do not rely on an explicit $\mathcal{L}_{\text{INFO}}$ to constrain the sparsity of the generated explanation graph. One common strategy is that we create trajectories $\tau = (s_0, \cdots, s_K)$ by iteratively sampling $a_k \sim g_\theta(G_{k-1})$ and stop this process once the stopping criteria are attained, e.g.K achieves the pre-defined size of explanation graphs.

3.3 Taxonomies of Information Constraint

Only maximizing the likelihood of the explanatory subgraph with Eq. 9 typically leads to a trivial solution of the whole input graph, which is unsatisfactory. An ideal explanatory subgraph is supposed to have a small portion of the original graph information as well as be faithful for the prediction. Hence, existing methods [29, 49] introduce an additional information constraint $\mathcal{L}_{\text{INFO}}$ as a regularization term to restrict the information of the generated explanation apart from the attribution loss $\mathcal{L}_{\text{ATTR}}$. The information constraint $\mathcal{L}_{\text{INFO}}$ can be categorized as Size Constraint, Mutual Information Constraint, and Variational Constraint.

Size Constraints [48] The size constraint is a straightforward approach to restricting subgraph information. Given an input graph G and the size tolerance $K \in (0, |G|)$, the size constraint is $|G_e| \leq K$. Here, $|\cdot|$ denotes the volume of a graph, and K is an integer hyperparameter to constrain the volume of explanatory subgraphs. This constraint is first introduced in GNNExplainer [48]. Since applying the same size constraint to different

graph sizes is problematic, some work employs the sparsity constraint $k \in (0, 1)$ and constrains the volume of explanatory subgraph with $|G_e| \leq k \cdot |G|$. Recent works [28, 26] further utilize a soft version of size constraint, i.e.$\mathcal{L}_{\text{INFO}} = d(G, G_e)$, where $d(G, G_e)$ is the element-wise distance between the adjacency matrices of G and G_e.

Although the size constraint is intuitive, it has several limitations. Firstly, the topological size is insufficient in measuring the subgraph information as they ignore the information within node and edge features. Secondly, one has to choose different sparsity tolerance k to achieve the best explanation performance, which is difficult due to the trade-off between the sparsity and validity of the explanations.

Mutual Information Constraint [49] The mutual information constraint restricts the subgraph information by reducing the relevance between the original graphs and the explanatory subgraphs. Given the graph G and the explanatory subgraph G_e, the mutual information constraint is formulated as:

$$\mathcal{L}_{\text{INFO}} = MI(G, G_e) = \mathbb{E}_{p(G)}\mathbb{E}_{p(G_e|G)} \log \frac{p(G_e|G)}{p(G_e)}. \tag{20}$$

Here $MI(x, y)$ is the mutual information of two random variables. Minimizing Eq. 20 reduces the relevance between G and G_e. Thus, the explanation generator tends to leverage limited input graph information to generate the explanation subgraph. Compared with the size constraints, the mutual information constraint is more fundamental in information measurement and flexible to different graph sizes. However, mutual information is intractable to compute, making the constraint impractical to use. One solution is resorting to computationally expensive estimation techniques, such as the Donsker-Varadhan representation of mutual information [7, 49].

Variational Constraint [29] Since the mutual information constraint is intractable, the variational constraint is proposed by deriving a tractable variational upper bound of the mutual information constraint. One can plug a prior distribution $q(G_e)$ into $MI(G, G_e)$ as the variational approximation to $p(G_e|G)$:

$$\begin{aligned} MI(G, G_e) &= \mathbb{E}_{p(G)}\mathbb{E}_{p(G_e|G)} \log \frac{p(G_e|G)}{p(G_e)} = \mathbb{E}_{p(G)}\mathbb{E}_{p(G_e|G)} \log \frac{p(G_e|G)}{q(G_e)} - \mathrm{D}_{\mathrm{KL}}(q(G_e)\|p(G_e)) \\ &\leq \mathbb{E}_{p(G)}\mathbb{E}_{p(G_e|G)} \log \frac{p(G_e|G)}{q(G_e)} := \mathcal{L}_{\text{VC}}. \end{aligned} \tag{21}$$

Here, the inequality is due to the non-negative nature of the Kullback–Leibler (KL) divergence. The posterior distribution $p(G_e|G) = \prod_{i=1}^{N} p(e_i|\theta)$ is factorized into the multiplication of the marginal distributions of edge sampling $p(e_i|\theta)$, which is parameterized with the generative explanation network θ. The prior distribution $q(G_e) = \prod_{i=1}^{N} q(e_i)$ is factorized into the prior distributions of edge sampling $q(e_i)$. N is the total edge number. In practice, $q(e_i)$ is usually chosen as the Bernoulli distribution or Gaussian distribution. Thus, the variational constraint is an upper bound of the mutual information $MI(G, G_e)$ that can be simplified as

$$\mathcal{L}_{\text{INFO}} = \mathcal{L}_{\text{VC}} = \sum_{i=1}^{N} \mathrm{D}_{\mathrm{KL}}(p(e_i|\theta)\|q(e_i)). \tag{22}$$

3.4 Extension of Explanation Scenarios

Counterfactual Explanation Most explanation methods focus on discovering the prediction-relevant subgraph to explain GNNs based on Eq. 9. Although these methods can highlight the important substructures for the predictions, they cannot answer the *counterfactual* problem such as: "Will the removal of certain substructure lead to prediction change of GNNs?" Counterfactual explanations provide insightful information on how the model prediction would change if some event had occurred differently, which is crucial in some real-world

scenarios, e.g.drug design and molecular modification [30, 18, 45]. Given an input graph G, the goal of the generative counterfactual explanation is to train a subgraph generator g_θ to generate a minimal substructure of the input. If the substructure is removed, the prediction of GNNs will change the most. Formally, the objective for counterfactual explanations is:

$$\mathcal{L}_{\text{CF}} = -\log P_{Y^*}(G_{ce}|\theta, G) + \mathcal{L}_{\text{INFO}}(G, \overline{G_{ce}}) \tag{23}$$

Here, $\overline{G_{ce}}$ denotes the generated substructure, which is also the modification applied to G to obtain a counterfactual explanation $\overline{G_{ce}} = G - G_{ce}$. $\mathcal{L}_{\text{INFO}}$ is a regularization term that constrains the information amount contained in $\overline{G_{ce}}$ to be minimal compared with the input graph G.

Connection to Graph Adversarial Attack. The counterfactual explanation methods can capture the vulnerability of GNN's prediction since the counterfactual explanation subgraph leads to the prediction change. This problem setting is similar to graph adversarial attacks as they both aim to alter the prediction behavior of the pre-trained GNN by modifying testing graphs. Recall that graph adversarial attacks modify the node features or graph structures to decrease the average performance of a pre-trained GNN. However, the explanation methods change the prediction of each testing sample by instance-level subgraph deletion instead of decreasing the overall testing performance after a one-time graph modification [2].

Connection to Graph Out-of-distribution Generalization. Deep learning models have been found to rely on spurious patterns in the input to make predictions. These patterns are often unstable under distribution shifts, leading to a drop in performance when testing on out-of-distribution (OOD) data. To address this issue, counterfactual augmentation has been proposed as an effective method for improving the OOD generalization ability of deep learning models. This technique involves minimally modifying the training data to change their labels and training the model with both the original and counterfactually augmented data. For graphs, counterfactually augmented subgraphs can be generated by removing subgraphs to create the complementary subgraph, which is a natural form of counterfactual augmentation. However, this approach has received less attention in the context of graph neural networks, presenting an avenue for future research.

Model-level Explanation The goal of model-level explanation in the context of GNNs is to identify important graph patterns that contribute to the decision boundaries of the model. Unlike instance-level explanation, model-level explanation provides insights into the general behavior of the model across a range of input graphs with the same predicted label. A brute-force approach to finding these patterns is to mine the subgraphs that commonly appear in graphs with the same predicted label. However, this is computationally expensive due to the exponentially large search space. Recently, generative methods have been proposed to generate model-level explanations, such as reinforcement learning [50] and probabilistic generative models [43].

In these approaches, a generator function $g_\theta(\cdot)$ is used to generate the model-level explanation G_m for a given predicted label Y^* based on a set of graphs $\mathcal{G}$ via $G_m \sim g_\theta(\mathcal{G}, Y^*)$. The optimization objective for the generator function is to minimize the negative log-likelihood of the explanation given the graphs, while also ensuring that the generated explanation is a compact and recurrent substructure in the set of input graphs:

$$\mathcal{L} = -\log P_{Y^*}(G_m|\theta, \mathcal{G}) + \mathcal{L}_{\text{INFO}}(G_m, \mathcal{G}). \tag{24}$$

The first term in Eq. 24 measures whether G_m captures the most determinant graph patterns for the prediction of Y^*. The generator $g_\theta(\cdot)$ can be modeled by other applicable generative models discussed in Sec. 3.2. $\mathcal{L}_{\text{INFO}}(G_m, \mathcal{G})$ ensures that G_m is a compact substructure that commonly appears in $\mathcal{G}$.

4 Method Taxonomy

The information constraints $\mathcal{L}_{\text{INFO}}$ in Sec. 3.3 and the attribution constraints $\mathcal{L}_{\text{ATTR}}$ in Sec. 3.2 can be combined to construct an overall optimization objective for GNN explainability. We provide a comprehensive comparison

Table 2: A comprehensive summary of existing generative explanation methods for Graph Neural Networks. RL-MDP denotes the reinforcement learning approach based on Markov Decision Process and RL-DAG denotes the reinforcement learning approach based on Direct Acyclic Graph.

Method	Generator	Information Constraint	Level	Scenario	Output
PGExplainer [27]	Mask Generation	size	instance	factual	E
GIB [49]	Mask Generation	mutual information	instance	factual	N
GSAT [29]	Mask Generation	variational	instance	factual	E
GNNInterpreter [43]	Mask Generation	size	model	factual	N / E / NF
GEM [25]	VGAE	size	instance	factual	E
CLEAR [28]	VGAE	size	instance	counterfactual	E / NF
OrphicX [26]	VGAE	variational & size	instance	factual	E
D4Explainer	Diffusion	size	instance & model	counterfactual	E
GANExplainer [24]	GAN	-	instance	factual	E
RCExplainer [42]	RL-MDP	size	instance	factual	SUBGRAPH
XGNN [50]	RL-MDP	size	model	factual	SUBGRAPH
GFlowExplainer [22]	RL-DAG	size	instance	factual	SUBGRAPH

and summary of existing generative explanation methods and their corresponding generators and information constraints in Table 2. Most existing approaches focus on instance-level factual explanations, while CLEAR [28] focuses on counterfactual explanation and D4Explainer is applicable for both counterfactual and model-level explanations. GIB [49] proposes to deploy mutual information between the generated explanation graph and the original graph as the information constraint, while GSAT [29] utilizes the variational constraint. We further compare the outputs of these approaches (the last column in Table 2), where E denotes outputting edge importance with continuous values, N denotes node importance with continuous values, NF denotes the importance of node features and SUBGRAPH denotes hard masks for discrete explanatory subgraphs.

5 Evaluation

5.1 Experimental setting

Datasets We evaluate the explainability methods on both synthetic and real-world datasets in different domains, including MUTAG, BBBP, MNIST, BA-2Motifs and BA-MultiShapes. **BA-2Motifs** [27] is a synthetic dataset with binary graph labels. The house motif and the cycle motif give class labels and thus are regarded as ground-truth explanations for the two classes. **BA-MultiShapes** [5] is a more complicated synthetic dataset with multiple motifs. Class 0 indicates that the instance is a plain BA graph or a BA graph with a house, a grid, a wheel, or the three motifs together. On the contrary, Class 1 denotes BA graphs with two of these three motifs. **MUTAG** is a collection of $\sim$3000 nitroaromatic compounds and it includes binary labels on their mutagenicity on Salmonella typhimurium. The chemical fragments -NO2 and -NH2 in mutagen graphs are labeled as ground-truth explanations [27]. The Blood–brain barrier penetration **BBBP** dataset includes binary labels for over 2000 compounds on their permeability properties. In molecular datasets, node features encode the atom type and edge features encode the type of bonds that connect atoms. **MNIST75sp** contains graphs that are converted from images in MNIST [21] using superpixels. In these graphs, the nodes represent the superpixels, and the edges are determined by the spatial proximity between the superpixels. The coordinates and intensity of the corresponding superpixel construct the node features. Dataset statistics are summarized in Table 3.

GNN models For each dataset, we first train a GNN model. We have tested four GNN models: GCN [19], GIN[15], GAT [37], and GraphTransformer [35]. We only display results for the GraphTransformer model for the real-world datasets and the GIN model for the synthetic datasets since they give the highest accuracy scores

	MUTAG	BBBP	MNIST75sp	BA-2Motifs	BA-MultiShapes
# graphs	2,951	2,039	70,000	1,000	1,000
# node features	14	9	5	1	10
# edge features	1	3	1	1	1
Avg # nodes	30	24	67	25	40
Avg # edges	61	52	541	51	87
Avg degree	2.0	2.1	7.9	2.0	2.2
# classes	2	2	10	2	2
GNN performance	0.94	0.92	0.96	1.00	0.71

Table 3: Dataset statistics and accuracy performance of the GNN model on the test set

on the test sets respectively. GraphTransformer and GIN give high accuracy on the real-world and synthetic datasets respectively, with a reasonable training time and fast convergence. Unlike GCN, GraphTransformer and GIN have also the advantage of taking edge features, extending their use to more complex graph datasets. The network structure of the GNN model for graph classification is a series of 3 layers with ReLU activation, followed by a max pooling layer to get graph representations before the final fully connected layer. We adopt the Adam optimizer with an initial learning rate of 0.001. We split train/validation/test with 80/10/10% for all datasets. Each model is trained for 200 epochs with an early stop. The accuracy performances of GNN models are shown in Table 3. The results show that the designed GNN models are sufficiently powerful for graph classifications on both synthetic and real-life datasets.

Explainability methods We compare non-generative methods: Saliency [6], Integrated Gradient [36], Occlusion [53], Grad-CAM [32], GNNExplainer [48], PGMExplainer [39], and SubgraphX [52], with generative ones: PGExplainer [27], GSAT [29], GraphCFE (CLEAR) [28], D4Explainer and RCExplainer [42]. Following GraphFramEx [3], we define an explanation as an edge mask on the existing edges in the initial graph to be explained. First, this constraint facilitates the comparison of very diverse explainability methods. Moreover, in the context of our study, all datasets are expected to be explained by some entities that already exist in the initial graphs, i.e.motifs in synthetic datasets and groups of atoms in molecular datasets. We follow the original setting to train PGExplainer, GSAT, and RCExplainer. We implement the diffusion-based explainer as introduced in Sec. 3.2, and name it D4Explainer. D4Explainer generates an explanatory graph that can contain additional edges that are not in the initial graph. To keep consistent, we retrieve the common edges with the initial graph to evaluate D4Explainer in this work. GraphCFE is a simplified version of CLEAR [28] without the causality component, which is an explainability method for counterfactual explanations. Indeed, the causal models introduced in [28] are constructed from simulations because it is hard to get the ground-truth causal model from datasets. Since we ignore the existence of any causal model in our datasets, we decide not to focus on the causality and use only the CLEAR-VAE backbone, i.e.GraphCFE, in this work. We retrieve the important edges by subtracting the counterfactual explanation generated by GraphCFE from the initial graph. The remaining edges have weights of 1, while the rest have weights of 0.

Metrics To evaluate the explainability methods, we use the systematic evaluation framework GraphFramEx [3]. We evaluate the methods on the faithfulness measure $fidelity{-}_{acc}$, which is defined as

$$fidelity{-}_{acc} = \frac{1}{N}\sum_{i=1}^{N}\left|\mathbb{1}(\hat{Y}_f(G^i) = Y^i) - \mathbb{1}(\hat{Y}_f(G_e^i) = Y^i)\right|,$$

where G^i and G_e^i denote the initial graph and the explanatory graph, respectively. $fidelity{-}_{acc}$ measures if the generated explanatory subgraph is faithful to the initial graph, i.e.leads to the same GNN prediction.

5.2 Instance-level explanations

Faithfulness We conducted a comprehensive comparison of the faithfulness between generative and non-generative methods using three real-world datasets (BBBP, MUTAG, and MNIST) and two synthetic datasets (BA-2Motifs and BA-MultiShapes). The results, depicted in Figure 1, indicate that generative methods are generally performing the same or better than non-generative methods. Specifically, for MNIST, generative methods outperform non-generative methods across the board. In the cases of MUTAG and BA-2Motifs, the generative methods RCExplainer, GraphCFE, and GSAT closely follow Grad-CAM and Occlusion in terms of faithfulness. Regarding BBBP and BA-MultiShapes, both generative and non-generative methods exhibit similar results. Consequently, generative methods achieve state-of-the-art performance on benchmark graph datasets. Furthermore, we demonstrate that generative methods possess additional desirable properties, such as efficiency and generalization capacity, which make them more appealing than non-generative methods.

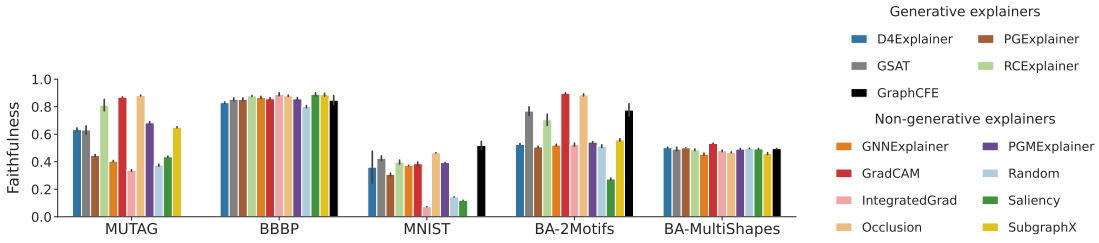


Figure 1: Faithfulness of explainability methods. On the y-axis, we report the faithfulness computed as $1 - fidelity{-}_{acc}$. On the x-axis, generative methods are always on the left-hand side of the methods (the bars). If the score is close to 1, the explanation is very faithful. The score is averaged over all explanations with less than 20 edges to enforce sparse and human-intelligible explanations.

Efficiency To measure the efficiency of explainability methods, we report the computation time to produce an explanation for a new instance in Figure 2. Comparing generative methods with other learnable methods (e.g.GNNExplainer, PGMExplainer) in Figure 2, we observe that once the model is trained, generative explainability methods require shorter inference time than non-generative ones in general. The time is reported in logarithmic scale and generative methods always have inference times of the order of 10^0 or less, except for the case of RCExplainer for MNIST. The advantage of shorter inference time is especially pronounced on large-scale datasets, e.g.MNIST. We also report the time required to train a generative model from scratch in Table 4.

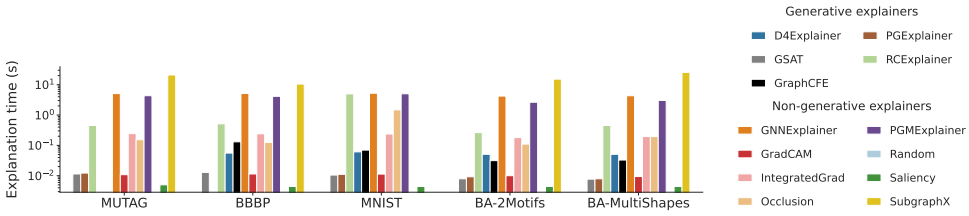


Figure 2: Inference time of explainability methods to explain one single graph. The computation time is averaged over 100 explanations over 5 seeds and reported in logarithmic scale.

Generalization To compare generative and non-generative explainability methods on their generalization capacity, we split the datasets into seen and unseen data. The split ratio is 90/10%. We further split the seen data into training, validation, and test set. The GNN model and the generative explainability methods are trained on

	D4Explainer	GraphCFE	GSAT	PGExplainer	RCExplainer
BA-2Motifs	475.3	320.9	23.1	11.6	194.0
BA-MultiShapes	309.3	211.8	20.0	17.2	251.0
BBBP	385.6	1350.0	-	26.0	303.4
MNIST	934.6	929.5	41.4	28.6	3271.0
MUTAG	253.1	-	79.8	27.7	434.6
Mean	471.6	703.1	41.1	22.2	890.8

Table 4: Training times (s) of the generative methods with 1 GPU (Nvidia GeForce RTX 2080)

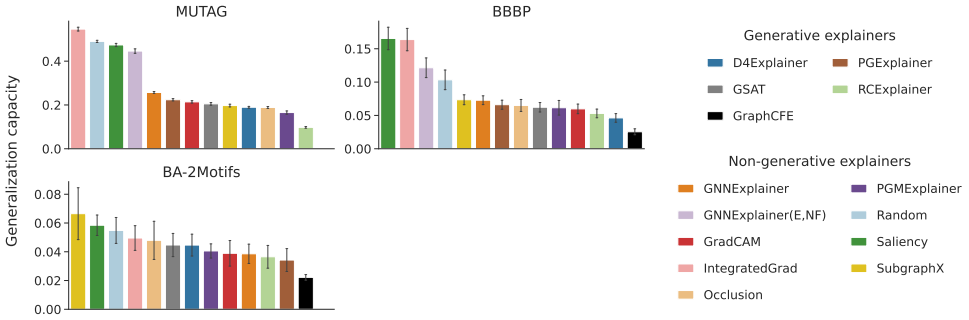


Figure 3: Generalization capacity of explainability methods is computed by subtracting the performance on data seen during training and the performance on unseen data. The lower the discrepancy reported on the y-axis, the better the method can generalize to unseen data. GNNExplainer indicates the explanations at only edge-level and GNNExplainer(E,NF) represents the explanations for both edges and node features.

the seen data. For non-generative methods, we explain 100 graphs from the seen dataset. Then, we test the trained methods on the unseen data. In Figure 3, we report the scores discrepancies between the test set of the seen data and the 10% unseen data for each explainability method. We also visualize the standard error on the five random seeds in Figure 3. Methods with higher absolute score discrepancies cannot generalize well to unseen data, while the ones with lower score discrepancies have a powerful generalization capacity. We can observe from Figure 3 that generative explainability methods have lower scores than non-generative methods across three datasets in general, which demonstrates the better generalization capacity.

6 Conclusion

In this paper, we present a comprehensive review of explanation methods for Graph Neural Networks (GNNs) from the perspective of graph generation. By proposing a unified optimization objective for generative explanation methods, encompassing Attribution and Information constraints, we provide a framework to analyze and compare existing approaches. Our study reveals shared characteristics and distinctions among current methods, laying the foundation for future advancements in the field. Moreover, we highlight the advantages and limitations of different approaches in terms of explanation performance, efficiency, and generalizability through empirical results. Notably, generative-based approaches demonstrate enhanced efficiency and generalizability compared to instance-dependent methods. Overall, our work contributes to the advancement of transparent and trustworthy graph-based models, paving the way for improved outcomes in various applications through better feature extraction and understanding of complex graph-structured data.

References

[1] Chirag Agarwal, Owen Queen, Himabindu Lakkaraju, and Marinka Zitnik. Evaluating explainability for graph neural networks. Scientific Data, 10(1):144, 2023.

[2] Ulrich Aïvodji, Alexandre Bolot, and Sébastien Gambs. Model extraction from counterfactual explanations. arXiv preprint arXiv:2009.01884, 2020.

[3] Kenza Amara, Rex Ying, Zitao Zhang, Zhihao Han, Yinan Shan, Ulrik Brandes, Sebastian Schemm, and Ce Zhang. Graphframex: Towards systematic evaluation of explainability methods for graph neural networks. arXiv preprint arXiv:2206.09677, 2022.

[4] Maximilian Augustin, Valentyn Boreiko, Francesco Croce, and Matthias Hein. Diffusion visual counterfactual explanations. arXiv preprint arXiv:2210.11841, 2022.

[5] Steve Azzolin, Antonio Longa, Pietro Barbiero, Pietro Liò, and Andrea Passerini. Global explainability of gnns via logic combination of learned concepts. arXiv preprint arXiv:2210.07147, 2022.

[6] Federico Baldassarre and Hossein Azizpour. Explainability techniques for graph convolutional networks. CoRR, abs/1905.13686, 2019.

[7] Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeswar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and R Devon Hjelm. Mine: mutual information neural estimation. arXiv preprint arXiv:1801.04062, 2018.

[8] Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. In NeurIPS, 2021.

[9] Pietro Bongini, Monica Bianchini, and Franco Scarselli. Molecular generative graph neural networks for drug discovery. Neurocomputing, 450:242–252, 2021.

[10] Anshika Chaudhary, Himangi Mittal, and Anuja Arora. Anomaly detection using graph neural networks. In 2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon), pages 346–350. IEEE, 2019.

[11] Dawei Cheng, Fangzhou Yang, Sheng Xiang, and Jin Liu. Financial time series forecasting with multi-modality graph neural network. Pattern Recognition, 121:108218, 2022.

[12] Enyan Dai, Tianxiang Zhao, Huaisheng Zhu, Junjie Xu, Zhimeng Guo, Hui Liu, Jiliang Tang, and Suhang Wang. A comprehensive survey on trustworthy graph neural networks: Privacy, robustness, fairness, and explainability. arXiv preprint arXiv:2204.08570, 2022.

[13] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In The world wide web conference, pages 417–426, 2019.

[14] Madelyn Glymour, Judea Pearl, and Nicholas P Jewell. Causal inference in statistics: A primer. John Wiley & Sons, 2016.

[15] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks, 2020.

[16] Qiang Huang, Makoto Yamada, Yuan Tian, Dinesh Singh, Dawei Yin, and Yi Chang. Graphlime: Local interpretable model explanations for graph neural networks. CoRR, abs/2001.06216, 2020.

[17] Guillaume Jeanneret, Loïc Simon, and Frédéric Jurie. Diffusion models for counterfactual explanations. In Proceedings of the Asian Conference on Computer Vision, pages 858–876, 2022.

[18] José Jiménez-Luna, Francesca Grisoni, and Gisbert Schneider. Drug discovery with explainable artificial intelligence. Nature Machine Intelligence, 2(10):573–584, 2020.

[19] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. CoRR, abs/1609.02907, 2016.

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

[21] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[22] Wenqian Li, Yinchuan Li, Zhigang Li, Jianye Hao, and Yan Pang. Dag matters! gflownets enhanced explainer for graph neural networks. arXiv preprint arXiv:2303.02448, 2023.

[23] Yiqiao Li, Jianlong Zhou, Sunny Verma, and Fang Chen. A survey of explainable graph neural networks: Taxonomy and evaluation metrics. arXiv preprint arXiv:2207.12599, 2022.

[24] Yiqiao Li, Jianlong Zhou, Boyuan Zheng, and Fang Chen. Ganexplainer: Gan-based graph neural networks explainer. arXiv preprint arXiv:2301.00012, 2022.

[25] Wanyu Lin, Hao Lan, and Baochun Li. Generative causal explanations for graph neural networks. In International Conference on Machine Learning, pages 6666–6679. PMLR, 2021.

[26] Wanyu Lin, Hao Lan, Hao Wang, and Baochun Li. Orphicx: A causality-inspired latent variable model for interpreting graph neural networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 13729–13738, 2022.

[27] Dongsheng Luo, Wei Cheng, Dongkuan Xu, Wenchao Yu, Bo Zong, Haifeng Chen, and Xiang Zhang. Parameterized explainer for graph neural network. In NeurIPS, 2020.

[28] Jing Ma, Ruocheng Guo, Saumitra Mishra, Aidong Zhang, and Jundong Li. Clear: Generative counterfactual explanations on graphs. arXiv preprint arXiv:2210.08443, 2022.

[29] Siqi Miao, Mia Liu, and Pan Li. Interpretable and generalizable graph learning via stochastic attention mechanism. In International Conference on Machine Learning, pages 15524–15543. PMLR, 2022.

[30] Tri Minh Nguyen, Thomas P Quinn, Thin Nguyen, and Truyen Tran. Counterfactual explanation with multi-agent reinforcement learning for drug target prediction. arXiv preprint arXiv:2103.12983, 2021.

[31] Phillip E Pope, Soheil Kolouri, Mohammad Rostami, Charles E Martin, and Heiko Hoffmann. Explainability methods for graph convolutional neural networks. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 10772–10781, 2019.

[32] Phillip E. Pope, Soheil Kolouri, Mohammad Rostami, Charles E. Martin, and Heiko Hoffmann. Explainability methods for graph convolutional neural networks. In CVPR, pages 10772–10781, 2019.

[33] Mario Alfonso Prado-Romero, Bardh Prenkaj, Giovanni Stilo, and Fosca Giannotti. A survey on graph counterfactual explanations: Definitions, methods, evaluation. arXiv preprint arXiv:2210.12089, 2022.

[34] Michael Sejr Schlichtkrull, Nicola De Cao, and Ivan Titov. Interpreting graph neural networks for NLP with differentiable edge masking. CoRR, abs/2010.00577, 2020.

[35] Yunsheng Shi, Zhengjie Huang, Wenjin Wang, Hui Zhong, Shikun Feng, and Yu Sun. Masked label prediction: Unified massage passing model for semi-supervised classification. CoRR, abs/2009.03509, 2020.

[36] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In ICML, volume 70, pages 3319–3328, 2017.

[37] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[38] Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation. arXiv preprint arXiv:2209.14734, 2022.

[39] Minh N. Vu and My T. Thai. Pgm-explainer: Probabilistic graphical model explanations for graph neural networks. In NeurIPS, 2020.

[40] Jianian Wang, Sheng Zhang, Yanghua Xiao, and Rui Song. A review on graph neural network methods in financial applications. arXiv preprint arXiv:2111.15367, 2021.

[41] Xiang Wang, Ying-Xin Wu, An Zhang, Xiangnan He, and Tat-Seng Chua. Towards multi-grained explainability for graph neural networks. In Proceedings of the 35th Conference on Neural Information Processing Systems, 2021.

[42] Xiang Wang, Yingxin Wu, An Zhang, Fuli Feng, Xiangnan He, and Tat-Seng Chua. Reinforced causal explainer for graph neural networks. IEEE Trans. Pattern Anal. Mach. Intell., 2022.

[43] Xiaoqi Wang and Han-Wei Shen. Gnninterpreter: A probabilistic generative model-level explanation for graph neural networks. arXiv preprint arXiv:2209.07924, 2022.

[44] Dana Warmsley, Alex Waagen, Jiejun Xu, Zhining Liu, and Hanghang Tong. A survey of explainable graph neural networks for cyber malware analysis. In 2022 IEEE International Conference on Big Data (Big Data), pages 2932–2939. IEEE, 2022.

[45] Geemi P Wellawatte, Aditi Seshadri, and Andrew D White. Model agnostic generation of counterfactual explanations for molecules. Chemical science, 13(13):3697–3705, 2022.

[46] Oliver Wieder, Stefan Kohlbacher, Mélaine Kuenemann, Arthur Garon, Pierre Ducrot, Thomas Seidel, and Thierry Langer. A compact review of molecular property prediction with graph neural networks. Drug Discovery Today: Technologies, 37:1–12, 2020.

[47] Bingzhe Wu, Jintang Li, Junchi Yu, Yatao Bian, Hengtong Zhang, CHaochao Chen, Chengbin Hou, Guoji Fu, Liang Chen, Tingyang Xu, et al. A survey of trustworthy graph learning: Reliability, explainability, and privacy protection. arXiv preprint arXiv:2205.10014, 2022.

[48] Zhitao Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. Gnnexplainer: Generating explanations for graph neural networks. In NeurIPS, pages 9240–9251, 2019.

[49] Junchi Yu, Tingyang Xu, Yu Rong, Yatao Bian, Junzhou Huang, and Ran He. Graph information bottleneck for subgraph recognition. In International Conference on Learning Representations, 2021.

[50] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. XGNN: towards model-level explanations of graph neural networks. In Rajesh Gupta, Yan Liu, Jiliang Tang, and B. Aditya Prakash, editors, KDD, pages 430–438, 2020.

[51] Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. Explainability in graph neural networks: A taxonomic survey. CoRR, 2020.

[52] Hao Yuan, Haiyang Yu, Jie Wang, Kang Li, and Shuiwang Ji. On explainability of graph neural networks via subgraph explorations. ArXiv, 2021.

[53] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13, pages 818–833. Springer, 2014.

[54] He Zhang, Bang Wu, Xingliang Yuan, Shirui Pan, Hanghang Tong, and Jian Pei. Trustworthy graph neural networks: Aspects, methods and trends. arXiv preprint arXiv:2205.07424, 2022.

[55] 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.

Graph Contrastive Learning: An Odyssey towards Generalizable, Scalable and Principled Representation Learning on Graphs

Yan Han[†] Yuning You[‡] Wenqing Zheng[†] Scott Hoang[†] Tianxin Wei[§]
Majdi Hassan[¶] Tianlong Chen[†] Ying Ding[†] Yang Shen[‡] Zhangyang Wang[†]

[†]University of Texas at Austin
{yh9442, w.zheng, hoangd, tianlong.chen, atlaswang}@utexas.edu
ying.ding@austin.utexas.edu

[‡]Texas A&M University
{yuning.you, yshen}@tamu.edu

[§]University of Illinois Urbana-Champaign
twei10@illinois.edu

[¶]AbbVie
majdi.mhas@gmail.com

Abstract

Graph Contrastive Learning (GCL), an uprising regime of learning representations of graph-structured data, has gained significant attention in recent years. At its core, GCL leverages the idea of comparing different views of a graph to learn representations that capture desirable characteristics of the graph structures. GCL has been applied to a wide range of graph-structured data, including attributed graphs, multi-relational graphs, temporal graphs, hierarchical graphs, heterogeneous graphs, and hypergraphs. The learned graph representations yield predictive performance that generalizes well in various downstream tasks at the node, link, and graph levels, and can scale up to graphs with millions of nodes. In this paper, we present a review of representative GCL approaches with a major emphasis on our own recent efforts. Beginning with the original GCL approach with ad-hoc view generation and simple homogeneous graphs, we demonstrate how the framework can be further extended to more complex heterogeneous graphs and hypergraphs, as well as improved via principled view generation towards generalizability, fairness, interpretability, and other aspects. Theoretical explorations are covered at the end. In conclusion, we discuss the future prospects and ongoing challenges in the field of GCL.

1 Introduction

Graph-structured data are ubiquitous in various real-world applications, including social networks, biological networks, transportation systems, and recommender systems [25, 17, 57]. The analysis and learning from graph data have gained increasing importance as they can unveil hidden patterns and relationships, thereby enhancing decision-making in practical scenarios. In recent years, graph contrastive learning (GCL) has emerged as a promising approach for graph representation learning. GCL has demonstrated remarkable success in capturing the underlying structural properties of graphs and achieving generalized predictive performance.

The key concept behind GCL, inspired by image representation learning [5], is simple yet effective: comparing different views of a graph to learn representations that encapsulate its desirable structural properties, which can scale up to large graphs. However, determining what and how to contrast presents a non-trivial challenge when it comes to graph data. Unlike images, graph data exhibit high heterogeneity across applications in terms of both semantics and graph types (e.g., social networks versus molecular scaffolds). Consequently, achieving a universally beneficial design is exceedingly difficult.

In this paper, we present a comprehensive review of various GCL approaches, with a particular focus on our recent contributions. We start from introducing the vanilla GCL approach, known as vanilla GCL, which involves the utilization of ad-hoc view generation and primarily targets simple homogeneous graphs. Subsequently, we systematically explore the extensions of the GCL framework to accommodate more intricate heterogeneous graphs and hypergraphs. This involves the integration of principled view generation techniques to enhance generalizability, fairness, interpretability, and other important aspects. We in addition cover the preliminary but critical investigations of the theoretical aspects, providing valuable insights into the underlying mechanisms and potential limitations of GCL. We conclude the paper by discussing future prospects and open challenges in the GCL field, including potential directions for further research and development in this area.

2 GCL Approaches and Graph Types

2.1 Homogeneous Graphs

A homogeneous graph, denoted as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\} \in \mathbb{G}$, is characterized by a set of vertices $\mathcal{V} = \{v_1, ..., v_{|\mathcal{V}|}\}$ and a set of edges $\mathcal{E} = \{(v_i, v_j)|v_i, v_j \in \mathcal{V}\}$. In different contexts, each vertex and edge can be associated with specific attributes for feature representation. For example, in social networks [36], vertices may correspond to user side-information and edges to connections, while in molecular graphs [7], vertices may represent atoms and edges may represent bond types. These attributes can be mapped to vectorial representations of dimension D using graph neural networks (GNNs) [24, 48, 64]. Thus, GNNs can be represented as $f_\theta : \mathbb{G} \to \mathbb{R}^D$, where θ denotes the parameters of the GNN, enabling downstream utility.

In the supervised learning setting, the annotated dataset $\mathcal{D}_{\text{lab}} = \{(\mathcal{G}_1, \mathcal{Y}_1), ...\}$ is provided, where machine learning models, such as GNNs, can be trained to make predictions on new, unseen data. However, the effectiveness of supervised learning is hindered by the challenge of limited graph labels [37]. One dominant solution to address the challenge is to employ self-supervised learning, where models are pre-trained on large-scale unlabeled data [5, 9]. The rationale behind this approach is well-appropriate principled objectives could enhance their generalizability. Fortunately, accessing unlabeled datasets $\mathcal{D}_{\text{unlab}} = \{\mathcal{G}_1, ...\}$ for graphs is often viable. A key question then arises: *how can self-supervised objectives be designed specifically for graph-structured data?*

Among the various self-supervised tasks for graphs (for a more comprehensive review, please refer to [62, 34]), graph contrastive learning (GCL) [68, 50, 79, 20, 43, 40] stands out due to its consistently generalizable performance across diverse applications. The fundamental concept of GCL is to compare different views of a graph to learn representations that capture the desired structural properties of the graph (refer to Figure 1 for the overall pipeline). In GCL, prior knowledge is explicitly incorporated into graph neural networks (GNNs) through the construction of graph views. Consequently, the primary focus of GCL research is to investigate effective methods for constructing graph views that incorporate appropriate inductive biases. The training of GCL with a batch of graph samples $\{\mathcal{G}_1, ..., \mathcal{G}_N\}$ is then formulated using the NT-Xent loss [5] as follows:

$$\min_{\theta,\phi} \mathcal{L}_{\text{GCL}}(\theta, \phi) = \min_{\theta,\phi} -\frac{1}{N}\sum_{n=1}^{N} \log \frac{\exp\left(\text{sim}\left(g_\phi \circ f_\theta(\tilde{\mathcal{G}}_{n,1}), g_\phi \circ f_\theta(\tilde{\mathcal{G}}_{n,2})\right)/\tau\right)}{\sum_{n'=1,n'\neq n}^{N} \exp\left(\text{sim}\left(g_\phi \circ f_\theta(\tilde{\mathcal{G}}_{n,1}), g_\phi \circ f_\theta(\tilde{\mathcal{G}}_{n',2})\right)/\tau\right)} \tag{25}$$

where $g_\phi(\cdot)$ represents the projection head implemented as a multi-layer perceptron [5], $\tilde{\mathcal{G}}_{n,i} = h_i(\mathcal{G}_n)$ denotes

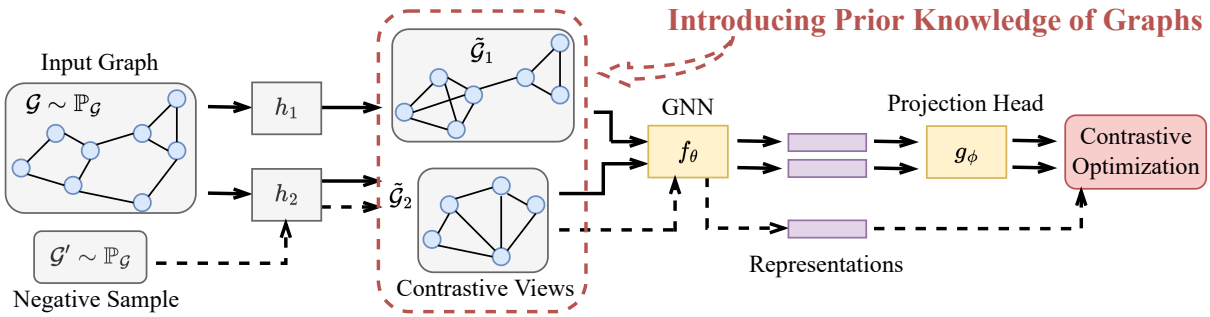


Figure 1: The generic pipeline of homogeneous GCL.

the contrastive view of $\mathcal{G}_n$, $\mathcal{G}'_n$ is treated as the negative sample, $\mathrm{sim}(\cdot,\cdot)$ is the similarity function (typically implemented using cosine similarity $\mathrm{sim}(z_1, z_2) = z_1^\top z_2 / \|z_1\| \|z_2\|$), and $\tau \in \mathbb{R}_{>0}$ is the temperature parameter.

Graph data augmentations for view construction. A well-established approach to construct contrastive views is through graph data augmentations [68, 80, 10, 76, 77, 35], which effectively introduce inductive biases specific to the target applications. For example, in one of the earliest works [68], four generic graph augmentations are designed, as shown in Table 1, each with an underlying prior incorporated. These augmentations operate in the spatial domain and include node dropping, edge perturbation, attribute masking, and subgraph transformations, the resulting views of which capture different aspects of the graph's semantics.

Table 1: Four generic augmentation strategies (in the spatial domain) with the corresponding underlying inductive bias.

Data Augmentation	Type	Underlying Prior
Node dropping	Nodes, edges	Vertex missing does not alter semantics.
Edge perturbation	Edges	Semantic robustness against connectivity variations.
Attribute masking	Nodes	Semantic robustness against partial attribute loss.
Subgraph	Nodes, edges	Local structure can provide hints to full semantics.

In addition to augmentations in the spatial domain, spectral augmentations have been proposed for graph data, leveraging the graph spectrum where the semantic information is more accessible [16, 72, 31]. Furthermore, some works explore augmentations in the latent space implicitly learned by the model [61, 26, 51], which is believed to better capture the underlying semantics. Other approaches focus on augmenting specific graph model-based parameters, such as graphons [18, 41] or contextual stochastic block models (CSBMs) [55], based on the explicit downstream data generation assumptions.

Overall, the choice of graph data augmentations plays a crucial role in constructing effective contrastive views and capturing the desired inductive biases for the specific graph-structured data. In certain designated applications, augmentations can be specifically designed to cater to the unique characteristics of the data. For example, small molecules have been shown to benefit from motif-based [54, 13] or energy-guided [33] perturbations. Similarly, protein structures can be augmented through cropping while preserving consecutive amino-acid sequences [73, 71]. The construction of effective graph views for contrastive learning, which are robust across diverse domains or exhibit strong generalizability in specific applications, remains an active area of research.

2.2 Heterogeneous Graphs

A heterogeneous graph is defined as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{T}_\mathcal{V}, \mathcal{T}_\mathcal{E}\} \in \mathbb{G}$, where $\mathcal{V} = \{v_1, ..., v_{|\mathcal{V}|}\}$ represents the set of vertices, $\mathcal{E} = \{(v_i, v_j) | v_i, v_j \in \mathcal{V}\}$ denotes the set of edges, $\mathcal{T}_\mathcal{V}$ denotes the set of different vertex types, and $\mathcal{T}_\mathcal{E}$ represents the set of different edge types. In various contexts, each vertex and edge in a heterogeneous graph may be associated with specific attributes (e.g., user profiles and relationships in social networks [44], or gene expressions and regulatory interactions in biological networks [39]), which serve as features for graph learning.

Differing from homogeneous graphs, the complicated relations in heterogeneous graphs are more effec-

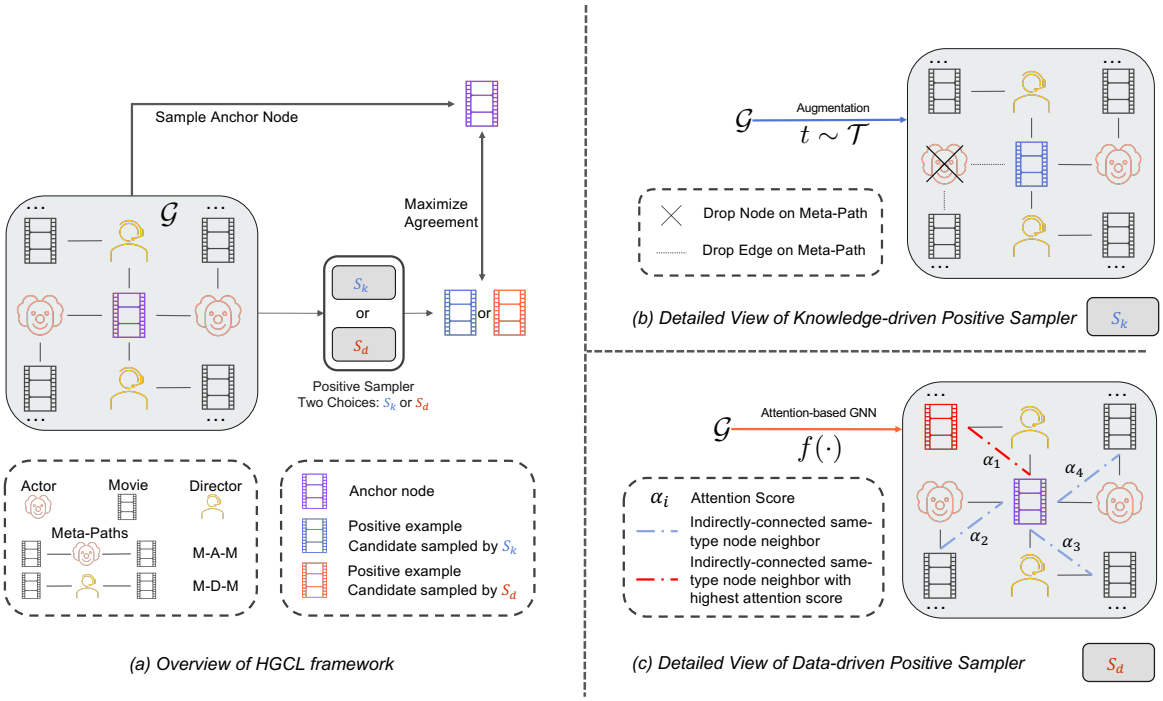


Figure 2: HGCL framework: (a) **Overview**: Anchor node u and positive example node v in a heterogeneous graph $\mathcal{G}$ are processed with a GNN backbone and projection head using a contrastive loss. (b) S_k **(knowledge-driven)**: Pre-defined meta-paths guide positive example node v_k generation. (c) S_d **(data-driven)**: GNN-based encoder with attention module generates positive example node v_d.

tively captured by meta-paths. A meta-path [45, 11] P is a path that connects multiple nodes in the form of $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \dots \xrightarrow{R_l} A_{l+1}$ (abbreviated as $A_1 A_2 \dots A_{l+1}$). It describes a composition of different relations $R_1, R_2, \dots, R_l$ that link objects A_1 and A_{l+1}. For example, in the heterogeneous graph shown in Figure 2(a), there are two different meta-paths: Movie-Director-Movie (MDM), which represents the co-director relation between two movies, and Movie-Actor-Movie (MAM), which represents the co-actor relation between two movies. Different meta-paths reveal diverse semantics and guide the connection of even distant objects based on their semantic similarities. Utilizing meta-paths as prior knowledge in positive sampling ensures that the selected candidates are semantically related to some extent.

Heterogeneous graph contrastive learning (HGCL) has witnessed significant progress through various methods. HeCo [53] utilizes meta-path-based random walks for self-supervised learning, while STENCIL [78] employs structural templates to encode neighborhood information. HGCLR [4] learns embeddings by contrasting multiple meta-path-derived views, and CPT-HG [23] captures node relations through pairwise contrastive learning. MVSE [75] combines intra-view and inter-view contrastive learning tasks using different meta-path-based views. Generally, contrastive learning methods can be classified into two categories: **knowledge-driven** and **data-driven**, with a primary focus on generating positive views.

Knowledge-driven views. As illustrated in Figure 2(b), the framework leverages meta-paths to drop nodes or perturb edges, resulting in related views. This strategy shares similarities with homogeneous graphs, with the key difference being that the augmentation in heterogeneous graphs depends on meta-paths.

Data-driven views. As demonstrated in Figure 2(c), data-driven methods rely on attention mechanisms employed in graph neural networks [52, 49, 15]. The attention score, e_{ij}, of node j's embedding to node i serves

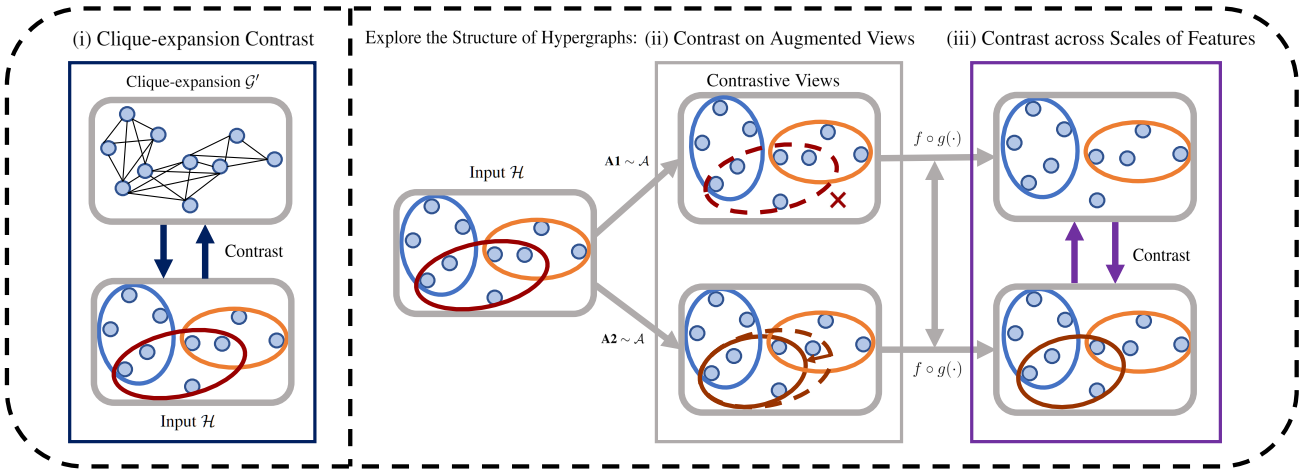


Figure 3: Three major lines of HyperGCL pipelines: (i) clique-expansion contrast, (ii) contrasting on augmented hypergraph views, and (iii) contrasting across scales of augmented hypergraph features.

for the learnable sampling distribution as:

$$e_{ij} = \text{LeakyReLU}(a(Wh_i||Wh_j)), \quad \alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}i}\exp(e_{ik})} \tag{26}$$

To ensure differentiability in selecting the positive example $\mathbf{v}^+$, the Gumbel-Softmax trick [22] is introduced, using the softmax function as a continuous, differentiable approximation (Gumbel-Softmax trick), facilitating joint updates of the learned sampling distribution p_d during training, resulting in a dynamic and adaptive contrastive learning framework:

$$v_j^+ = \frac{\exp((\log(\alpha_{ij}) + g_j)/\tau)}{\sum_{m=1,m\neq i}^{k}\exp(\log(\alpha_{im}) + g_m)/\tau)}, \text{ for } j = 1, \cdots, \not{i}, \cdots, k \tag{27}$$

2.3 Hypergraphs

Hypergraphs extend the concept of homogeneous and heterogeneous graphs by allowing many-body interactions across nodes, represented by hyperedges. They have attracted significant attention in the research community [14, 65, 6]. A hypergraph is denoted as $\mathcal{H} = \{\mathcal{V}, \mathcal{E}\} \in \mathbb{H}$, where $\mathcal{V} = \{v_1, ..., v_{|\mathcal{V}|}\}$ is the set of vertices and $\mathcal{E} = \{e_1, ..., e_{|\mathcal{E}|}\}$ is the set of hyperedges. Each hyperedge $e_n = \{v_1, ..., v_{|e_n|}\}$ represents a higher-order interaction among a set of vertices. Hypergraph neural networks (HyperGNNs) [14, 65, 6] have been proposed as state-of-the-art approaches to encode such complex structures, mapping the hypergraph to a D-dimensional latent space via $f : \mathbb{H} \to \mathbb{R}^D$ using higher-order message passing. For contrastive learning, a projection head $h(\cdot)$ is applied to $f(\cdot)$. Currently, most hypergraph contrastive learning (HyperGCL) methods focus on node-level applications.

Table 2: Five generic augmentation operations for HyperGCL.

Hypergraph Augmentations
Naïve Hyperedge Perturbation
Generalized Hyperedge Perturbation
Vertex Dropping
Attribute Masking
Subgraph
Generative Augmentation

When dealing with complex relationships in hypergraphs, the challenge is how to construct contrastive views for hypergraphs. However, building effective hypergraph views is non-trivial due to the overly complicated topology of hypergraphs. Unlike graphs, where there are $\binom{N}{2}$ possibilities for one edge with N vertices, hyperedges in hypergraphs can have 2^N possibilities. To address this challenge, three lines of methods have

emerged in HyperGCL research, which focus on (i) clique-expansion contrast, (ii) hypergraph view generation, and (iii) hypergraph objective augmentation, as summarized in Figure 3.

Clique-expansion contrast. The first line of HyperGCL methods contrasts the representations of hypergraphs with clique-expansion views [59, 3]. This method is intuitive but computationally expensive in terms of time and memory, as it requires optimizing multiple neural networks of different modalities. Additionally, contrasting between clique expansion poses the risk of losing higher-order awareness by bringing the representations of hypergraphs and graphs closer together [56].

Contrasting on augmented hypergraph views. The second line of methods explores the structure of hypergraphs itself to construct contrastive views [56]. They assess whether generic augmentations are suitable for HyperGCL. Since hypergraphs are composed of hyperedges and vertices, they propose two strategies to augment hyperedges: direct perturbation on hyperedges and perturbation on the "edges" between hyperedges and vertices in the converted bipartite graph. To augment vertices, they adopt three schemes: vertex dropping, attribute masking, and subgraph, which are borrowed from graph-structured data augmentations [68]. Their findings show that while vertex augmentations benefit graphs more, hypergraphs mostly benefit from hyperedge augmentations, revealing that higher-order information encoded in hyperedges is usually more downstream-relevant than information in vertices.

Contrasting across scales of augmented hypergraph features. The last line of methods [29, 42] focuses on contrasting augmented features beyond the node-level. The main idea is to perform multi-level contrast, aiming to maximize the agreement between the same node, the node members of the same hyperedge, and each hyperedge and its node members in two augmented views. This approach expects that complementary information can be captured from different views to enhance HyperGCL.

2.4 Principled Graph Views for Contrastive Learning

The construction of appropriate contrastive views is crucial in GCL, but it often relies on empirical rules of thumb, which can vary significantly depending on the nature of the graph dataset. Therefore, the question arises: *could we develop principled ways to construct graph views for contrastive learning?* The answer lies in two folds: defining the space of GCL views and formulating principles to search within that space (Figure 4).

Space of graph contrastive views. Defining a good search space is essential for well-behaved search algorithms. The view constructor can be represented by a mapping function $h_\psi : \mathbb{G} \to \tilde{\mathbb{G}}$, where $\tilde{\mathcal{G}} = h_\psi(\mathcal{G})$ and ψ represents the parameters of the mapping function. The search space is defined on this family of functions, specifically on the parameters ψ. Various approaches have been proposed to construct the function space, such as using learnable sampling distributions combined with prefabricated graph augmentation functions [67], masking operators on topology or node features [80, 46], and training graph generative models to define the augmentation space in a data-driven manner [69, 66]. While the construction of search spaces is domain-agnostic, there is a need for further research to explore how to construct search spaces tailored to specific applications, e.g., in molecular graph analysis, chemical knowledge can be incorporated to define the graph augmentation functions. Future research can focus on customizing the search space construction to the characteristics of the target application.

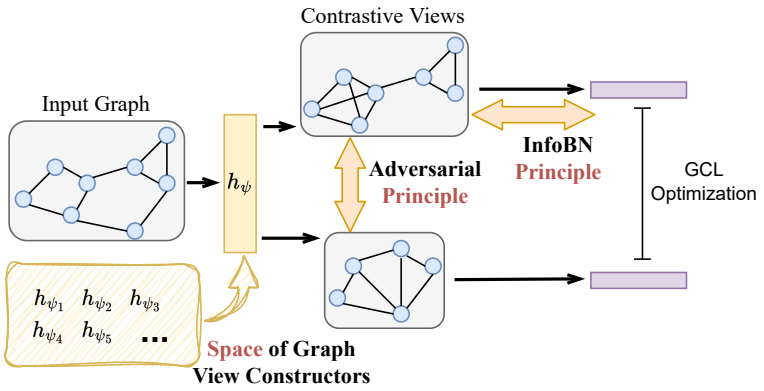


Figure 4: Two indispensable steps for principled graph view construction: defining view space and formulating search principles.

Principles for view searching. The choice of graph contrastive views can be sensitive to different datasets in

GCL. However, evidence shows that views derived from certain principles, conditioned on the dataset, provide more robust benefits. View searching can be formulated as a bi-level optimization problem, where the upper-level optimization aims to minimize the GCL loss with respect to the parameters θ and ϕ, while the lower-level optimization finds the optimal parameters ψ^* that minimize a principled objective $\mathcal{L}_{\text{Principle}}$. Various principles have been proposed for view construction. For example, adversarial training can enforce GCL to optimize on "difficult" views [67], leveraging graph properties can guide the view construction [80], and the information bottleneck principle can eliminate superficial features from raw graph data [69, 46, 63]. Future research can focus on unifying different principles and understanding the relationships among them. Additionally, theoretical analysis to bridge principled view construction and downstream performance is needed for further development.

2.5 Principled View towards Interpretability and Fairness

While graph contrastive learning has shown success in various tasks, fairness and interpretability are important aspects that have received less attention in this area. In this section, we discuss the principled view for fairness and interpretability in graph contrastive learning, highlighting recent studies.

Fairness. Fairness in graph representation learning is crucial to prevent biased results, especially towards underrepresented groups. Graph contrastive learning methods can benefit from incorporating fairness-aware data augmentations to promote fair and unbiased representations. Studies such as Graphair [32], as well as other approaches [28, 27], propose fairness-aware data augmentations that are learned from data. These augmentations, which can be integrated into graph contrastive learning frameworks, aim to mitigate sensitive information while preserving other useful information. By doing so, they improve the fairness-accuracy trade-off performance in various node classification datasets, leading to more equitable graph contrastive learning outcomes.

Interpretability. Interpretability is another important aspect of graph contrastive learning, as it helps users understand and trust the decisions made by graph neural networks (GNNs). Task-Agnostic GNN Explainer (TAGE) [60] is a self-supervised, task-independent explanation approach that can be applied to GNNs used in graph contrastive learning. TAGE enables the explanation of GNN embedding models with unseen downstream tasks and allows efficient explanation of multitask models. Integrating TAGE into graph contrastive learning frameworks can significantly enhance their explanation efficiency while achieving similar or better explanation quality than existing state-of-the-art GNN explanation methods.

Another approach to improve the interpretability of GNNs in graph contrastive learning is to focus on their reasoning capabilities. Existing neural reasoners often struggle with out-of-distribution (OOD) test data featuring larger input sizes. A recent study [1] proposes data augmentation procedures that leverage causal frameworks to develop self-supervised objectives, which can also be applied to graph contrastive learning. By incorporating these data augmentation procedures into graph contrastive learning, the OOD generalization capabilities of the reasoner can be improved, resulting in better performance on OOD test data.

In summary, enhancing fairness and interpretability in graph contrastive learning is crucial for developing trustworthy and unbiased node representations. By incorporating fairness-aware data augmentations, self-supervised objectives, and interpretable explanation methods, we can create more robust and equitable graph contrastive learning frameworks that remain interpretable and generalizable across various application domains.

2.6 Theoretical Exploration

Although numerous algorithms and principles have been developed for GCL pretraining, the explicit theoretical connection between pretraining and downstream fine-tuning performance still lags behind. The gap between pretraining and fine-tuning performance can be attributed to several factors, including the misalignment between the optimization objectives of pretraining and fine-tuning, the discrepancy between the data distributions used in pretraining and fine-tuning, and the inductive biases encoded in neural networks. While the latter two factors have been more heavily studied in graph out-of-distribution generation [2, 70], little work has been done to analyze

the sources of error arising from the misalignment between pretraining and fine-tuning optimization objectives, especially in the context of GCL.

One major challenge in analyzing the misalignment between pretraining and fine-tuning in GCL is the diverse range of downstream applications for graph-structured data. Unlike Euclidean data, which has more standardized downstream tasks such as image classification [12, 19], graph-structured data is used in a wide range of applications, from molecule generation to social network analysis. This diversity makes it difficult to establish appropriate assumptions about downstream data distributions for analysis.

In a recent study, Trivedi et al. [47] attempted to bridge the gap between pretraining and fine-tuning in GCL by assuming a highly generic "label-preserving" behavior of graph views. They justified this assumption through the use of graph edit distances between contrastive views and label-preserved samples, which measure the similarity between two graphs based on the number of operations required to transform one into the other. While this approach provides a starting point for analyzing the misalignment between pretraining and fine-tuning in GCL, more fine-grained analyses are needed to fully understand the relationship between these two processes.

3 Applications and Benchmarks

3.1 Graph Types

The benchmarks and datasets used to evaluate graph contrastive learning methods come from a variety of domains and cover a range of graph types and tasks. These graph types include bioinformatics, social networks, molecules, computer vision, and synthetic graphs. Each of these types is characterized by unique properties that affect the tasks they are suited for and the evaluation metrics used to assess contrastive learning methods.

TUDataset. TUDataset [38] is extensively used in the evaluation of graph contrastive learning methods. It contains a diverse set of graph data from various domains, including small molecules and proteins, computer vision, and social networks. The datasets within TUDataset cover different types of relational networks, including graphs with discrete or continuous node and edge attributes. The small molecules datasets contain class labels representing toxicity or biological activity, with the graphs representing molecules and nodes representing atoms, while edges represent chemical bonds. The bioinformatics datasets in TUDataset represent macromolecules such as proteins and use a graph model where nodes represent secondary structure elements, annotated by their type, and several physical and chemical information, with edges connecting neighboring nodes. The computer vision datasets contain graphs representing various tasks such as image processing, fingerprint recognition, and letter recognition. Finally, the social network datasets within TUDataset include Reddit discussion threads, scientific collaboration networks, actor collaborations, and GitHub users, each with different tasks, such as distinguishing between discussion-based and question-answer-based subreddits, predicting the research field of researchers, predicting the genre of actor collaborations, and identifying GitHub users who starred popular repositories.

Pokec-z and Pokec-n. Pokec-z and Pokec-n [8] are two social graphs sampled from a larger Facebook-like social network in Slovakia called Pokec. The nodes in these graphs correspond to users living in two major regions, with the region information being used as the sensitive attribute. The Recidivism graph is built upon the information of defendants who got released on bail at the US state courts, where the edges are created based on the similarity of past criminal records and demographics. The sensitive attribute for this graph is race, where the node classification task is built upon classifying defendants into bail or no bail. Similarly, the Credit defaulter graph is generated by creating links between people based on the similarity of their spending and payment patterns, with labels for node classification corresponding to whether a person will handle the credit card payment or not, and age being used as the sensitive attribute.

MoleculeNet. MoleculeNet [58] is a collection of molecular graph datasets used for the prediction of different molecule properties. Each atom in the molecule is considered a node in the graph, with each bond considered an edge. The prediction of molecule properties is a graph-level task, and three graph classification tasks from MoleculeNet are used in the evaluation of contrastive learning methods.

PPI. The Protein-Protein Interaction (PPI) [81] dataset documents the physical interactions between proteins in 24 different human tissues. In PPI graphs, each protein is considered as a node with its motif and immunological features, and there is an edge between two proteins if they interact with each other. The prediction of each protein function is considered an individual task instead of a multi-class classification, and hence typical approaches require individual explainers for the 121 tasks.

ACM. The ACM [74] heterogeneous graph dataset comprises academic papers divided into three classes based on research areas. Each paper is associated with multiple attributes, such as authors, subjects, and publication venues. With an average of 3.33 authors per paper and one subject, the ACM dataset captures the collaboration and knowledge-sharing dynamics among researchers in various disciplines.

DBLP. In DBLP [15] heterogeneous graph dataset, the target nodes represent authors classified into four research areas. The dataset captures the relationships between authors, their publications, and the venues in which they are published. With an average of 4.84 papers per author, the DBLP dataset offers insights into the academic productivity and research contributions of authors across different fields.

Freebase. The Freebase [30] heterogeneous graph dataset contains information about movies, which are categorized into three genres. The dataset captures various relationships, such as the ones between movies, actors, directors, and writers. With an average of 18.7 actors, 1.07 directors, and 1.83 writers per movie, the Freebase dataset provides a comprehensive view of the complex interactions among entities in the film industry.

AMiner. The AMiner dataset [21] is a heterogeneous graph that focuses on academic papers extracted from a subset of the original dataset, divided into four research areas. In addition to capturing relationships between papers, authors, and references, the dataset includes an average of 2.74 authors and 8.96 references per paper. This comprehensive dataset offers valuable insights into the citation patterns, research trends, and collaborations in the academic world.

3.2 Special Tasks

In addition to the normal evaluation setting, GCL methods are evaluated in various special learning settings [68, 67, 69, 56, 28, 27, 32, 60], including semi-supervised learning, unsupervised representation learning, transfer learning, adversarial robustness, and the accuracy-fairness trade-off. These settings are important because they reflect real-world scenarios and challenges that GCL methods may encounter in practical applications, such as limited labeled data, unsupervised learning, transferability, robustness, and fairness. By evaluating the performance of GCL methods in these special settings, we can gain a better understanding of their strengths and limitations and improve their applicability and robustness in real-world scenarios.

Semi-supervised learning. Semi-supervised learning addresses the issue of limited labeled data in graph classification. Existing works on graph contrastive learning evaluate their models in this setting using small social network benchmarks and large-scale graph datasets. They compare their models' performance against conventional pre-training schemes such as adjacency information reconstruction and local and global representation consistency enforcement.

Unsupervised learning. Unsupervised learning is another important setting for graph contrastive learning because it allows for the learning of representations without any labeled data, which is useful for many applications where labeled data is scarce. Existing works on graph contrastive learning evaluate their models in unsupervised learning tasks using graph embeddings generated by unsupervised methods, which are then fed into a downstream SVM classifier.

Transfer learning. Transfer learning enables the evaluation of a model's transferability across different datasets, tasks, and domains. Existing works on graph contrastive learning evaluate their models in transfer learning tasks on molecular property prediction in chemistry and protein function prediction in biology. They pre-train and fine-tune their models on different datasets and evaluate the transferability of their pre-training schemes.

Adversarial robustness. Adversarial robustness is an important setting for graph contrastive learning because

it addresses the issue of adversarial attacks on graph data, which are becoming increasingly common in many applications. Existing works on graph contrastive learning evaluate their models' robustness on synthetic data to classify the component number in graphs, facing the RandSampling, GradArgmax, and RL-S2V attacks. They show that graph contrastive learning boosts GNN robustness compared to training from scratch under these evasion attacks.

Accuracy and fairness trade-off. The trade-off between accuracy and fairness is a crucial setting for graph contrastive learning because it enables the evaluation of a model's performance with respect to both accuracy and fairness metrics. Existing works on graph contrastive learning compare the accuracy-fairness trade-off performance of their models with several baselines using demographic parity as the fairness metric. They show that their model achieves the best ACC-DP trade-off compared to all fairness-aware baselines on three datasets.

4 Conclusion

The paper provides an overview of GCL, an emerging pipeline for generating generalizable graph representations that are crucial in real-world graph applications. The paper first introduces the vanilla formulations of GCL on various graph types and then discusses advanced variants guided by different principles, such as robustness, interpretability, and fairness. The paper also covers routine assessment and datasets commonly used for evaluating GCL methods. Based on the reviewed foundations, the paper summarizes several future perspectives and open challenges in the field of GCL.

Tailored pipeline for focused applications. While the reviewed methods are generally designed to be task-agnostic, real-world applications often require more focused approaches that leverage domain-specific knowledge. Designing tailored GCL pipelines that address domain-specific problems can leverage algorithmic advancements and incorporate domain priors. Examples of such tailored pipelines include applications in the fields of molecules and proteins. It is expected that more sophisticated and effective designs will be developed in the future.

Explicit theoretical bridge between pre-training and downstream. A notable phenomenon in graph pre-training is negative transfer, where inappropriate pre-training strategies can lead to performance degradation. This is due to the heterogeneous nature of graph data and the absence of a universal good prior for downstream tasks. In addition to being guided by implicit principles, there is a need for an explicit understanding of the relationship between GCL pre-training and downstream performance in theory. This will help answer questions about why and when to use pre-trained graph representations.

Handling more complicated and composed graph structures. With the rise of artificial general intelligence and digital medicine, the field of graph learning is entering an era of multi-modality learning, where in-silico models serve as proxies for real-world intelligent and biological systems. In this context, data structures are becoming more complex, with multi-modal graphs that are heterogeneous and contain multi-body relations. This presents a new frontier for graph learning, particularly in the area of generalizable representation learning that can handle extremely complex and composed graph data structures.

Unifying graph pre-training strategies under the umbrella of GCL. One advantage of the GCL pipeline is its simplicity and flexibility, allowing for the incorporation of various strategies into a unified framework. A future ambition is to further unify different lines of graph pre-training strategies, including both predictive and generative approaches, within this framework. This will provide a better understanding of the theoretical underpinnings and the optimal approach for specific applications by exploring the full space of graph pre-training methods within the GCL framework.

References

[1] Beatrice Bevilacqua, Kyriacos Nikiforou, Borja Ibarz, Ioana Bica, Michela Paganini, Charles Blundell, Jovana Mitrovic, and Petar Veličković. Neural algorithmic reasoning with causal regularisation. arXiv preprint arXiv:2302.10258, 2023.

[2] Beatrice Bevilacqua, Yangze Zhou, and Bruno Ribeiro. Size-invariant graph representations for graph classification extrapolations. In International Conference on Machine Learning, pages 837–851. PMLR, 2021.

[3] Derun Cai, Chenxi Sun, Moxian Song, Baofeng Zhang, Shenda Hong, and Hongyan Li. Hypergraph contrastive learning for electronic health records. In Proceedings of the 2022 SIAM International Conference on Data Mining (SDM), pages 127–135. SIAM, 2022.

[4] Mengru Chen, Chao Huang, Lianghao Xia, Wei Wei, Yong Xu, and Ronghua Luo. Heterogeneous graph contrastive learning for recommendation. In Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining, pages 544–552, 2023.

[5] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In International conference on machine learning, pages 1597–1607. PMLR, 2020.

[6] Eli Chien, Chao Pan, Jianhao Peng, and Olgica Milenkovic. You are allset: A multiset function framework for hypergraph neural networks. ICLR, 2022.

[7] David P Clark and Nanette J Pazdernik. Molecular biology. Elsevier, 2012.

[8] Enyan Dai and Suhang Wang. Say no to the discrimination: Learning fair graph neural networks with limited sensitive attribute information. In Proceedings of the 14th ACM International Conference on Web Search and Data Mining, pages 680–688, 2021.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.

[10] Kaize Ding, Zhe Xu, Hanghang Tong, and Huan Liu. Data augmentation for deep graph learning: A survey. ACM SIGKDD Explorations Newsletter, 24(2):61–77, 2022.

[11] 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.

[12] Simon S Du, Wei Hu, Sham M Kakade, Jason D Lee, and Qi Lei. Few-shot learning via learning the representation, provably. arXiv preprint arXiv:2002.09434, 2020.

[13] Yin Fang, Qiang Zhang, Haihong Yang, Xiang Zhuang, Shumin Deng, Wen Zhang, Ming Qin, Zhuo Chen, Xiaohui Fan, and Huajun Chen. Molecular contrastive learning with chemical element knowledge graph. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 36, pages 3968–3976, 2022.

[14] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, pages 3558–3565, 2019.

[15] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. Proceedings of The Web Conference 2020, Apr 2020.

[16] Amur Ghose, Yingxue Zhang, Jianye Hao, and Mark Coates. Spectral augmentations for graph contrastive learning. In International Conference on Artificial Intelligence and Statistics, pages 11213–11266. PMLR, 2023.

[17] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. arXiv preprint arXiv:1706.02216, 2017.

[18] Xiaotian Han, Zhimeng Jiang, Ninghao Liu, and Xia Hu. G-mixup: Graph data augmentation for graph classification. In International Conference on Machine Learning, pages 8230–8248. PMLR, 2022.

[19] Jeff Z HaoChen, Colin Wei, Adrien Gaidon, and Tengyu Ma. Provable guarantees for self-supervised deep learning with spectral contrastive loss. Advances in Neural Information Processing Systems, 34:5000–5011, 2021.

[20] Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. In International conference on machine learning, pages 4116–4126. PMLR, 2020.

[21] Binbin Hu, Yuan Fang, and Chuan Shi. Adversarial learning on heterogeneous information networks. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 120–129, 2019.

[22] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. arXiv preprint arXiv:1611.01144, 2016.

[23] Xunqiang Jiang, Yuanfu Lu, Yuan Fang, and Chuan Shi. Contrastive pre-training of gnns on heterogeneous graphs. In Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pages 803–812, 2021.

[24] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.

[25] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In ICLR. OpenReview.net, 2017.

[26] Kezhi Kong, Guohao Li, Mucong Ding, Zuxuan Wu, Chen Zhu, Bernard Ghanem, Gavin Taylor, and Tom Goldstein. Flag: Adversarial data augmentation for graph neural networks. arXiv preprint arXiv:2010.09891, 2020.

[27] O Deniz Kose and Yanning Shen. Fair node representation learning via adaptive data augmentation. arXiv preprint arXiv:2201.08549, 2022.

[28] Oyku Deniz Kose and Yanning Shen. Fair contrastive learning on graphs. IEEE Transactions on Signal and Information Processing over Networks, 8:475–488, 2022.

[29] Dongjin Lee and Kijung Shin. I'm me, we're us, and i'm us: Tri-directional contrastive learning on hypergraphs. arXiv preprint arXiv:2206.04739, 2022.

[30] Xiang Li, Danhao Ding, Ben Kao, Yizhou Sun, and Nikos Mamoulis. Leveraging meta-path contexts for classification in heterogeneous information networks. In 2021 IEEE 37th International Conference on Data Engineering (ICDE), pages 912–923. IEEE, 2021.

[31] Lu Lin, Jinghui Chen, and Hongning Wang. Spectral augmentation for self-supervised learning on graphs. arXiv preprint arXiv:2210.00643, 2022.

[32] Hongyi Ling, Zhimeng Jiang, Youzhi Luo, Shuiwang Ji, and Na Zou. Learning fair graph representations via automated data augmentations. In The Eleventh International Conference on Learning Representations, 2023.

[33] Shengchao Liu, Hongyu Guo, and Jian Tang. Molecular geometry pretraining with se (3)-invariant denoising distance matching. arXiv preprint arXiv:2206.13602, 2022.

[34] Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and Philip Yu. Graph self-supervised learning: A survey. IEEE Transactions on Knowledge and Data Engineering, 2022.

[35] Maria Marrium and Arif Mahmood. Data augmentation for graph data: Recent advancements. arXiv preprint arXiv:2208.11973, 2022.

[36] J Clyde Mitchell. Social networks. Annual review of anthropology, 3(1):279–299, 1974.

[37] Tom Michael Mitchell et al. Machine learning, volume 1. McGraw-hill New York, 2007.

[38] Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. arXiv preprint arXiv:2007.08663, 2020.

[39] Jiajie Peng, Yuxian Wang, Jiaojiao Guan, Jingyi Li, Ruijiang Han, Jianye Hao, Zhongyu Wei, and Xuequn Shang. An end-to-end heterogeneous graph representation learning-based framework for drug–target interaction prediction. Briefings in bioinformatics, 22(5):bbaa430, 2021.

[40] Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining, pages 1150–1160, 2020.

[41] Luana Ruiz, Luiz Chamon, and Alejandro Ribeiro. Graphon neural networks and the transferability of graph neural networks. Advances in Neural Information Processing Systems, 33:1702–1712, 2020.

[42] Yumeng Song, Yu Gu, Tianyi Li, Jianzhong Qi, Zhenghao Liu, Christian S Jensen, and Ge Yu. Chgnn: A semi-supervised contrastive hypergraph learning network. arXiv preprint arXiv:2303.06213, 2023.

[43] Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. arXiv preprint arXiv:1908.01000, 2019.

[44] Yizhou Sun and Jiawei Han. Mining heterogeneous information networks: principles and methodologies. Synthesis Lectures on Data Mining and Knowledge Discovery, 3(2):1–159, 2012.

[45] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. Proceedings of the VLDB Endowment, 4(11):992–1003, 2011.

[46] Susheel Suresh, Pan Li, Cong Hao, and Jennifer Neville. Adversarial graph augmentation to improve graph contrastive learning. Advances in Neural Information Processing Systems, 34:15920–15933, 2021.

[47] Puja Trivedi, Ekdeep S Lubana, Mark Heimann, Danai Koutra, and Jayaraman Thiagarajan. Analyzing data-centric properties for graph contrastive learning. Advances in Neural Information Processing Systems, 35:14030–14043, 2022.

[48] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. arXiv preprint arXiv:1710.10903, 2017.

[49] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks, 2018.

[50] Petar Velickovic, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. ICLR (Poster), 2(3):4, 2019.

[51] Vikas Verma, Meng Qu, Kenji Kawaguchi, Alex Lamb, Yoshua Bengio, Juho Kannala, and Jian Tang. Graphmix: Improved training of gnns for semi-supervised learning. In Proceedings of the AAAI conference on artificial intelligence, volume 35, pages 10024–10032, 2021.

[52] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In The World Wide Web Conference, pages 2022–2032, 2019.

[53] Xiao Wang, Nian Liu, Hui Han, and Chuan Shi. Self-supervised heterogeneous graph neural network with co-contrastive learning. In Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining, pages 1726–1736, 2021.

[54] Yuyang Wang, Rishikesh Magar, Chen Liang, and Amir Barati Farimani. Improving molecular contrastive learning via faulty negative mitigation and decomposed fragment contrast. Journal of Chemical Information and Modeling, 62(11):2713–2725, 2022.

[55] Rongzhe Wei, Haoteng Yin, Junteng Jia, Austin R Benson, and Pan Li. Understanding non-linearity in graph neural networks from the bayesian-inference perspective. arXiv preprint arXiv:2207.11311, 2022.

[56] Tianxin Wei, Yuning You, Tianlong Chen, Yang Shen, Jingrui He, and Zhangyang Wang. Augmentations in hypergraph contrastive learning: Fabricated and generative. arXiv preprint arXiv:2210.03801, 2022.

[57] Shiwen Wu, Wentao Zhang, Fei Sun, and Bin Cui. Graph Neural Networks in Recommender Systems: A Survey. arXiv:2011.02260 [cs], 2020.

[58] 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.

[59] Lianghao Xia, Chao Huang, Yong Xu, Jiashu Zhao, Dawei Yin, and Jimmy Xiangji Huang. Hypergraph contrastive collaborative filtering. arXiv preprint arXiv:2204.12200, 2022.

[60] Yaochen Xie, Sumeet Katariya, Xianfeng Tang, Edward Huang, Nikhil Rao, Karthik Subbian, and Shuiwang Ji. Task-agnostic graph explanations. arXiv preprint arXiv:2202.08335, 2022.

[61] Yaochen Xie, Zhao Xu, and Shuiwang Ji. Self-supervised representation learning via latent graph prediction. In International Conference on Machine Learning, pages 24460–24477. PMLR, 2022.

[62] 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.

[63] Dongkuan Xu, Wei Cheng, Dongsheng Luo, Haifeng Chen, and Xiang Zhang. Infogcl: Information-aware graph contrastive learning. Advances in Neural Information Processing Systems, 34:30414–30425, 2021.

[64] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? arXiv preprint arXiv:1810.00826, 2018.

[65] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. Hypergcn: A new method for training graph convolutional networks on hypergraphs. Advances in neural information processing systems, 32, 2019.

[66] Yihang Yin, Qingzhong Wang, Siyu Huang, Haoyi Xiong, and Xiang Zhang. Autogcl: Automated graph contrastive learning via learnable view generators. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 36, pages 8892–8900, 2022.

[67] Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. Graph contrastive learning automated. In International Conference on Machine Learning, pages 12121–12132. PMLR, 2021.

[68] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. Advances in neural information processing systems, 33:5812–5823, 2020.

[69] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. Bringing your own view: Graph contrastive learning without prefabricated data augmentations. In Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining, pages 1300–1309, 2022.

[70] Yuning You, Tianlong Chen, Zhangyang Wang, and Yang Shen. Graph domain adaptation via theory-grounded spectral regularization. In The Eleventh International Conference on Learning Representations, 2023.

[71] Yuning You and Yang Shen. Cross-modality and self-supervised protein embedding for compound–protein affinity and contact prediction. Bioinformatics, 38(Supplement_2):ii68–ii74, 2022.

[72] Yifei Zhang, Hao Zhu, Zixing Song, Piotr Koniusz, and Irwin King. Spectral feature augmentation for graph contrastive learning and beyond. arXiv preprint arXiv:2212.01026, 2022.

[73] Zuobai Zhang, Minghao Xu, Arian Jamasb, Vijil Chenthamarakshan, Aurelie Lozano, Payel Das, and Jian Tang. Protein representation learning by geometric structure pretraining. arXiv preprint arXiv:2203.06125, 2022.

[74] Jianan Zhao, Xiao Wang, Chuan Shi, Zekuan Liu, and Yanfang Ye. Network schema preserving heterogeneous information network embedding. In International Joint Conference on Artificial Intelligence (IJCAI), 2020.

[75] Jianan Zhao, Qianlong Wen, Shiyu Sun, Yanfang Ye, and Chuxu Zhang. Multi-view self-supervised heterogeneous graph embedding. In Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part II 21, pages 319–334. Springer, 2021.

[76] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. Data augmentation for graph neural networks. In Proceedings of the aaai conference on artificial intelligence, volume 35, pages 11015–11023, 2021.

[77] Jiajun Zhou, Jie Shen, and Qi Xuan. Data augmentation for graph classification. In Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pages 2341–2344, 2020.

[78] Yanqiao Zhu, Yichen Xu, Hejie Cui, Carl Yang, Qiang Liu, and Shu Wu. Structure-enhanced heterogeneous graph contrastive learning. In Proceedings of the 2022 SIAM International Conference on Data Mining (SDM), pages 82–90. SIAM, 2022.

[79] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep graph contrastive representation learning. arXiv preprint arXiv:2006.04131, 2020.

[80] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph contrastive learning with adaptive augmentation. In Proceedings of the Web Conference 2021, pages 2069–2080, 2021.

[81] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. Bioinformatics, 33(14):i190–i198, 2017.

Limitations of low dimensional graph embeddings

C. Seshadhri*

Abstract

The learning of graph representations, also called graph embeddings, is a fundamental technique in machine learning (ML). The aim is to represent the vertices of a graph by low-dimensional real-valued vectors, which can be used for a variety of downstream ML tasks. The popular technique of Graph Neural Networks (GNNs) can be thought of a type of graph representation learning method. This article surveys some recent work on the limitations of graph embedding methods. These results provide mathematical theorems proving that low-dimensional embeddings cannot recreate "community-like" structure, through commonly used kernels. These theorems are supported by empirical results, showing that many classic graph embedding methods actually perform poorly on important machine learning tasks. Low-dimensional representations often lose much of the fine-grained community structure of real-world data. The results surveyed in this article provide an interesting counterpoint to the popularity of graph embeddings and GNNs for machine learning.

1 Introduction

Capturing the rich structure of graph is central for many machine learning tasks. The heterogenous, non-local, and massive structure of modern graphs are a problem for many basic machine learning tasks, such as ranking, link prediction, and classification [14]. Graph representation learning provides a convenient solution to this problem. Each vertex of a graph is represented as a vector in low-dimensional space. These vectors form the (low-dimensional) embedding of the graph. The vectors can be used for a plethora of downstream machine learning tasks. The aim is to have these vectors combine the graph structure with vertex features into a compact geometric representation.

The study of low-dimensional graph embeddings is an incredibly popular research area, and has generated many exciting results over the past few years (see surveys [19, 8] and a Chapter 23 in [27]). The most successful methods for graph embeddings often use Deep Learning techniques, together with classic approaches like matrix factorization [29, 16, 28, 30, 31, 24]. Graph Neural Networks (GNNs) can be thought of as specific class of graph embeddings algorithms [18, 37, 40]. Despite the large variety of algorithms used for graph representation learning (and specifically GNNs), their output has a simple, consistent structure. Given a graph G on n vertices, these methods map each vertex to a vector in $\mathbb{R}^d$, where $d \ll n$. In a typical applications, n is the order of millions or more, while d is in the hundreds.

Much of the advances in this field are primarily empirical; there are numerous papers in major machine learning conferences on more complex GNN architectures or more involved graph embedding methods. Nonetheless, there is limited principled understanding of the power of low-dimensional graph embeddings. Small changes in training or input data can lead to major differences in the output [17]. Much of the research on graph embeddings and GNNs, for example, report significant success in prediction tasks on graphs [16, 18, 31]. On the other hand, some papers suggest that low-dimensional graph embeddings can be beaten by simpler hand-tuned methods [17, 22]. It is useful to have a rigorous mathematical framework to understand graph embeddings. Specially, we wish to address the following questions.

*University of California, Santa Cruz. sesh@ucsc.edu. Supported by NSF DMS-2023495, CCF-1740850, 1839317, 1908384

- To what extent can relevant graph structure be captured by low-dimensional graph embeddings?
- How does low-dimensional geometry relate to downstream ML tasks on graphs?
- How does the choice of kernel functions (for prediction tasks) affect the task?

Contrary to most research in this area, our approach to investigating these questions is through limitations or lower bounds. Our aim is develop a theoretical framework that is agnostic to the specific algorithm performing the embedding. We first describe some basic terminology and central concepts.

The importance of matrix factorizations: A low-dimensional matrix factorization approximates gives an algorithm-independent view of how low-dimensional embeddings represent graphs. We think of M as an adjacency matrix, or any matrix representation of our graph G. In many Deep Learning methods, M is constructed by long walks with appropriate vertex-centric aggregation functions [29, 16, 30]. The matrix V is the collection of embeddings, where each column vector corresponds to a vertex. Recent work has shown the many different graph embedding methods can be cast as matrix factorizations, with different choices of matrices M, and different notions of approximations [31]. Indeed, the simplest low-dimensional embedding is obtained by a low-rank SVD of the adjacency matrix, arguably the most basic of matrix factorizations.

The importance of dot products: Regardless of how the graph embedding vectors are obtained, the general strategy is to use "nearness" in geometric space as a proxy for the similarity of the corresponding vectors. Thus, the downstream ML tasks treat vertices i and j as similar if their corresponding vectors $\vec{v}_i$ and $\vec{v}_j$ are similar. Arguably, the most important measure of similarity is the dot product $\vec{v}_i \cdot \vec{v}_j$ or cosine similarity (which is a normalized dot product). This has special significance for matrix factorizations, since $V^T V$ is precisely the matrix of all pairs of dot products.

Even when the embedding vectors are not obtained by matrix factorizations, the Gram matrix $V^T V$ is commonly used for downstream prediction tasks. A natural strategy for any prediction task is to use nearest neighbor (k-NN) search according to the dot product/cosine similarity. There are many open course packages for k-NN, making it the default choice in industrial applications of graph embeddings [2].

We now can formalize our initial questions in matrix language. Observe how there is no reference to any embedding method or GNN; we directly analyze the behavior of the output representation.

Can a Gram matrix $V^T V$ for $V \in \mathbb{R}^{d \times n}$ ($d \ll n$) capture the structure of either real-world networks or the properties of prediction matrices arising from downstream ML tasks (on graphs)?

This framework captures the vast majority of graph embedding constructions and applications.

1.1 Limitations of graph embeddings

This article presents two results on the limitations of low-dimensional graph embeddings for prediction and machine learning on real-world graphs [34, 36]. The underlying theoretical results are algorithm agnostic, and the limitations hold for any graph embedding algorithm. These results come with empirical backing, performed on classic and important graph embedding methods.

Inability to recreate triangle structure [34]: This result focuses on the fundamental premise of low-dimensional embeddings, rather than a specific ML task. Algorithms to construct (or predict from) low-dimensional embeddings implicitly pose a low-dimensional graph/matrix model. This model represents a distribution of graphs that are created from a set of real vectors (where each vector represents a vertex). The embedding is constructed by fitting this model to an input graph, typically using dot product as a proximity measure.

This result mathematically proves that no low-dimensional model (using a dot product kernel) can simultaneously recreate two central properties of real-world graphs: sparsity and triangle density. It is well-known from the early days of network science that real-world graphs are sparse but have high clustering coefficients [39, 32, 33, 13]. Hence, typical models used to construct low-dimensional embeddings cannot generate realistic graphs. They miss the (crucial) triangle structure of real data.

The theory is supported with empirical results. These empirical results are demonstrated on other kernels beyond the dot product, thus suggesting that the limitations are fundamental. We discuss these results in §2.

There are notable counterpoints to these results. Firstly, Chanpuriya et al show that asymmetric embeddings avoid the rank lower bounds discussed above [9]. Another result of Chanpuriya et al shows that graphs can sometimes be reconstructed from their embeddings. In some cases, the community structure of the reconstruction is actually enhanced [10]. We give more detail in §2.

Weak performance on community labeling tasks [36]: This result takes a complementary angle, and focuses on a downstream ML task. Community labeling is a binary prediction problem, where we wish to predict if two vertices belong to a community. This work sets up a community labeling problem for both real and synthetic data sets. First, the authors create a simple benchmark algorithm using logistic regression on classic graph features (like Personalized PageRank and short path counts). This benchmark has fairly good performance, as measured by distributions of local precision. On the other hand, many well-established graph embedding methods have surprisingly poor performance on the same task.

These observations are backed up with theoretical proofs. The prediction matrices based on low-dimensional embeddings provably do not have the typical community structure of real data. This work also investigates alternate kernels, like the normalized softmax (used in results like `DeepWalk` and `node2vec` [29, 16]). These kernels have other limitations in that slight noise can destroy community structure in the corresponding prediction matrices.

These results are discussed in §3.

1.2 Broader context

A reader may ask: how are the limitations stated in this article consistent with large body of work on the effectiveness of graph embeddings and GNNs? Our answer is two-fold, backed up by [17, 22]. First, most research on graph embeddings compare various representation learning methods with each other, and do not consider alternative baselines. Secondly, we believe that many graph embeddings methods do not have good predictions with respect to other metrics. For example, almost all these result use the AUC metric to measure link prediction performance. On the other hand, AUC is a bad measure for sparse ground truth [20, 25]. Indeed, in §3, we show poor prediction performance (for graph embedding methods) on a local precision metric.

There has been compelling empirical work showing that GNNs and embeddings can be outperformed by simpler methods. We mention two results in detail because they highlight empirical weaknesses in graph embeddings, and reinforce the previous points.

Gurukar et al, the lack of good experimental design [17]: Gurukar et al do a detailed comparison of twelve different graph representation learning methods on a variety of ML tasks. They focus on two of the most important tasks of link prediction and node classification. Despite there being many newer methods, they observe that the `M-NMF` algorithm is best for link prediction [38] and `NetMF` algorithm [31] is best for node classification. No graph representation method outperforms on both metrics. This paper does an exceptional job of clearly specifying the experimental design and thoroughly investigating previous work.

Along the lines of the current article, simple task specific baselines are competitive with graph embedding methods. These baselines are formed by a simple model that uses basic graph features (like Common Neighbors, Adamic-Adar index [4], Jaccard similarity, etc.). These results are analogous to what we observe in [36].

Gurukar et al point out a major problem with evaluations in graph embedding papers. Quoting from there: ". . . a new method almost always compares its performance against a subset of other methods [and] datasets previously evaluated. While great care is taken to tune the new method, the same care is often not taken when evaluating baselines."

Huang et al, easier methods beat GNNs [22]: Huang et al devise an alternate graph learning algorithm called Correct and Smooth (C&S). Consider the problem of node classification. This method starts with a "base predictor" that ignores the graph structure. Using only node features, one can train a basic model for classification. Then, the graph is introduced a post-processing step to "correct and smooth" the errors. The idea is that the errors should be correlated along edges. So this method tries to find a smooth error estimation on the graph, which is computed using standard label propagation methods. This smoothed error estimate is used to correct the base predictor.

The C&S method is shown to outperform GNNs on node classification tasks on the OGB leaderboard [1]. In some cases, a state-of-the-art GNN has slightly better performance, but the GNNs always have many orders of magnitude more parameters. Hence, it is much more expensive to train them than C&S.

We also mention another line of weaknesses, specific to message passing GNN architectures. These results show that certain graph-theoretic properties cannot be learned by common GNN architectures [40, 26, 15]. There are formal limits on what GNN-based representations can learn. Some of these results show that message passing GNNs are no more powerful than the classic Weisfeiler-Lehman (WL) isomorphism test [40].

The work presented in the current article is different from these results because of the (algorithm agnostic) focus on the geometry of graph representations.

2 Impossibility of capturing sparse, triangle-rich structure

The first result we present argues that low-dimensional embeddings with dot product geometries are not good representation of real-world graphs. Seshadhri, Sharma, Stolman, and Goel demonstrate mathematically and empirically that they lose local cluster structure, a central aspect of graphs that arise from real data [34].

Graph embeddings are often generated by assuming that the embeddings vectors lie in a hidden, latent space. We assume a model that generates graphs from these vectors, which can be thought of as the model parameters. The embeddings are constructed by optimizing for these parameters, given the input graph.

Consider the graph embedding vectors $\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_n \in \mathbb{R}^d$ (denoted by the $d \times n$ matrix V). Let $\mathcal{G}_V$ denote the following distribution of graphs over the vertex set $[n]$. For each index pair i, j, independently insert (undirected) edge (i, j) with probability $\max(0, \min(\vec{v}_i \cdot \vec{v}_j, 1))$. (If $\vec{v}_i \cdot \vec{v}_j$ is negative, (i, j) is never inserted. If $\vec{v}_i \cdot \vec{v}_j \geq 1$, (i, j) is always inserted.) This model subsumes the classic Stochastic Block Model [21] and Random Dot Product Model [42, 6]. Many graph embedding methods, including GNNs, effectively optimize over such a model to generate the embedding vectors.

Two hallmarks of real-world graphs are: (i) Sparsity: average degree is constant with respect to n, and (ii) Triangle density: there are many triangles incident to low degree vertices [39, 32, 33, 13]. The large number of triangles is an important aspect of community structure.

Definition 2.1: For parameters $c > 1$ and $\Delta > 0$, a graph G with n vertices has a (c, Δ)-triangle foundation if there are at least Δn triangles contained among vertices of degree at most c.

Typically, we think of both c and Δ as constants. We emphasize that n is the total number of vertices in G, not the number of vertices in S. In Figure 1, we plot the value of c vs Δ as the thick blue line. (Specifically, the y axis is the number of triangles divided by n.) Observe that for all graphs, for $c \in [10, 50]$, we get a value of $\Delta > 1$ (in many cases $\Delta > 10$). As mentioned earlier, there is much work in network science showing that there are often a linear number of triangles among low degree vertices [33, 13].

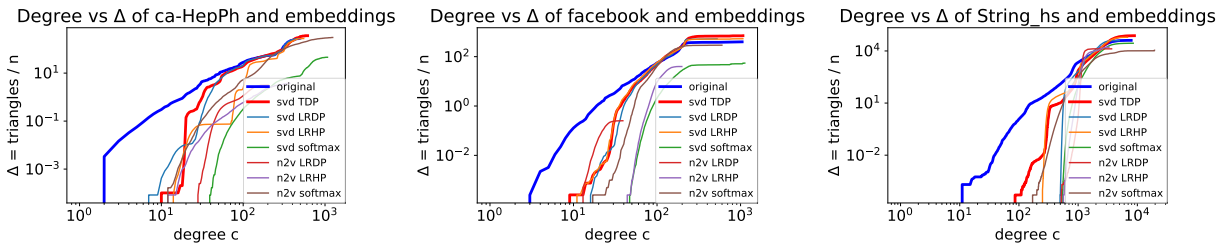


Figure 1: Plots of degree c vs Δ: We plot results for a variety of real-world graphs: (i) `ca-HepPh`, a High Energy Physics coauthorship network (ii) `Facebook`, a small snapshot of a Facebook social network, (iii) `String_hs`, a Protein-protein interaction network. We plot c versus the total number of triangles only involving vertices of degree at most c. We divide the latter by the total number of vertices n, so it corresponds to Δ, as in Definition 2.1. We plot these both for the original graph (in thick blue), and for a variety of embeddings and kernel functions. For each embedding, we plot the maximum Δ in a set of 100 samples from a 100-dimensional embedding. The embedding analyzed by Theorem 2.1 (TDP) is given in thick red. Observe how the embeddings generate graphs with very few triangles among low degree vertices. The gap in Δ for low degree is 2-3 orders of magnitude. The other lines correspond to alternate embeddings, using the NODE2VEC vectors and/or different functions of the dot product.

Our main result is that any embedding of graphs that generates graphs with (c, Δ)-triangle foundations, with constant c, Δ, must have near linear rank. This contradicts the belief that low-dimensional embeddings capture the structure of real-world complex networks.

Theorem 2.1: Fix constant $c > 4, \Delta > 0$. Suppose the expected number of triangles in $G \sim \mathcal{G}_V$ that only involve vertices of expected degree c is at least Δn. Then, the rank of V is at least $\Omega(n/\lg^2 n)$.

Equivalently, graphs generated from low-dimensional embeddings cannot contain many triangles only on low-degree vertices. In all applications, d is thought of as a constant, or at least much smaller than n. On the contrary, Theorem 2.1 implies that d must be $\Omega(n/\lg^2 n)$ to accurately model the low-degree triangle behavior. This lower bound holds regardless of how the vectors are constructed.

2.1 Empirical validation

We empirically validate the theory on a collection of complex networks. For each real-world graph, we compute a 100-dimensional embedding through SVD and an important Deep Learning method, `node2vec` [16]. We generate 100 samples of graphs from these embeddings, and compute their c vs Δ plot. This is plotted with the true c vs Δ plot. (To account for statistical variation, we plot the maximum value of Δ observed in the samples, over all graphs. The variation observed was negligible.) Fig. 1 shows such a plot for three different real-world networks.

In all cases, this plot is significantly off the mark at low degrees for the embedding. Around the lowest degree, the value of Δ (for the graphs generated by the embedding) is 2-3 orders of magnitude smaller than the original value. The local triangle structure is destroyed around low degree vertices. The total number of triangles is preserved well, as shown towards the right side of each plot. A nuanced view of the triangle distribution, as given in Definition 2.1, is required to see the shortcomings of low dimensional embeddings.

We note that several other functions of dot product have been proposed in the literature, such as the softmax function [29, 16] and linear models of the dot product [18]. Theorem 2.1 does not have direct implications for such models, but our empirical validation holds for them as well. The embedding in Theorem 2.1 uses the truncated dot product (TDP) function $max(0, \min(\vec{v}_i \cdot \vec{v}_j, 1))$ to model edge probabilities. We construct other embeddings that compute edge probabilities using machine learning models with the dot product and Hadamard

product as features. This subsumes linear models as given in [18]. We also consider (scaled) softmax functions, as in [29], and standard machine learning models (LRDP, LRHP).

For each of these models, we perform the same experiment described above. Fig. 1 also shows the plots for these other models. Observe that none of them capture the low-degree triangle structure, and their Δ values are all 2-3 orders of magnitude lower than the original. Chanpuriya et al also validate these results on various other embedding methods [9].

2.2 Counterpoints

We discuss two important results of Chanpuriya, Musco, Sotiropoulos, and Tsourakakis [9, 10]. The first result show that the rank lower bound of Theorem 2.1 can be circumvented with an asymmetric embedding. They prove that one can construct two matrices $U, V \in \mathbb{R}^{d \times n}$ such that the graph distribution from UV^T can generate realistic triangle structure. They show that any bounded degree graph can be embedded in this method with at most max-degree dimensions. This introduces a new technique for graph embeddings. We note, however, that such asymmetric embeddings lose the geometric structure of the standard embeddings. One wants geometric proximity of vectors to represent structural closeness. For vertices i and j, an asymmetric embedding approximates the edge probability by $\vec{u}_i \cdot \vec{v}_j$, but the similarity of i and j would be measured by looking at (say) $\vec{u}_i$ and $\vec{u}_j$. It would be interesting to incorporate similarity in asymmetric embeddings.

Another result shows that, in some cases, community structure can be reconstructed from `DeepWalk` embeddings [10]. This shows that some structure is being retained by the embeddings. We note that these results mostly focus on SBMs with a constant (at most 5) blocks, or only the largest few communities in real data. Theorem 2.1 and the other results in this article focus on cases where the number of blocks/communities is large. We believe that low-dimensional embeddings can recover the top few communities, but fail to capture the rich structure of many small communities. In the next section, we discuss this point further.

3 Challenges for community labeling using graph embeddings

One central promise of unsupervised graph embedding methods is to preserve network structure in the geometry. To what extent do embedding methods capture graph structure relevant to downstream ML tasks?

This section is based on the paper of Stolman, Levy, Seshadhri, and Sharma [36]. We begin this discussion with the empirical results, because they highlight the core observation. After that, we will go into the mathematical explanations that relate to low-dimensional embeddings and factorization.

Consider the following well-defined pairwise community labeling problem. Given two vertices i and j, the binary classification task is to determine whether they belong to the same community. We note that this community labeling problem is an instance of a broad range of community detection problems that have a long history of study in the graph mining literature [23].

3.1 Empirical setup

We use a set of real-world datasets with ground truth community labels, an Amazon co-purchase graph of products and a DBLP citation network [41]. We also create a synthetic Stochastic Block model, with 100K vertices, and small blocks of size 20 each. There is a dense graph within each block, and a random sparse graph connecting all the blocks.

As explained earlier, the prediction task is to determine if an input pair i, j of vertices belong to a community. (They may belong to multiple communities; to make the problem simpler, we do not require any community labels to be determined.)

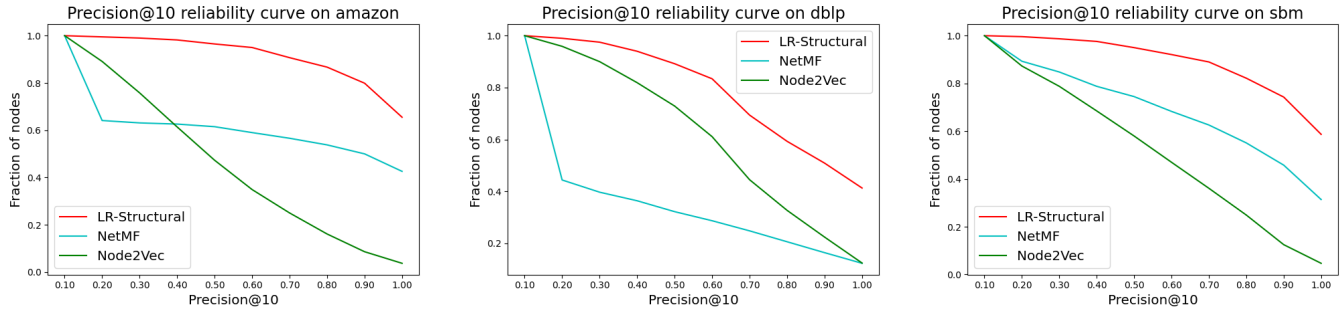


Figure 2: Each point, (x, y), on the curve represents the approximate fraction of vertices, y, for which the given method produces a precision@10 score of at least x. `LR-Structural` is plotted against the two best performing embedding methods. 1000 vertices are sampled and for each vertex v sampled, the vertices of the graph $u_1, \ldots, u_n$, are ordered by decreasing score assigned by the given classifier. The precision@10 is the fraction of $u_1, \ldots, u_{10}$ which share a community with v. Across all instances, the simple baseline `LR-Structural` handily outperforms more complex graph embedding methods.

The setup for graph embeddings: We experiment with a set of important graph embedding methods based on factorizations and Deep Learning (`GraRep, DeepWalk, node2vec, NetMF` [29, 7, 28, 16, 31]). We note that the `NetMF` method was reported to be one of the best embedding methods for node classification [17]. Given the embedding for each vertex, we need to construct pairwise features for pairs in $(i, j) \in V \times V$, for the community prediction task. The standard approach is to either take the dot product $\vec{v}_i \cdot \vec{v}_j$ or the Hadamard product $\vec{v}_i \circ \vec{v}_j$. (Recall that the Hadamard product is a d-dimensional vector whose rth coordinate is the product of the rth coordinates of $\vec{v}_i$ and $\vec{v}_j$.) We finally train a logistic regression model on these features for the prediction problem. This is the standard pipeline used in prior work [11, 16, 8].

A simple baseline: We compute four basic structural graph features for pairs of vertices (i, j). We look at the cosine similarity and cut size between neighborhoods, and the Personalized PageRank values. These are well-known classic features used in the literature [35, 5]. We train a simple logistic regression model on these four features; the model is denoted `LR-Structural` .

Performance metric: For both real and simulated data, the ground truth is sparse, i.e. the vast majority of node pairs do not belong to the same community. We do not use AUC because of its problems in measuring sparse data [20, 25]. Instead, it is appropriate to measure the prediction performance using precision-recall curves for this highly imbalanced label distribution [12].

The methods are evaluated by comparing the "precision@10" distributions. We sample 1000 random vertices. For each of 1000 vertices sampled, v, we order the other vertices of the graph, $u_1, \ldots, u_n$, in decreasing order of their prediction score. (The model predictors based on logistic regression on the embedding vectors or structural features assign a score in $[0, 1]$.) We compute the precision, per vertex, of the classifier among the top 10 scores, with respect to the ground truth. When the predictor is based on dot product, this is simply the top 10 neighbors in geometric space. In other words, we sample a vertex at random and report the fraction of its ten nearest neighbors with which it shares a community.

We represent the distribution of values of precision@10 scores as a reliability curve. This is the curve (x, y) such that at least a y fraction of vertices sampled had a precision@10 score score of at least x. Higher y values for a given x indicate better performance. Fig. 2 contains the curves for the best methods against the baseline (we leave out methods with poorer performance).

Main observation: Across all instances, the baseline `LR-Structural` method heavily outperforms the more complex graph embedding methods. The performance gap for the simple Stochastic Block Model instance is striking. The graph is practically learnable (just a collection of dense blocks with sparse connections), but the embedding methods fail to accurately predict pairs within blocks. For the `LR-Structural` method, in all instances, at least 90% of the vertices have a precision@10 of at least 0.5. This means, for at least 90% of the vertices, at least five of the top 10 scores are in the same community. By contrast, for the embedding methods, this fraction is less than 75%. When looking for a precision of more than 0.8, the embeddings methods have less than 20% of vertices achieving high scores. Overall, we observe that the embeddings methods do not perform the community labeling task well, despite the simple `LR-Structural` baseline having good precision values.

3.2 Theoretical explanation of limitations

As explained earlier, a large variety of graph embedding methods (including those using Deep Learning) implicitly factorize a matrix M as $V^T V$. Here, M denotes some matrix representing the input graph data or the final prediction, and $V \in \mathbb{R}^{d \times n}$ is the matrix of graph embeddings. Broadly speaking, we can classify these methods into two categories:

- **Direct factorizations:** Here, we set V as $\operatorname{argmin}_V \|V^T V - M\|_2$, where M is typically (some power of) the graph adjacency matrix. Methods such as Graph Factorization, GraRep [7], and HOPE [28] would fall under this category.
- **Softmax factorizations:** These methods factorize a stochastic matrix, such as (powers of) the random walk matrix. (A stochastic matrix has row sums equal to one.) Since $V^T V$ is not necessarily stochastic, these methods apply the softmax to generate a stochastic matrix. Notable examples are such methods are DeepWalk [29] and Node2vec [16]. Formally, consider the normalized softmax matrix $\operatorname{nsm}(V)$ given by

$$\operatorname{nsm}(V)_{ij} = \frac{\exp(\vec{v}_i \cdot \vec{v}_j)}{\sum_k \exp(\vec{v}_i \cdot \vec{v}_k)} \tag{28}$$

Note that $\operatorname{nsm}(V)$ is stochastic by construction.

The `NetMF` [31] method interpolates between these categories and shows that a number of existing methods can be expressed as factorization methods, especially of the above forms.

The notion of community pairs: We start with an abstraction of community structure from a matrix standpoint: many dense blocks in an overall sparse matrix. We quantify "how much" community structure can be present in a matrix $V^T V$ or $\operatorname{nsm}(V)$, for any matrix $V \in \mathbb{R}^{d \times n}$ (for $d \ll n$). This formulation captures the fundamental notion of a low-dimensional embedding, without referring to any specific method to compute it.

Let us start with an $n \times n$ matrix M that represents the "similarity" or likelihood of connection between vertices. This is the final prediction matrix for community labeling. For convenience, let us normalize so that the $\forall i \in [n], \sum_{j \leq n} M_{i,j} \leq 1$. (So the sum of similarities of a vertex is at most 1.) A communities is essentially a dense block of entries, which motivates the following definition. We use ε to denote a parameter for the threshold of community strength. One should think of ε as a small constant, or something slowly decreasing in n (like $1/\operatorname{poly}(\log n)$).

Definition 3.1: A pair of vertices (i, j) is a potential community pair if both M_{ij} and M_{ji} are at least ε.

Note that we do not expect all such pairs (i, j) to truly be together in a community. Hence, we only consider such a pair a potential candidate. We expect community relationships to be mutual, even if the matrix M is not. A community can be thought of as a submatrix where at least a constant fraction of pairs are potential community pairs. It is natural to expect that $\Theta(n)$ pairs are community pairs; indeed, most vertices should participate in communities, and will have at least a constant number of community neighbors. Our mathematical analyses shows that direct and softmax factorizations cannot produce these many potential community pairs.

Lower bound for direct factorizations: We prove that the number of potential community pairs in V^TV is linear in the rank, and thus, a low-dimensional factorization cannot capture community structure. The proof uses the rotational invariance of Frobenius norms.

Theorem 3.1: Consider any matrix $V \in \mathbb{R}^{d\times n}$ such that row sums in V^TV have absolute value at most 1. Then V has at most $d/2\varepsilon^2$ potential community pairs.

Proof: Since V^TV has row sums of absolute value at most 1, the largest absolute value of eigenvalue is also at most 1 (a consequence of the Gershgorin circle theorem [3].). The rank of V^TV is at most d, so V^TV has at most d non-zero eigenvalues. We can express the Frobenius norm squared, $\|V^TV\|_2^2$, by the sums of squares of eigenvalues. By the arguments above, $\|V^TV\|_2^2 \leq d$.

But the Frobenius norm squared $\|V^TV\|_2^2$ is also the sums of squares of entries. Each potential community pair contributes at least $2\varepsilon^2$ to this sum. Hence, there can be at most $d/2\varepsilon^2$ potential community pairs.

The instability of softmax factorizations: The properties of softmax factorizations are more nuanced. Firstly, we can prove that softmax factorizations can represent community structure quite effectively.

Theorem 3.2: For $d = O(\log n)$, there exists $V \in \mathbb{R}^{d\times n}$ such that $\mathrm{nsm}(V)_{ij}$ exhibits community structure. Specifically, for any natural number $b \leq n$, there exists $V \in \mathbb{R}^{d\times n}$ such that $\mathrm{nsm}(V)$ has n/b blocks of size b, such that all entries within blocks are at least $1/2b$.

Indeed, this covers the various SBM settings we study, and demonstrates the superiority of softmax factorizations for modeling community structure. We note that a similar theorem, for assymmetric factorizations, was proved in [9].

On the other hand, these factorizations are highly unstable to small perturbations. Indeed, with a tiny amount of noise, any community pair can be destroyed with high probability. The noise model scales each vector with small $(1 \pm \delta)$ Gaussian noise to get the matrix $\mathrm{nsm}(\widetilde{V}^{(\delta)})$. (The formal definition is given in [36].)

Theorem 3.3: Let c denote some absolute positive constant. Consider any $V \in \mathbb{R}^{d\times n}$. For any $\delta > c\ln(1/\varepsilon)/\ln n$, the following holds in $\mathrm{nsm}(\widetilde{V}^{(\delta)})$ (this is the matrix formed by $\mathrm{nsm}(V)$ with δ Gaussian noise). For at least $0.98n$ vertices i, for any pair (i, j), the pair is not a potential community pair with probability at least 0.99.

Thus, with overwhelming probability, any community structure in $\mathrm{nsm}(V)$ is destroyed by adding $o(1)$ (asymptotic) noise. This is strong evidence that either noise in the input or numerical precision in the final optimization lead to destruction of community structure. These theorems give an explanation of the poor performance of the embeddings.

4 Conclusion

Instead of interpreting these limitations pessimistically, we reiterate the need for rigorous, foundational work in graph embeddings and GNNs. The work in this article merely scratches the surface. The limitations given in [34, 36] might not hold for all low-dimensional embedding methods, but they cover a large class of them. The limitations certainly hold for the most popular methods used, and is reinforced by the empirical results. The counterpoints of [9, 10] lead to a more nuanced picture for specialized embedding methods. We need a deeper understanding of how limitations can be avoided and how they relate to the downstream ML tasks.

The limitations question a purely empirical approach of designing better and better embedding methods and GNNs. As [17] correctly point out, each method comes with many hyperparameters, so it might be possible to tune one method to beat another and vice versa. Small improvements on some test datasets might not reveal the

complete picture. The theoretical and mathematical framework discussed in this article provide a more rigorous basis for research. If there are fundamental limitations from low-dimensional geometry for certain methods, we should not try to "tune" the problems away by experimenting with hyperparameters.

Overall, we believe that the work surveyed in this article provide an exciting new research perspective for graph embeddings and GNNs.

References

[1] Ogb leaderboards. https://ogb.stanford.edu/docs/lsc/leaderboards/.

[2] Scikit-learn: Nearest neighbors. https://scikit-learn.org/stable/modules/neighbors.html.

[3] Gershgorin circle theorem. https://en.wikipedia.org/wiki/Gershgorin_circle_theorem, 2020.

[4] L. Adamic and E. Adar. Friends and neighbors on the web. Social Networks, 25(3), 2003.

[5] Reid Andersen, Fan Chung, and Kevin Lang. Using pagerank to locally partition a graph. Internet Mathematics, 4:35–64, 2007.

[6] Avanti Athreya, Donniell E. Fishkind, Keith Levin, Vince Lyzinski, Youngser Park, Yichen Qin, Daniel L. Sussman, Minh Tang, Joshua T. Vogelstein, and Carey E. Priebe. Statistical inference on random dot product graphs: a survey. Journal of Machine Learning Research, 18:1–92, 2018.

[7] Shaosheng Cao, Wei Lu, and Qiongkai Xu. GraRep: Learning graph representations with global structural information. In Conference on Information and Knowledge Management (CIKM), pages 891–900, 2015.

[8] Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. Machine learning on graphs: A model and comprehensive taxonomy. arXiv:2005.03675, 2020.

[9] Sudhanshu Chanpuriya, Cameron Musco, Konstantinos Sotiropoulos, and Charalampos E. Tsourakakis. Node embeddings and exact low-rank representations of complex networks. In Neural Information Processing Systems (NeurIPS), 2020.

[10] Sudhanshu Chanpuriya, Cameron Musco, Konstantinos Sotiropoulos, and Charalampos E. Tsourakakis. Deepwalking backwards: From embeddings back to graphs. In International Conference on Machine Learning (ICML), volume 139, pages 1473–1483, 2021.

[11] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In Conference on Recommender Systems (RecSys), pages 191–198, 2016.

[12] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In International Conference on Machine Learning (ICML), pages 233–240, 2006.

[13] N. Durak, A. Pinar, T. G. Kolda, and C. Seshadhri. Degree relations of triangles in real-world networks and graph models. In Conference on Information and Knowledge Management (CIKM), pages 1712–1716, 2012.

[14] David Easley and Jon Kleinberg. Networks, crowds, and markets, volume 8. Cambridge university press Cambridge, 2010.

[15] Vikas K. Garg, Stefanie Jegelka, and Tommi S. Jaakkola. Generalization and representational limits of graph neural networks. In International Conference on Machine Learning (ICML), volume 119, pages 3419–3430, 2020.

[16] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Conference on Knowledge Discovery and Data Mining (KDD), pages 855–864, 2016.

[17] Saket Gurukar, Priyesh Vijayan, Aakash Srinivasan, Goonmeet Bajaj, Chen Cai, Moniba Keymanesh, Saravana Kumar, Pranav Maneriker, Anasua Mitra, Vedang Patel, Balaraman Ravindran, and Srinivasan Parthasarathy. Network representation learning: Consolidation and renewed bearing. arXiv, abs/1905.00987, 2019.

[18] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In Neural Information Processing Systems (NeurIPS), pages 1024–1034, 2017.

[19] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. IEEE Data Eng. Bull., 40(3):52–74, 2017.

[20] David J. Hand. Measuring classifier performance: a coherent alternative to the area under the ROC curve. Machine Learning, 77(1):103–123, Oct 2009.

[21] P. W. Holland, K. Laskey, and S. Leinhardt. Stochastic blockmodels: First steps. Social Networks, 5:109–137, 1983.

[22] 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 International Conference on Learning Representations (ICLR), 2021.

[23] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. Mining of massive data sets. Cambridge university press, 2020.

[24] Jundong Li, Liang Wu, and Huan Liu. Multi-level network embedding with boosted low-rank matrix approximation. In Advances in Social Networks Analysis and Mining, pages 49–56, 2019.

[25] Ryan Lichtenwalter and Nitesh V. Chawla. Link prediction: Fair and effective evaluation. In Advances in Social Networks Analysis and Mining, pages 376–383, 2012.

[26] Andreas Loukas. What graph neural networks cannot learn: depth vs width. In International Conference on Learning Representations (ICLR), 2020.

[27] Kevin P. Murphy. Probabilistic Machine Learning: An introduction. MIT Press, 2021.

[28] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In Conference on Knowledge Discovery and Data Mining (KDD), pages 1105–1114, 2016.

[29] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online learning of social representations. In Conference on Knowledge Discovery and Data Mining (KDD), pages 701–710, 2014.

[30] Bryan Perozzi, Vivek Kulkarni, Haochen Chen, and Steven Skiena. Don't walk, skip! online learning of multi-scale network embeddings. In Advances in Social Networks Analysis and Mining, page 258–265, 2017.

[31] 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 Conference on Web Science and Data Mining (WSDM), pages 459–467, 2018.

[32] A. Sala, L. Cao, C. Wilson, R. Zablit, H. Zheng, and B. Y. Zhao. Measurement-calibrated graph models for social network experiments. In Conference on the World Wide Web (WWW), pages 861–870, 2010.

[33] C. Seshadhri, Tamara G. Kolda, and Ali Pinar. Community structure and scale-free collections of Erdös-Rényi graphs. Physical Review E, 85(5):056109, 2012.

[34] C. Seshadhri, Aneesh Sharma, Andrew Stolman, and Ashish Goel. The impossibility of low-rank representations for triangle-rich complex networks. Proceedings of the National Academy of Sciences, 117(11):5631–5637, 2020.

[35] Aneesh Sharma, C. Seshadhri, and Ashish Goel. When hashes met wedges: A distributed algorithm for finding high similarity vectors. In Conference on the World Wide Web (WWW), pages 431–440, 2017.

[36] Andrew Stolman, Caleb Levy, C. Seshadhri, and Aneesh Sharma. Classic graph structural features outperform factorization-based graph embedding methods on community labeling. In SIAM Conference on Data Mining (SDM), pages 388–396, 2022.

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

[38] Xiao Wang, Peng Cui, Jing Wang, Jian Pei, Wenwu Zhu, and Shiqiang Yang. Community preserving network embedding. AAAI Conference on Artificial Intelligence, 31(1), 2017.

[39] D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. Nature, 393:440–442, 1998.

[40] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In International Conference on Learning Representations (ICLR)International Conference on Learning Representations (ICLR), 2019.

[41] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In SIGKDD Workshop on Mining Data Semantics, 2012.

[42] Stephen J. Young and Edward R. Scheinerman. Random dot product graph models for social networks. In Algorithms and Models for the Web-Graph, pages 138–149, 2007.

Customized Graph Nerual Networks

Yiqi Wang[1*], Yao Ma[2*], Wei Jin[1], Chaozhuo Li[3], Charu Aggarwal[4], Jiliang Tang[1]
[1] Michigan State University
[2] New Jersey Institute of Technology
[3] Microsoft Research Asia
[4] IBM T. J. Watson Research Center
{wangy206,mayao4,jinwei2,tangjili}@msu.edu, {cli}@microsoft.com, {charu}@us.ibm.com

Abstract

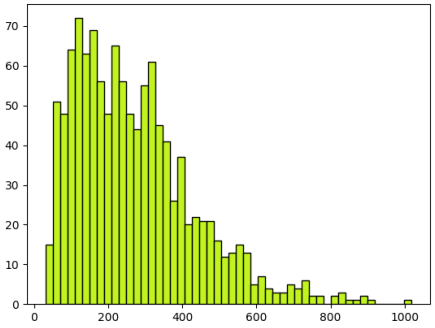
Recently, Graph Neural Networks (GNNs) have greatly advanced the task of graph classification. Typically, we first build a unified GNN model with graphs in a given training set and then use this unified model to predict labels of all the unseen graphs in the test set. However, graphs in the same dataset often have dramatically distinct structures, which indicates that a unified model may be sub-optimal given an individual graph. Therefore, in this paper, we aim to develop customized graph neural networks for graph classification. Specifically, we propose a novel customized graph neural network framework, i.e., Customized-GNN. Given a graph sample, Customized-GNN can generate a sample-specific model for this graph based on its structure. Meanwhile, the proposed framework is very general that can be applied to numerous existing graph neural network models. Comprehensive experiments on various graph classification benchmarks demonstrate the effectiveness of the proposed framework.

1 Introduction

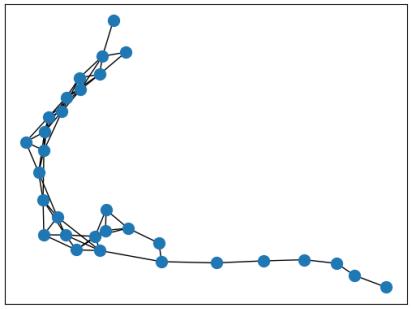
Graphs are natural representations for many real-world data such as social networks [1, 2, 3, 4], biological networks [5, 6] and chemical molecules [7, 8]. A crucial step to perform downstream tasks on graph data is to learn better representations. Deep neural networks have demonstrated great capabilities in representation learning for Euclidean data and thus have advanced numerous fields including speech recognition [9], computer vision [10] and natural language processing [11]. However, they cannot be directly applied to graph-structured data since graphs have complex topological structures. Recently, graph neural networks (GNNs) have generalized deep neural networks to graph data. GNNs typically update node representations by transforming, propagating and aggregating node features across the graph. They have boosted the performance of many graph related tasks such as node classification [3, 2], link prediction [12, 13, 14], and graph classification [15, 16, 17, 18].

Graph classification is one of the most important and prevalent graph related tasks [19], and in this work, we aim to advance graph neural networks for the graph classification task. There are numerous real-world applications for graph classification. For example, proteins can be denoted as graphs [20] and the task to infer whether a protein functions as an enzyme or not can be regarded as a graph classification task; and it can also be applied to forecast Alzheimer's disease progression in which individual brains are represented as graphs [21]. Unlike data samples in classification tasks in other domains such as computer vision [22] and natural language processing [23], graph samples in the graph classification task are described not only by the input (node) features but their graph structures. Both the input node features and the graph structures play crucial roles in the graph classification tasks [15, 16, 18].

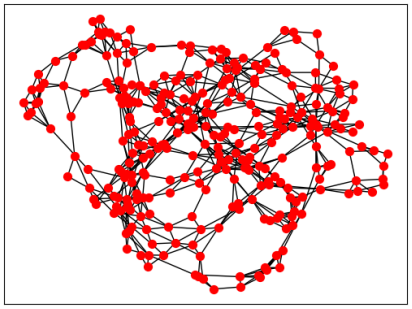
In reality, graphs in the same data set can present significantly different structural properties. Figure 1a demonstrates the distribution of graph size (i.e., the number of nodes) for protein graphs in the D&D dataset [20], where the graph size varies dramatically from 30 to $5,748$. We further illustrate two graphs sampled from the



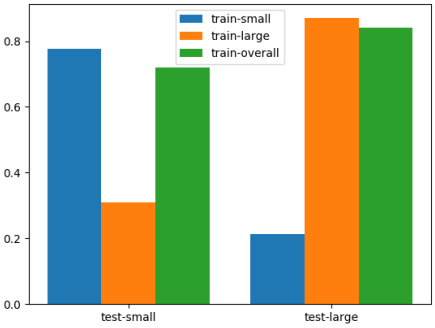
(a) Node size distribution



(b) A graph with 31 nodes



(c) A graph with 302 nodes



(d) Classification accuracy

Figure 1: An illustrative example of varied structural information and its impact on the performance of graph neural network based graph classification. (a) demonstrates the distribution of graph size (i.e., the number of nodes) for protein graphs in the D&D dataset, where the graph size varies dramatically from 30 to 5, 748. ; (b) and (c) show two graphs sampled from the D&D dataset, which present very different structural properties; (d) shows classification performance of three models trained on training sets with various graph sizes.

D&D dataset in Figures 1b and 1c, respectively. These two graphs present very different structural properties such as the number of nodes, graph shapes, and diameters.

The above investigations indicate that graphs in the same data set could have dramatically distinct structural properties. It naturally raises the question – whether we should treat these graphs differently? To investigate this question, we take the graph size as the representative structure-property and demonstrate how it affects the graph classification performance. Specifically, we divide graphs from D&D into two groups based on their graph size – one for graphs with a small number of nodes and the other for graphs with a large number of nodes. Then, we split each group into a training set and a test set. Next, we train three GNN models [1] based on three training sets – the small one, the large one and the overall (a combination of the small and large one), separately. Then, we test their performance on the two test sets. The results are shown in the Figure 1d. In the test set with small/large graph sizes, the model trained on the training set with the same graph size can significantly outperform the other two models.

Investigations and consistent observations on more settings can be found in the Section "Preliminary Data Analysis". These investigations suggest that a unified model is not optimal for graphs with diverse structure properties and efforts are desired to consider the structure-property difference among graphs. Hence, in this paper, we aim to learn "customized models" for graphs with different structural properties. A natural way is to divide the dataset into different splits according to structure properties and train a model for each split. However, we face enormous challenges to achieve this goal in practice. First, there are potentially several structure properties (graph size, density, and etc.) affecting the performance, and we have no explicit knowledge about how the graphs should be split according to these properties. Second, dividing the dataset leads to small training sets for the splits, which may not be sufficient to train satisfactory models. To address these challenges, we propose a novel graph neural network framework, Customized-GNN, for graph classification. The Customized-GNN framework is trained on all graphs in the given training set (without splitting) and able to produce customized GNN models for each individual graph. Specifically, we design an adaptor, which is able to smoothly adjust a general GNN model to a specific one according to the structural properties of a given graph. The general GNN model and the adaptor are learned during the training stage simultaneously utilizing all graphs.

Our major contributions are listed as follows: 1) We empirically observed that graphs in a given dataset could have dramatically distinct structural properties. Furthermore, it is not optimal to train a unified model for graphs with various structure properties for a graph classification task; 2) We propose a framework, Customized-GNN, which is able to generate a customized GNN model for each graph sample based on its structural properties. The proposed framework is general and can be directly applied to many existing graph neural network models; 3) We designed and conducted comprehensive experiments on numerous graph datasets from various domains to verify the effectiveness of the proposed framework.

2 Related Work

Graph Neural Networks have recently drawn great interest due to its strong representation capacity in graph-structured data in many real-world applications. Generally, graph neural networks can be divided into two categories: the spectral approaches and the non-spectral approaches. The spectral methods aim at defining the parameterized filters based on graph spectral theory by using graph Fourier transform and graph Laplacian [31, 32, 33, 34], and the non-spectral methods aim at defining parameterized filters based on nodes' spatial relations by aggregating information from neighboring nodes directly [2, 35].

Graph neural networks have advanced a wide variety of tasks including node classification [3, 2], link prediction [36, 12, 13] and graph classification [15, 16]. In the task of graph classification, one of the most important step is to get a good graph-level representation. A straight-forward way is to directly summarize the graph representation by globally combining the node representations [37]. Recently, there are some works investigating

[1]The GNN model for graph classification uses GCN [3] as the filtering operation and maxpooling as the pooling operation.

learning hierarchical graph representations by leveraging deterministic graph clustering algorithms [32, 38]. There also exist end-to-end models aiming at learning hierarchical graph representations, such as DiffPool [15]. MuchGNN [39] proposed to learn a set of graph channels at each layer to shrink the graph hierarchically. Furthermore, some methods [13, 40, 41] propose principles to select the most important k nodes to form a coarsened graph in each network layer. EigenPooling [16] is based on graph Fourier transform and is able to capture the local structural information. In [26], conditional random fields (CRF) are used to design the pooling operation.

3 Preliminary Data Analysis

Table 1: Graph classification accuracy on different node-size sets

	D&D		**ENZ**		**PROT**		**RE-BI**	
Accuracy (%)	S-test	L-test	S-test	L-test	S-test	L-test	S-test	L-test
S-training	66.2	55.7	45.0	20.0	76.5	51.4	88.6	46.6
L-training	47.2	77.8	27.0	39.0	45.6	78.6	29.3	83.1

Table 2: The classification accuracy of the models trained from four training sets in D&D dataset and Statistics for four training sets.

Training set	Node size range	#Graphs	Accuracy			
			test 1	test 2	test 3	test
training 1	[0,200]	369	76.1	41.2	26.7	50.9
training 2	[200,400]	392	43.5	75.3	82.2	62.4
training 3	[400,2000]	180	23.9	75.3	88.9	56.8
training	[0,2000]	941	64.1	61.9	75.6	66.2

In Figure 1, we have demonstrated that graphs in D&D have varied properties, which affected the performance of GNNs for graph classification. In this section, we aim to further investigate this phenomenon by answering the following two questions – (1) can the observations on D&D be extended to other datasets? and (2) whether incorporating these properties into the models can facilitate the performance?

We choose four representative graph datasets from different domains for this study including **D&D** [20], **ENZ** [6], **PROT** [5] and **RE-BI** [1]. We checked the properties such as node size and edge size. Similar to D&D, graphs in all datasets present very diverse properties. More details about these datasets can be found in Section 5. Following the same setting as D&D, we divide each data into two groups according to the node size, i.e., large training and test (denoted as “L-training" and “L-test") and small training and test (indicated as “S-training" and “S-test"). The results are demonstrated in Table 1. From the table, we make consistent observations with these in D&D – models trained on one property group (e.g., L-training) cannot perform well on the other property group (e.g., S-test).

To answer the second question, we divide D&D into several subsets based on the node size, and then divide each subset into a sub-training set (80%) and a sub-test set (20%). We train models on different sub-training sets separately, and then test their performance on all the sub-test sets. Specifically, we have trained four models on

four different training sets from D&D, which are *training 1*, *training 2*, *training 3* containing graph samples with node sizes from different ranges, and *training* which is the combination of *training 1*, *2* and *3*. Then, we test four models on four test sets, i.e., *test 1*, *test 2*, *test 3* and *test* which is the combination of *test 1*, *2* and *3*. Statistics about these training sets are summarized in Table 2.

The performance of four models on the test sets are illustrated in Table 2. We note that the model trained on a specific training set performs much better on the corresponding test set that shares the same node size range than the other test sets. This suggests the potential to incorporate the structure properties into the model training. In addition, though *training 1*, *training 2* and *training 3* have much fewer training samples, the models trained on specific training sets can achieve better performance on the corresponding test sets compared to these trained from the entire training set (or *training*). This indicates that a unified graph neural network that is trained from the entire training set is not optimal for graphs with various structure properties in the test set.

Discussion. Via the preliminary data analysis, we have established: (1) graphs in real-world data present distinct structure properties that tend to impact the graph classification performance of GNNs; and (2) incorporating the difference has the potential to boost the graph classification performance. These observations lay the foundations of the model design in the next section.

4 The Proposed Framework

In this section, we introduce the proposed framework Customized-GNN that has been designed for graphs with inherently distinct structure properties.

4.1 The Overall Design

As mentioned in earlier sections, graphs in real-world data inherently present distinct structural properties. Thus, we are desired to build distinct GNN models for them. To achieve this goal, we face tremendous challenges. First, we have no explicit knowledge about how the graph structure properties will influence graph neural network models. Second, if we separately train different models for graphs with different structure properties, we have to split the training sets for each model; as a consequence, the training data for each model could be very limited. For example, in the extreme case where each graph has unique graph structural properties, we only have one training sample for the corresponding model. Third, even if we can well train distinct GNN models for different graphs, during the test stage, for an unlabelled graph with unseen structural property, it is hard to decide which trained model we should adopt to make the prediction. In this work, we propose a customized graph neural network framework, i.e., Customized-GNN, which can tackle the aforementioned challenges simultaneously.

An overview of the architecture of Customized-GNN is demonstrated in Figure 2. The basic idea of Customized-GNN is – it generates customized adaptor parameters for each graph sample g_i via an adaptor network with the graph structure properties as input. These generated adaptor parameters are used to adapt a shared GNN model denoted as GNN (this could be any GNN model that works for the graph classification task) to a model specific for the graph sample g_i. The adapted model GNN_i incorporates the structure information of graph g_i, and thus, is customized for the graph sample g_i.

With the proposed Customized-GNN framework, the first challenge is handled, since the influence is implicitly modeled by the adaptor networks, which can customize the shared GNN model to a graph sample specific one. Furthermore, Customized-GNN can be trained on the entire training set without splitting it according to graphs' structure properties. This not only solves the second challenge but also ensures that the trained model can preserve common knowledge from the entire training set. The third challenge is also automatically addressed by the Customized-GNN framework. Given an unseen graph g_j, the Customized-GNN framework first takes its graph structure information as input and generates adaptor parameters. Then, these generated adaptor parameters can be used to customize the general GNN model to a customized one GNN_j to predict the label of g_j.

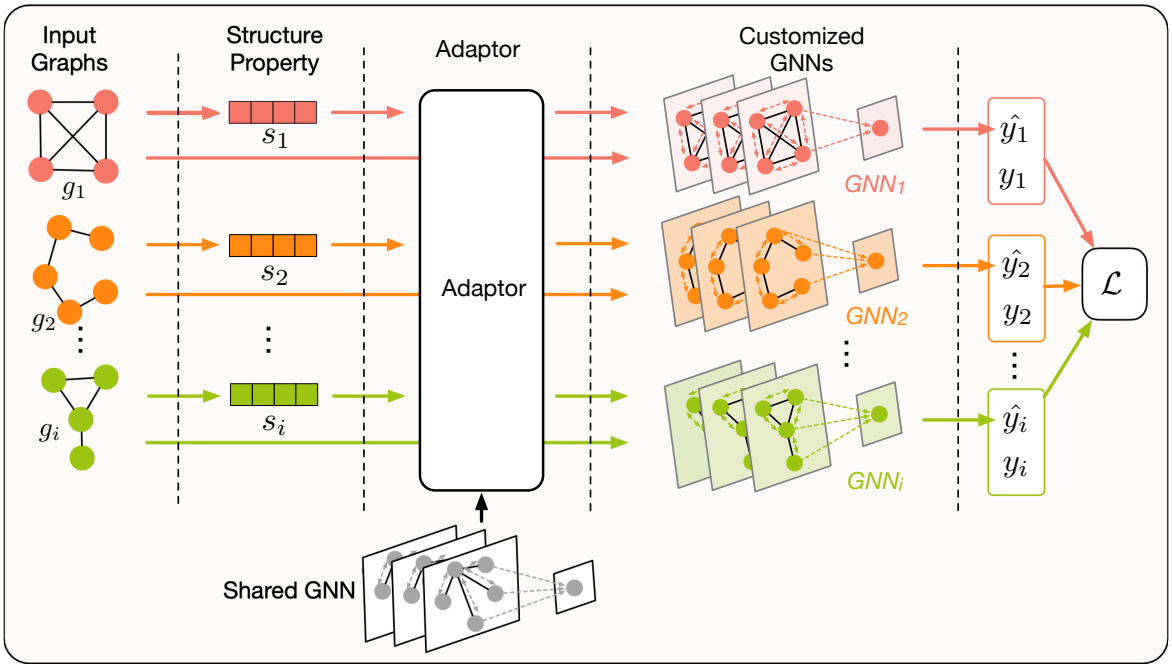


Figure 2: An overview of the proposed customized graph neural networks. Given a set of graph samples, the adaptor networks take their structure properties as input, and generate corresponding adaptor parameters, which are used to adapt a shared GNN model (this could be any GNN model that works for the graph classification task) to a model specific for each graph sample. Each adapted model incorporates the structure information a graph, and thus, is customized for this graph sample to make label prediction. With the predicted label and the ground truth label, we can calculate the overall loss, which is used to guide the optimization of the adaptor networks and the shared GNN model.

4.2 The Adapted Graph Neural Network

Next, we introduce details about the adaptor network, the process of adapting a shared model to a specific one for a given graph, and the time complexity analysis of the proposed framework.

4.3 The Adaptor Network

The goal of the adaptor network is to generate the adaptor parameters for a given graph. From the preliminary data analysis, we have the intuition that the customized GNN model for a specific graph sample should be correlated to its structural properties. However, there is no explicit knowledge about how these structural properties influence graph neural network models. To model this implicit mapping function, we propose to utilize a powerful neural network to generate the model adaptor parameters from the observed structure information of a given sample.

In addition, graph neural networks often consist of several subsequent filtering and pooling layers, which can be viewed as different GNN blocks. For example, K GNN blocks are shown in Figure 3. The graph structure properties of a given sample may have different influences on different GNN blocks. Hence, for each GNN block, we introduce one adaptor network to generate adaptor parameters for each block.

Specifically, we first extract a vector $\mathbf{s}_i$ to denote the structure information of a given graph g_i. We will discuss more details about $\mathbf{s}_i$ in the experiment section. As shown in the Figure 3, the adaptor networks take the structure information $\mathbf{s}_i$ as input and generate the adaptation parameters for each block. In the case where there are K blocks in the graph neural network, we have K independent adaptor networks corresponding to the K

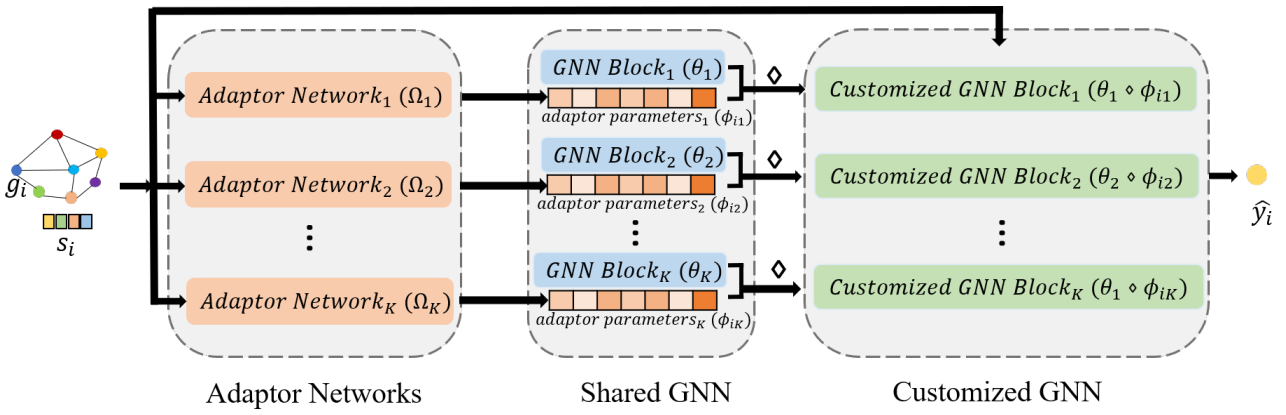


Figure 3: An overview of the GNN adaptation process. Given a specific graph sample g_i, the adaptor networks consisting of K blocks take its graph structure properties s_i as input, and generate corresponding adaptor parameters to adapt each shared GNN block to get a customized GNN for g_i. This customized GNN is then used to make prediction for g_i.

blocks. Note that these adaptor networks share the same input $\mathbf{s}_i$ while their outputs are different. Specifically, the adaptor network for the j-th block can be expressed as follows:

$$\boldsymbol{\phi}_{ij} = h_j(\boldsymbol{s}_i; \Omega_j), j = 1, \ldots, K, \tag{29}$$

where Ω_j denotes the parameters of the j-th adaptor network and $\boldsymbol{\phi}_{ij}$ denotes its output, which will be used to adapt the j-th learning block. The adaptor network h_j can be modeled using any functions. In this work, we utilize feed-forward neural networks due to their strong capability. According to the universal approximation theorem [24], a feed-forward neural network can approximate any nonlinear functions. For convenience, we summarize the process of the K adaptor networks with s_i as input below:

$$\Phi_i = H(\mathbf{s}_i; \boldsymbol{\Omega}_H), \tag{30}$$

where Φ_i contains the generated adaptation parameters of all the GNN blocks for graph g_i and $\boldsymbol{\Omega}_H$ denotes the parameters of the K adaptor networks.

Any existing graph neural network model can be adapted by the Customized-GNN framework to generate sample-specific models based on the structure information. Therefore, we first generally introduce the GNN model for graph classification and describe how to adapt it given a specific sample. Then, we illustrate how to adapt specific GNN models.

4.3.1 A General Adapted Framework

A typical GNN framework for graph classification contains two types of layers, i.e., the filtering layer and the pooling layer. The filtering layer takes the graph structure and node representations as input and generates refined node representations as output. The pooling layer takes graph structure and node representations as input to produce a coarsened graph with a new graph and new node representations. A general GNN framework for graph classification contains K_p pooling layers, each of which follows K_f stacking filtering layers. Hence, there are $K = K_p * K_f$ learning blocks in the GNN framework. A graph-level representation can be obtained from these layers that can be further utilized to perform the prediction. Given a graph sample g_j, we need to adapt each of the K layers according to its adaptor parameters generated from the adaptor network. Via this process, we can generate a GNN model GNN_j specific to g_j.

Without loss of generality, when introducing a filtering layer or a pooling layer, we use an adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and node representations $\mathbf{X} \in \mathbb{R}^{n \times d}$ to denote the input of these layers where n is the number of nodes and d is the dimension of node features. Then, the operation of a filtering layer can be described as follows:

$$\mathbf{X}_{new} = f(\mathbf{A}, \mathbf{X}; \theta_f) \tag{31}$$

where θ_f denotes the parameters in the filtering layer and $\mathbf{X}_{new} \in \mathbb{R}^{n \times d_{new}}$ denotes the refined node representations with dimension d_{new} generated by the filtering layer. Assuming ϕ_f is the corresponding adaptor parameters for this filtering layer, we adapt the model parameter θ_f of this filtering layer as follows:

$$\theta_f^m = \theta_f \diamond \phi_f, \tag{32}$$

where θ_f^m is the adapted model parameter that has the same dimension as the original model parameter θ_f; and $\diamond$ is the adaptation operator. The adaption operator can have various designs, which can be determined according to the specific GNN model. We will provide the details of the adaptation operator when we introduce concrete examples in the following subsections. Then, with the adapted model parameters, we can define the adapted filtering layer as follows:

$$\mathbf{X}_{new} = f(\mathbf{A}, \mathbf{X}; \theta_f \diamond \phi_f). \tag{33}$$

On the other hand, the process of a pooling layer can be described as follows:

$$\mathbf{A}_{new}, \mathbf{X}_{new} = p(\mathbf{A}, \mathbf{X}; \theta_p), \tag{34}$$

where θ_p denotes the parameters of the pooling layer, $\mathbf{A}_{new} \in \mathbb{R}^{n_{new} \times n_{new}}$ with $n_{new} < n$ is the adjacency matrix for the newly generated coarsened graph and $\mathbf{X}_{new} \in \mathbb{R}^{n_{new} \times d_{new}}$ is the learned node representations for the coarsened graph. Similarly, we adapt the model parameters of the pooling layer as follows:

$$\theta_p^m = \theta_p \diamond \phi_p, \tag{35}$$

which leads to the following adapted pooling layer:

$$\mathbf{A}_{new}, \mathbf{X}_{new} = p(\mathbf{A}, \mathbf{X}; \theta_p \diamond \phi_p), \tag{36}$$

where ϕ_p is the adaptation parameters generated by the adaptor network for this pooling layer.

For convenience, we summarize a general GNN model as $GNN(\cdot \mid \Theta_{GNN})$, where Θ_{GNN} is the parameters in all GNN blocks(i.e., θ_f, θ_p in all filtering and pooling layers). Then, for a graph sample g_i, we can adapt the GNN model $GNN(\cdot \mid \Theta_{GNN})$ to a customized model for g_i denoted as $GNN(\cdot \mid \Theta_{GNN} \diamondsuit \Phi_i)$. Note that, as shown in Eq. equation 30, Φ_i contains adaptation parameters of all GNN blocks for a graph sample g_i. The adaptation operations in all GNN blocks (including filtering and pooling layers) are summarized in $\Theta_{GNN} \diamondsuit \Phi_i$. There are numerous GNN models designed for graph classification [13, 25, 16, 26]. The proposed framework can be applied to the majority of these models, i.e., these models all can serve as the $GNN(\cdot \mid \Theta_{GNN})$ model mentioned above. In this work, we focus on three representative GNN models including GCN [3], DiffPool [15] and gPool [13]. We would like to leave the investigations of other GNN models as one future work. Next, we will give details on how to adapt GCN and DiffPool since gPool follows a similar adaptation process.

4.3.2 Adapted GCN: Customized-GCN

Graph Convolutional Network (GCN) [3] is originally proposed for semi-supervised node classification task. The filtering layer in GCN is defined as follows:

$$\mathbf{X}_{new} = f(\mathbf{A}, \mathbf{X}; \theta_f) = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X} \mathbf{W}), \tag{37}$$

where $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ represents the adjacency matrix with self-loops, $\tilde{\mathbf{D}} = \sum_j \tilde{\mathbf{A}}_{ij}$ is the diagonal degree matrix of $\tilde{\mathbf{A}}$ and $\mathbf{W} \in \mathbb{R}^{d \times d_{new}}$ denotes the trainable weight matrix in filtering layer and $\sigma(\cdot)$ is a nonlinear activation function. With the adaptation parameter ϕ_f for this corresponding filtering layer, the adapted filtering layer can be represented as follows:

$$\mathbf{X}_{new} = f(\mathbf{A}, \mathbf{X}; \theta_f \diamond \phi_f) = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \mathbf{X} (\mathbf{W} \diamond \phi_f)). \tag{38}$$

Specifically, we adopt FiLM [27] as the adaption operator. In this case, the dimension of the adaptor parameter is $2d$, i.e., $\phi_f \in \mathbb{R}^{2d}$. We split ϕ_f into two parts $\gamma_f \in \mathbb{R}^d$ and $\beta_f \in \mathbb{R}^d$ and then the adaptation operation can be expressed as follows:

$$\mathbf{W} \diamond \phi_f = (\mathbf{W} \odot br(\gamma_f, d_{new})) + br(\beta_f, d_{new}), \tag{39}$$

where $br(\mathbf{a}, k)$ is a broadcasting function that repeats k times for the vector $\mathbf{a}$; hence, $br(\gamma_f, d_{new}) \in \mathbb{R}^{d \times d_{new}}$ and $br(\beta_f, d_{new}) \in \mathbb{R}^{d \times d_{new}}$ have the same shape as $\mathbf{W}$ and $\odot$ denotes the element-wise multiplication between two matrices.

To utilize GCN for graph classification, we introduce a node-wise max pooling layer to generate graph representation from the node representations as follows:

$$\mathbf{x}_G = p(\mathbf{A}, \mathbf{X}; \theta_p) = max(\mathbf{X}), \tag{40}$$

where $\mathbf{x}_G \in \mathbb{R}^{1 \times d_{new}}$ denotes the graph-level representation and $max()$ takes the maximum over all the nodes. Note that the max-pooling operation does not involve learnable parameters and thus no adaptation is needed for it. We refer to an adapted GCN framework as Customized-GCN.

4.3.3 Adapted diffpool: Customized-DiffPool

DiffPool is a hierarchical graph level representation learning method for graph classification [15]. The filtering layer in DiffPool is the same as Eq. equation 37 and its corresponding adapted version is shown in Eq. equation 38. Its pooling layer is defined as follows:

$$\mathbf{S} = softmax(f_a(\mathbf{A}, \mathbf{X}; \theta_{f_a})), \tag{41}$$

$$\mathbf{X}_{new} = \mathbf{S}^T \mathbf{Z}, \tag{42}$$

$$\mathbf{A}_{new} = \mathbf{S}^T \mathbf{A} \mathbf{S}, \tag{43}$$

where f_a is a filtering layer embedded in the pooling layer, $\mathbf{S} \in \mathbb{R}^{n \times n_{new}}$ is a soft-assignment matrix, which softly assigns each node into a supernode to generate a coarsened graph. Specifically, the structure and the node representations for the coarsened graph are generated by Eq. equation 43 and Eq. equation 42 respectively, where $\mathbf{Z} \in \mathbb{R}^{n \times d_{new}}$ is the output of the filtering layers. To adapt the pooling layer, we only need to adapt Eq. equation 41, which follows the same way as introduced in Eq. equation 38 as it is also a filtering layer. We refer to the adapted diffpool model as Customized-DiffPool.

4.4 Training and Test via the Customized Framework

Given a graph sample g_i with the adjacency matrix $\mathbf{A}_i$, and the feature matrix $\mathbf{X}_i$, the Customized-GNN framework performs the classification task as follows:

$$\tilde{y_i} = GNN(\boldsymbol{A}_i, \mathbf{X}_i; \boldsymbol{\Theta}_{\mathcal{GNN}} \lozenge H(\boldsymbol{s}_i; \boldsymbol{\Omega}_H)). \tag{44}$$

During the training, we are given a set $\mathcal{G} = \{g_i, y_i\}$ of N graphs as training samples, where each graph g_i is associated with a ground truth label y_i. Then, the objective function of Customized-GNN can be represented as follows:

$$\min_{\boldsymbol{\Omega}_H, \boldsymbol{\theta}_{\mathcal{GNN}}} \sum_{i=1}^{N} \mathcal{L}(y_i, GNN(\boldsymbol{A}_i, \mathbf{X}_i; \boldsymbol{\Theta}_{\mathcal{GNN}} \diamondsuit H(\boldsymbol{s}_i; \boldsymbol{\Omega}_H))), \tag{45}$$

where N is the number of training samples and $\mathcal{L}$ is a loss function. In this work, we use Cross-Entropy as the loss function and adopt ADAM [28] to optimize the objective.

During the test phase, the label of a given sample g_ℓ can be inferred using equation 44. Specifically, the graph structure information $\mathbf{s}_\ell$ of the sample is first utilized as the input of the adaptor network $H(\cdot; \boldsymbol{\Omega})$ to identify its distribution information, which is then utilized to adapt the shared model parameter Θ_{GNN} to generate a sample-specific model GNN_ℓ. This sample-specific model finally performs the classification for this sample.

Table 3: The statistics of seven datasets. #Graphs denotes the number of graphs. #Class is the number of graph classes. Node size indicates range, average and standard deviation of the number of nodes among the graphs. Edge size represents range, average and standard deviation of the number of edges among the graphs.

Dataset	#Graphs	#Class	Node size			Edge size		
			range	mean	std	range	mean	std
COLLAB	5000	3	[32, 492]	74.5	62.3	[60, 40120]	2457.8	6439.0
ENZ	600	6	[2, 125]	32.6	14.9	[1, 149]	62.14	25.5
PROT	1113	2	[4, 620]	39.1	45.7	[5, 1049]	72.82	84.6
D&D	1178	2	[30, 5748]	284.3	272.0	[63, 14267]	715.65	693.9
RE-BI	2000	2	[63, 782]	429.6	554.0	[4, 4071]	497.8	623.0
RE-5K	4999	5	[22, 3648]	508.5	452.6	[21, 4783]	594.9	566.8
NCI109	4127	2	[4, 111]	29.6	13.6	[3, 119]	32.1	15.0

4.5 Time Complexity Analysis

In this subsection, we analyze the additional time required to calculate the adaptation parameters and perform the adaptation. Specifically, we use the FiLM adaptation operator, as an example for the adaptor network. For convenience, the dimension of the output node features in all layers is assumed to be the same d. The dimension of the output of the adaptation network ϕ_f is $2d$. Furthermore, we assume that the input of the adaptation network, i.e., the graph property information $\mathbf{s}_i$ is with dimension s. Then, the time complexity to generate the adaptation parameters for a single block using Eq. equation 29 is $O(2d \cdot s) = O(d \cdot s)$. Furthermore, the time required to adapt the parameters for a single block with Eq. equation 39 is $O(d^2)$. Hence, for graph neural networks with K learning blocks, the time complexity to calculate the adaptation parameters and perform the adaptation for all learning blocks is $O(K \cdot d \cdot s + K \cdot d^2)$. Note that, the time complexity of a single filtering operation in Eq. equation 37 is $O(m \cdot d + n \cdot d^2)$ where m denotes the number of edges while n is the number of nodes. Therefore, the total time complexity for K learning blocks without adaptation is $O(K \cdot m \cdot d + K \cdot n \cdot d^2)$. Furthermore, s is typically small (much smaller than m); hence, the additional time complexity introduced by the adaptation operation is rather small.

5 Experiments

In this section, we conducted comprehensive experiments to verify the effectiveness of the proposed Customized-GNN framework. We first describe the experimental settings. Then, we evaluate the performance of the

framework by comparing original GCN, DiffPool and gPool with the adapted GCN, DiffPool, gPool models by the Customized-GNN framework. Next, we analyze the importance of different components in the adaptor operator. Finally, we conduct case studies to further facilitate our understanding of the proposed method.

5.1 Experimental Settings

We carried out graph classification tasks on seven datasets. Some key statistics of these datasets used in our experiments are shown in Table 3, and more details of them are introduced as follows:

- **COLLAB** [1] is a dataset of scientific collaboration networks, which describes collaboration pattern of researchers from three different research fields.
- **ENZ** [6] is a dataset of protein tertiary structures of six classes of enzymes.
- **PROT** [5] is a dataset of protein structures, where each graph represents a protein and each node represents a secondary structure element (SSE) in the protein.
- **D&D** [20] is a dataset of protein structures. Each protein is represented as a graph, where each node in a graph represents an amino aid and each edge between two nodes denotes that they are less than 6 Ångstroms apart.
- **RE-BI** and **RE-5K** [1] are two datasets of online discussion threads crawled from different subreddits in Reddit, where each node represents an user and each edge between two users represents their interaction.
- **NCI109** [6] is a dataset of chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines, which are provided by Natinal Cancer Institue (NCI).

Next, we describe the baselines. In the Section "The Proposed Framework" , we apply the proposed framework to adapt three graph neural networks models: a basic graph convolutional network (GCN) [3], and two SOTA graph classification models DiffPool [15] and gPool [13], respectively. The corresponding adapted versions are Customized-GCN, Customized-DiffPool and Customized-gPool, respectively. *Our evaluation purpose is if the proposed framework can boost the performance of existing models by adapting them to their corresponding customized versions.* Thus, (1) to validate the effectiveness of the proposed model, we compare Customized-GCN, Customized-DiffPool, Customized-gPool with GCN, DiffPool and gPool; and (2) we have not chosen models in [25, 16, 26] as baselines here but the proposed framework can be directly applied to adapt them as well. Note that in this work, we construct a set of simple structural features $\mathbf{s}_i$ of g_i such as the number of nodes, the number of edges and the graph density; however, it is flexible to include other complex features by the proposed framework. Furthermore, we create baselines to directly concatenate the graph structure properties $\mathbf{s}_i$ to the output graph embedding of the GCN, DiffPool and gPool model. Correspondingly, we call these three methods as Concat-GCN, Concat-Diff and Concat-gPool. In addition, we develop baseline methods, Multi-GCN, Multi-Diff and Multi-gPool. They learn multiple graph convolutional networks for graph samples with different structural information. More details of these baselines are as follows:

- **GCN** [3] is originally proposed for semi-supervised node classification. It consists of a stack of GCN layers, where a new representation of each node is computed via transforming and aggregating node representations of its neighbouring nodes. Finally, a graph representation is generated from node representations in the last GCN layer via a global max-pooling layer, and then used for graph classification.
- **Diffpool** [15] is a recently proposed method which has achieved state-of-the-art performance on the graph classification task. It proposes a differentiable graph pooling approach to hierarchically generate a graph-level representation by coarsening the input graph level by level.

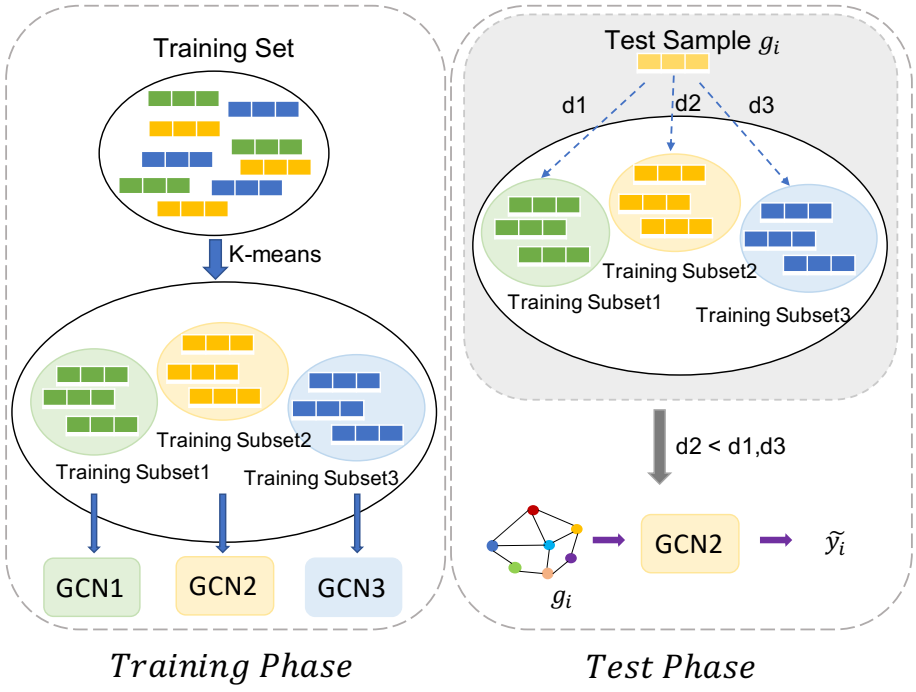


Figure 4: The framework of Multi-GCN with 3 clusters. we group training samples into 3 clusters, and train a GCN model for each cluster. Given a test sample g_i, we measure the distance between this sample and the centroids of the three clusters and utilize the corresponding model of the closest cluster to perform the prediction for this sample.

- **gpool** [13] is a newly proposed pooling method that has achieved state-of-the-art performance on the graph classification task. It develops a U-Net-like architecture for graph dat, consisting of graph pooling operation and unpooling operation based on node importance value.
- **Concat-GCN (or Concat-Diff, Concat-gpool)** is baseline method that directly concatenates the graph structure properties $\mathbf{s}_i$ to the output graph embedding of the GCN, DiffPool and gPool model.
- **Multi-GCN (or Multi-Diff, Multi-gpool)** consists of several GCN (or Diffpool, gpool) models trained from different subsets of the training datset. As shown in Figure 4, we first cluster data samples from training set into different training subsets via K-means method based on the graph structural information. Note that in this work, the structural information $\mathbf{s}_i$ of g_i includes the number of nodes, the number of edges and the graph density. Then train different models are trained from different training subsets. During the test phase, given a test graph sample, we first compute the euclidean distance between its graph structural properties and the centroids of different training subsets. Then, the model trained on the closest training subset is selected to do label prediction for this graph sample. In this experiment, we set the number of clusters to 2 and 3, and denote the corresponding frameworks as Multi-GCN-2 (or Multi-Diff-2, Multi-gpool-2) and Multi-GCN-3 (or Multi-Diff-3, Multi-gpool-3).

5.2 Graph Classification Performance Comparison

In this subsection, we first perform the comparison following the traditional setting. To further demonstrate the advantage of the proposed frameworks, we show their adaptability when the properties of test graphs are

Table 4: Comparisons of graph classification performance in terms of accuracy.

Accuracy (%)	Datasets						
	COLLAB	**ENZ**	**PROT**	**DD**	**RE-BI**	**RE-5K**	**NCI109**
GCN	67.9±1.4	50.4±3.0	77.0±2.3	79.3±5.3	82.6±4.9	50.7±1.3	74.9±2.7
Concat-GCN	68.4±1.4	52.5±5.1	78.4±1.9	77.6±3.2	80.7±3.5	50.7±1.0	75.6±1.2
Multi-GCN-2	68.3±1.4	47.0±1.8	79.5±1.3	77.1±2.4	80.6±3.5	50.3±1.9	74.2±1.9
Multi-GCN-3	67.0±1.4	44.6±5.4	79.9±2.2	76.7±3.5	77.5±7.3	48.5±2.1	75.8 ±1.5
Customized-GCN	71.3±1.0	55.4±4.6	78.8±3.2	79.6±3.9	91.5±1.6	53.3±1.3	76.7±1.1
DiffPool	70.6±1.2	57.9±2.5	78.6±2.5	81.5±4.1	89.6±1.1	56.2±1.1	77.5±0.7
Concat-Diff	70.7±0.7	60.0±1.7	77.7±2.5	81.3±2.9	91.1±1.7	54.9±1.4	78.0±0.5
Multi-Diff-2	70.7±0.6	56.3±1.3	80.0±1.2	79.3±2.9	89.9±2.5	53.7±0.6	76.8±0.7
Multi-Diff-3	70.8±1.1	52.5±0.8	80.9±1.7	80.6±2.3	88.8±0.7	53.4±2.4	78.5±1.2
Customized-DiffPool	73.6±0.5	57.9±7.2	78.6±2.9	80.6±2.6	95.1±1.6	55.8±1.1	78.2±0.9
gPool	69.4±2.2	53.8±3.2	77.3±3.0	78.9±5.5	88.9±1.6	51.3±0.6	77.1±1.2
Concat-gPool	69.7±0.5	57.1±1.8	79.1±2.2	78.0±2.6	88.5±1.3	50.9±2.2	76.3±0.7
Multi-gPool-2	69.0±1.9	50.8±5.1	79.7±1.0	79.5±2.7	84.0±3.2	49.3±2.3	73.5±2.1
Multi-gPool-3	68.9±1.6	46.2±3.2	80.6±0.8	80.0±3.6	83.1±4.5	48.9±1.8	75.2±1.9
Customized-gPool	72.3±1.0	62.9±3.6	80.6±1.6	80.0±3.1	91.1±0.7	53.3±1.4	76.5±1.9

different from these of training graphs. Following the setting in [15], for each graph dataset, we randomly shuffle the dataset and then split 90% of the data as the training set and the remaining 10% as test set. We train all the models on the training set and evaluate their performance on the test set with accuracy as the measure. We repeat this process with different data shuffling and initialization seeds for 4 times and report the average performance and standard variance. In terms of the implementation details, the GCN/Customized-GCN model consists of 3 filtering layers and a single max-poling layer; the hidden dimension of each filtering layer is 20; and ReLU [29] activation is applied after each filtering layer. For DiffPool/Customized-DiffPool and gPool/Customized-gPool, we set $K_p = 2$, $K_f = 3$ and the dimension of hidden filtering layer 20. We adopt fully-connected networks to implement the adaptor networks in the Customized-GNN frameworks. Its input dimension is the same as the dimension of the graph structural information.

The results are shown in Table 4. We notice that the Concat- and the Multi- version of the GNN models can, in some cases, achieve comparable or even better performance than their corresponding original versions. This indicates that utilizing the graph structure properties has the potential to help improve the model performance. However, the performance of these variants is not so stable across different datasets, which means that these simple methods are not suitable for all datasets. For example, the Concat- versions may work well on datasets where the label is directly related with the graph structure properties but fail on those datasets where graph structure properties have more implicit impact on the labels. On the other hand, the performance of the Multi-version of the GNN models is heavily dependent on how the data is split into different groups. It is not practical to find good splits manually. Furthermore, simply training different models for different graphs can lead to unsatisfactory performance because less training data is available for each model. In contrast, our proposed Customized models learn sample-wise adaptation, which automatically finds suitable models for different data samples according to their graph structure properties. Compared with the original GCN, DiffPool and gPool, the corresponding Customized models achieve better performance in most of the datasets. This demonstrates that the sample-wise adaptation performed by the Customized-GNN framework can boost the performance of GNN frameworks.

Adaptability Study. To further show the adaptability of the proposed framework to new graphs with different structures, we order graphs according to their node sizes in non-decreasing order. Then, we use the first 80% of the data as training set and the remaining 20% as test set. The purpose of this setting is to simulate the case where structures of graphs in the test set are different from those in the training set. We only show the results on

Table 5: Adaptability study. (Note here Cust-X denotes Customized-X)

Accuracy(%)	Methods					
	Cust-GCN	GCN	Cust-DiffPool	DiffPool	Cust-gPool	gPool
ENZ	22.2	20.5	25.6	22.2	35.0	24.8
RE-BI	70.2	50.4	78.6	52.7	80.0	59.9

the **ENZ** and **RE-BI** datasets in Table 5, since observation from other datasets are consistent. We note that (1) GCN, DiffPool and gPool cannot work properly in this setting; and (2) the customized frameworks perform much better under this setting. These results demonstrate the ability of the learned Customized-GNNs to adapt GNNs to graphs with new properties.

5.3 Ablation Study

In this subsection, we investigate the effectiveness of different components in the adaptor operator in Eq. equation 39 used in our model. Specifically, we want to investigate whether γ_f and β_f play important roles in the adaptor operator by defining the variants of Customized-GCN – **Customized-GCN**$_\gamma$: It is a variant of the adaptor operator with only element-wise multiplication operation where instead of Eq. equation 39, the adaptation process is now expressed as: $\mathbf{W} \diamond \phi_f = (\mathbf{W} \odot br(\gamma_f, d_{new}))$; and **Customized-GCN**$_\beta$: It is a variant of the adaptor operator with only element-wise addition operation where instead of Eq. equation 39, the adaptation process is now: $\mathbf{W} \diamond \phi_f = \mathbf{W} + br(\beta_f, d_{new})$.

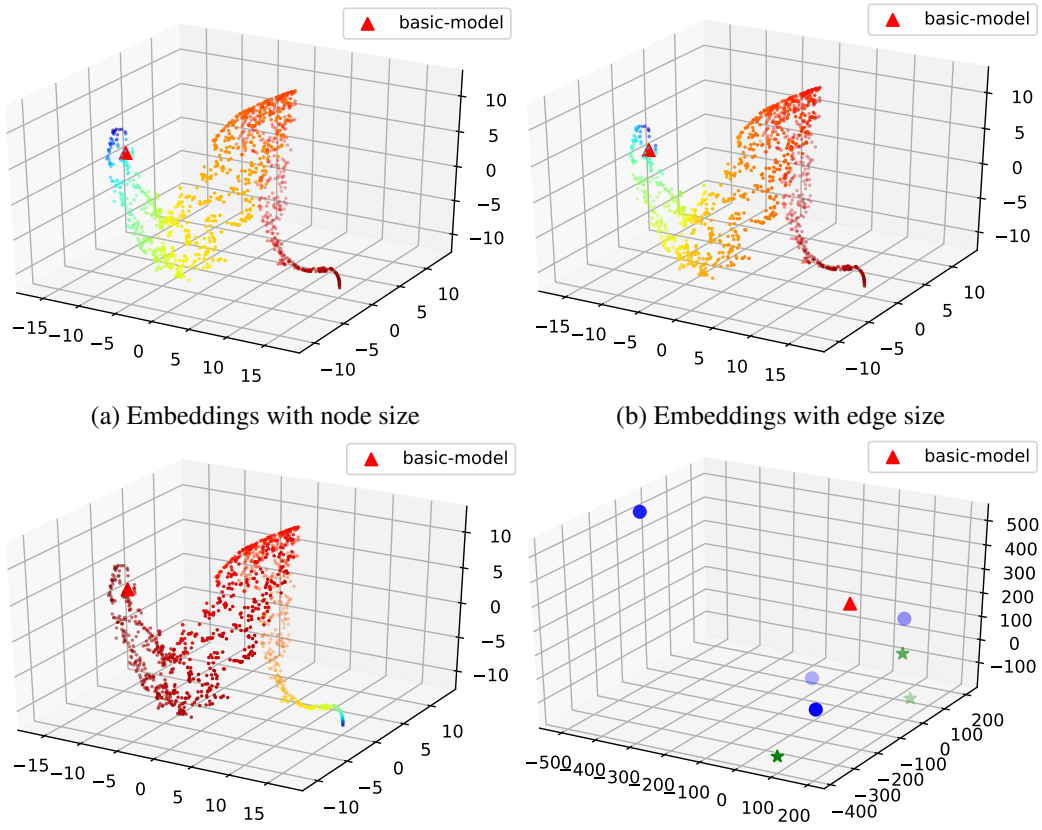
Table 6: Ablation study.

Accuracy (%)	Datasets						
	COLLAB	**ENZ**	**PROT**	**DD**	**RE-BI**	**RE-5K**	**NCI109**
GCN	69.9	51.8	76.6	77.2	81.9	50.4	75.7
Customized-GCN$_\gamma$	70.8	52.3	77.6	78.1	85.2	51.7	76.0
Customized-GCN$_\beta$	71.2	54.0	77.9	78.0	88.8	51.9	77.1
Customized-GCN	73.2	55.9	77.9	79.3	90.4	52.9	77.1

Following the previous experimental setting, we compared Customized-GCN with its variants. The results are presented in Table 6. We observe that both **Customized-GCN**$_\gamma$ and **Customized-GCN**$_\beta$ can outperform the original GCN model. It indicates that both terms with γ and β are effective for the adaptation and utilizing either one of them can already adapt the original model in a reasonable manner. We also note that the Customized-GCN model outperforms both **Customized-GCN**$_\gamma$ and **Customized-GCN**$_\beta$ on most of the datasets. It demonstrates that the adaption effects of the term with γ and β are complementary to each other and combining them together can further enhance the performance.

5.4 Case Study

To further illustrate the effectiveness of the proposed framework, we conducted case studies on D&D. First, we visualize the distribution of embeddings of sample-specific model parameters for different graph samples with various node sizes, edge sized and densities. Specifically, we take the parameters of the first filtering layer of each sample-specific Customized-GCN framework and then utilize t-sne [30] to project these parameters to 3-dimensional embeddings. We visualize these 3-d embeddings in the form of scatter plot as shown in Figure 5a, 5b and 5c. Note that in these figures, the red triangle denotes the embedding of the parameters (i.e. $\mathbf{W}$) of the



(a) Embeddings with node size

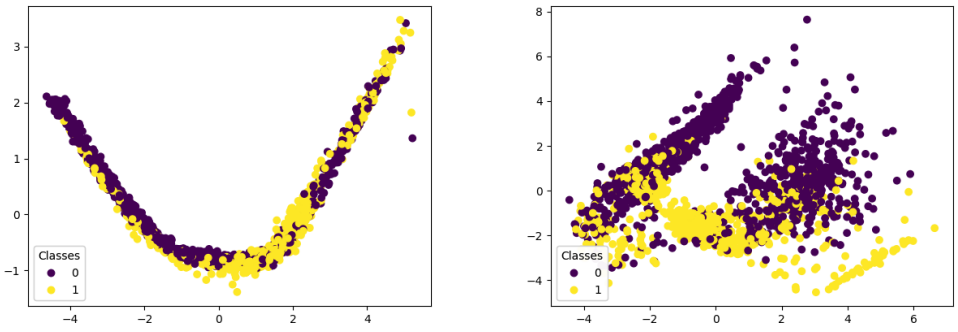
(b) Embeddings with edge size

(c) Embeddings with density

(d) Sample demonstration

Figure 5: Case Study. (a) depicts the model embeddings and (b) demonstrates model embeddings for graphs that are mistakenly classified by GCN, but correctly classified by Customized GCN; and (c) and (d) illustrate the embeddings for graphs extracted by GCN and Customized-GCN, respectively;

original GCN model (the one before adaptation). For each point in these figures, we use color to represent the scale of values in terms of node size (or edge size, density). Specifically, a deeper red color indicates a larger value, while a deeper blue color indicates a smaller value. We make some observations from Figure 5a, 5b and 5c. First, the proposed Customized-GCN framework indeed generates distinct models for different graph samples that are different from the original model. Second, the points with similar colors stay closely with each other, which means that graphs with similar structural information share similar models. In addition, in Figure 5d, we illustrate the sample-specific model parameters for seven samples with different number of nodes. They are mis-classified by the original GCN model but correctly classified by the proposed Customized-GCN framework. It is obvious that Customized-GCN has generated seven different GCN models for these graph samples, each of which can successfully predict the label for the corresponding sample. We further visualize the graph embeddings before the classification layer, extracted by the model GCN and Customized-GCN. These embeddings from the two models are then projected to a 2-dimensional space via PCA and shown in Figure 6a and 6b, respectively. We observe that the embeddings from different categories are better separated by Customized-GCN. This demonstrates that, compared to the original GCN, the proposed framework can get more distinct embeddings, and thus can achieve better classification performance.



(a) GCN (b) Customized-GCN

Figure 6: Embedding Visualization.

6 Conclusion

In this paper, we propose a general graph neural network framework, Customized-GNN, to deal with graphs that have various graph structure properties. Comprehensive experiments demonstrated that the Customized-GNN framework can effectively adapt both flat and hierarchical GNNs to enhance their performance. Future research directions include better modeling the adaptor networks, considering more complex properties, and adapting more existing graph neural networks models.

References

[1] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2015.

[2] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in Advances in neural information processing systems, 2017, pp. 1024–1034.

[3] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," arXiv preprint arXiv:1609.02907, 2016.

[4] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," arXiv preprint arXiv:1710.10903, 2017.

[5] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," Bioinformatics, vol. 21, no. suppl_1, 2005.

[6] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," Journal of Machine Learning Research, vol. 12, no. Sep, pp. 2539–2561, 2011.

[7] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017, pp. 1263–1272.

[8] Y. Rong, Y. Bian, T. Xu, W. Xie, Y. Wei, W. Huang, and J. Huang, "Self-supervised graph transformer on large-scale molecular data," in NeurIPS, 2020.

[9] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, "Speech recognition using deep neural networks: A systematic review," IEEE Access, vol. 7, pp. 19 143–19 165, 2019.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," arXiv preprint arXiv:1810.04805, 2018.

[12] Z. Zhang, H. Yang, J. Bu, S. Zhou, P. Yu, J. Zhang, M. Ester, and C. Wang, "Anrl: Attributed network representation learning via deep neural networks." in IJCAI, vol. 18, 2018, pp. 3155–3161.

[13] H. Gao and S. Ji, "Graph u-nets," arXiv preprint arXiv:1905.05178, 2019.

[14] S. Vashishth, S. Sanyal, V. Nitin, and P. Talukdar, "Composition-based multi-relational graph convolutional networks," in International Conference on Learning Representations, 2020. [Online]. Available: https://openreview.net/forum?id=BylA_C4tPr

[15] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in Advances in neural information processing systems, 2018, pp. 4800–4810.

[16] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, "Graph convolutional networks with eigenpooling," in Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 723–731.

[17] J. Li, Y. Ma, Y. Wang, C. Aggarwal, C.-D. Wang, and J. Tang, "Graph pooling with representativeness," in 2020 IEEE International Conference on Data Mining (ICDM). IEEE, 2020, pp. 302–311.

[18] H. Gao, Y. Liu, and S. Ji, "Topology-aware graph pooling networks," IEEE Transactions on Pattern Analysis and Machine Intelligence, 2021.

[19] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A fair comparison of graph neural networks for graph classification," arXiv preprint arXiv:1912.09893, 2019.

[20] P. D. Dobson and A. J. Doig, "Distinguishing enzyme structures from non-enzymes without alignments," Journal of molecular biology, vol. 330, no. 4, pp. 771–783, 2003.

[21] T.-A. Song, S. R. Chowdhury, F. Yang, H. Jacobs, G. El Fakhri, Q. Li, K. Johnson, and J. Dutta, "Graph convolutional neural networks for alzheimer's disease classification," in 2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019). IEEE, 2019, pp. 414–417.

[22] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," Neural computation, vol. 29, no. 9, pp. 2352–2449, 2017.

[23] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, "Text classification algorithms: A survey," Information, vol. 10, no. 4, p. 150, 2019.

[24] K. Hornik, "Approximation capabilities of multilayer feedforward networks," Neural networks, vol. 4, no. 2, pp. 251–257, 1991.

[25] E. Ranjan, S. Sanyal, and P. P. Talukdar, "Asap: Adaptive structure aware pooling for learning hierarchical graph representations," arXiv preprint arXiv:1911.07979, 2019.

[26] H. Yuan and S. Ji, "Structpool: Structured graph pooling via conditional random fields," in International Conference on Learning Representations, 2020.

[27] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville, "Film: Visual reasoning with a general conditioning layer," in Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.

[29] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in Proceedings of the 27th international conference on machine learning (ICML-10), 2010, pp. 807–814.

[30] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," Journal of machine learning research, vol. 9, pp. 2579–2605, 2008.

[31] J. Bruna, W. Zaremba, A. Szlam, and Y. 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.

[32] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in Advances in neural information processing systems, 2016, pp. 3844–3852.

[33] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," in Thirty-second AAAI conference on artificial intelligence, 2018.

[34] T. N. Kipf and M. 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.

[35] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. 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.

[36] K. Schütt, P.-J. Kindermans, H. E. S. Felix, S. Chmiela, A. Tkatchenko, and K.-R. Müller, "Schnet: A continuous-filter

convolutional neural network for modeling quantum interactions," in Advances in neural information processing systems, 2017, pp. 991–1001.

[37] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in Advances in neural information processing systems, 2015, pp. 2224–2232.

[38] M. Fey, J. Eric Lenssen, F. Weichert, and H. Müller, "Splinecnn: Fast geometric deep learning with continuous b-spline kernels," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 869–877.

[39] K. Zhou, Q. Song, X. Huang, D. Zha, N. Zou, and X. Hu, "Multi-channel graph neural networks," arXiv preprint arXiv:1912.08306, 2019.

[40] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," arXiv preprint arXiv:1904.08082, 2019.

[41] Z. Zhang, J. Bu, M. Ester, J. Zhang, C. Yao, Z. Yu, and C. Wang, "Hierarchical graph pooling with structure learning," arXiv preprint arXiv:1911.05954, 2019.

Fact Ranking over Large-Scale Knowledge Graphs with Reasoning Embedding Models

Hongyu Ren†,† Ali Mousavi*, Anil Pacaci*, Shihabur R. Chowdhury*, Jason Mohoney‡*,
Ihab F. Ilyas*, Yunyao Li*, Theodoros Rekatsinas*
*Apple
†Stanford University, hyren@stanford.edu
‡University of Wisconsin-Madison, mohoney2@wisc.edu
*amousavi, apacaci, shihab, iilyas, yunyaoli, thodrek@apple.com

Abstract

Knowledge graphs (KGs) serve as the backbone of many applications such as recommendation systems and question answering. All these applications require reasoning about the relevance of facts in a KG to downstream applications. In this work, we describe our efforts in building a solution to reason about the importance of facts over continuously updated industry-scale KGs. We focus on the problem of fact ranking and evaluate to what extent modern knowledge graph embedding (KGE) models provide a representation for addressing this problem. To this end, we discuss unique challenges associated with solving this task in industrial settings and evaluate how accurately different KGE models and text-based embedding models can solve the problem of fact ranking.

1 Introduction

Knowledge graphs (KGs) are the backbone of applications such as question answering in virtual assistants and recommendation systems in. These applications require a broad range of knowledge that is continuously updated with recent facts from disparate data sources [13, 20]. Reasoning about the importance of facts in an industry-scale KG with billions of entities and facts across diverse domains is a challenging problem. An automated and scalable solution scalable across entity types and domains in a KG has obvious benefits.

Here, we focus on the problem of fact ranking. Fact ranking provides an importance-based ranking over facts for a given real-world entity. For example, given the question "What is the occupation of LeBron James?", the answer "basketball player" should be ranked higher than "television actor" or "screenwriter" despite the fact that these two are also LeBron James's occupations. Fact ranking generalizes the problem of recommendation generation [3] over KGs. We are interested in facts that cannot be ranked using a simple importance or popularity score. For example, ranking occupation of entities as described earlier. Another example is to generate recommendation for entities that are relevant within users' search context. For example, for the user query "How tall is LeBron James", we want to recommend a ranked list of top KG entities that are related to the query entity "LeBron James" and are aligned with users' search intent, i.e., they are "Person" entities with a "height" attribute. Fact ranking is important during rendering these enriching entity-centric experiences in intelligent assistants.

In this paper, we propose a solution to fact ranking based on modern Knowledge Graph Embedding (KGE) models and present an experimental evaluation in large-scale settings. Our solution adopts state-of-the-art multi-hop reasoning models. Specifically, we build on the recent Query2Box model [24] and demonstrate how

†work done during an internship at Apple.

the embeddings obtained by this model can address fact ranking over large-scale KGs. A major challenge in employing KGE models for fact ranking in real-world applications is to reason about the importance score and the rank of a facts obtained by an embedding model. We address this challenge by proposing a new metric for measuring the stability of embedding models across different rounds, namely, an adaptive version of Kendall's Tau that also takes into account the importance scores obtained by the embedding models. In this way, we can better measure the effects of the learned embeddings on the downstream use cases. Our approach is in contrast to using the standard forms of Kendall's Tau or Rank-based Overlap metrics, which measure the consistency across two ranked lists by considering only the number of discordant pairs/swaps between the two lists. We demonstrate that the reasoning-based Query2Box model leads to significantly more stable embeddings compared to one-hop embedding models such as DistMult [37]. We also propose a new indexing scheme and apply multi-query optimization for efficient search over the generated embedding vectors for supporting use-cases such as vector similarity-based related entity search.

Finally, we compare Query2Box against modern generative natural language (NL) models [18] and demonstrate that NL models require significant fine-tuning of the prompt to obtain similar fact ranking results as the Query2Box model.

2 Preliminaries

We now review background relevant to our study. Our discussion focuses on knowledge graph representation models and aims to highlight the differences between shallow KG embedding models such as the popular DistMult [37], TransE [2], and RotatE [27] models and more recent reasoning-based embedding models [24, 25].

2.1 Shallow Knowledge Graph Embeddings

A knowledge graph (KG) $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ consists of a set of nodes $\mathcal{V}$, a set of edges $\mathcal{E}$. $\mathcal{G}$ also defines a set of relations $\mathcal{R}$, and each edge $e \in \mathcal{E}$ represents a triple (v_s, r, v_o) where $r \in \mathcal{R}$ and $v_s, v_o \in \mathcal{V}$. Here, v_s corresponds to the vector representation of the `subject` of the fact that corresponds to the edge and v_o to the `object` of the fact. Finally, r corresponds to the vector representation of the `predicate` associated with the triple.

Shallow KG embeddings [2, 37, 27, 31] learn an embedding function f_θ that maps all the entities and relations on the graph to latent space in order to preserve the structure of graph. Most KG embeddings implement the embedding function f_θ as a matrix lookup. Specifically, the parameters include an entity embedding matrix $\mathbf{V}_\theta \in \mathbb{R}^{|\mathcal{V}| \times d}$ and a relation embedding matrix $\mathbf{R}_\theta \in \mathbb{R}^{|\mathcal{R}| \times d}$, where d is the latent space dimension.

Training Shallow KG Embeddings: In order to train the two embedding matrices, these methods optimize a contrastive objective, which is to minimize a predefined distance function `Dist` of existing edges $e = (v_s, r, v_o) \in \mathcal{E}$ while maximize that of non-existing edges $e' = (v_s, r, v'_o) \notin \mathcal{E}$. Different shallow KG embeddings have different definitions of the distance function `Dist`, the detail is listed in Table 1. In previous KG embedding works [27, 39], the loss function in the contrastive objective is defined as:

$$\mathcal{L} = -\log \sigma \left(\gamma - \texttt{Dist}(v_s, r, v_o)\right) - \sum_{j=1}^{k} \frac{1}{k} \log \sigma \left(\texttt{Dist}(v_s, r, v'_{o_j}) - \gamma\right), \tag{46}$$

where σ is the sigmoid function, γ is the margin. We optimize over the loss function using stochastic training for several iterations. In each iteration, these methods sample a batch of existing edges from the graph and construct non-existing edges by keeping the subject v_s and the type of the edge r fixed while perturbing the object v_o.

Table 1: The distance function of shallow KG embeddings and KG reasoning embeddings.

Model	Embedding Space	Distance
TransE [2]	$f_\theta(v_s), f_\theta(v_o) \in \mathbb{R}^d, f_\theta(r) \in \mathbb{R}^d$	$\|f_\theta(v_s) + f_\theta(r) - f_\theta(v_o)\|$
RotatE [27]	$f_\theta(v_s), f_\theta(v_o) \in \mathbb{C}^d, f_\theta(r) \in \mathbb{C}^d$	$\|f_\theta(v_s) \circ f_\theta(r) - f_\theta(v_o)\|$
DistMult [37]	$f_\theta(v_s), f_\theta(v_o) \in \mathbb{R}^d, f_\theta(r) \in \mathbb{R}^d$	$- < f_\theta(v_s), f_\theta(r), f_\theta(v_o) >$
ComplEx [31]	$f_\theta(v_s), f_\theta(v_o) \in \mathbb{C}^d, f_\theta(r) \in \mathbb{C}^d$	$-\text{Re}(< f_\theta(v_s), f_\theta(r), \overline{f_\theta(v_o)} >)$
Q2B [24]	$f_\theta(v_s), f_\theta(v_o) \in \mathbb{R}^d, f_\theta(r) \in \mathbb{R}^{2d}$	$\texttt{Dist}_{\text{out}} + \alpha \texttt{Dist}_{\text{in}}$

2.2 KG Reasoning Embeddings

KG reasoning embedding methods generalize the shallow KG embeddings to more complex reasoning tasks. KG reasoning embeddings also consider multi-hop reasoning over the KG, i.e., answering complex logical queries with logical/set operators including conjunction, disjunction and negation, e.g., "Predict drugs that might target proteins that are associated with a given disease, and do not have a given side effect" [10]. In order to answer such complex queries, one may need to perform multiple reasoning steps and graph traversal – first find all the proteins associated with the disease, and predict drugs $\mathcal{D}_1 \subset \mathcal{V}$ that bind with the proteins, at the same time find the drugs $\mathcal{D}_2 \subset \mathcal{V}$ that have the side effect and take complement of the set $\overline{\mathcal{D}_2}$, and finally take the intersection of the two sets $\mathcal{D}_1 \cap \overline{\mathcal{D}_2}$ to achieve the answers to the query. One of the main challenges of the above graph traversal method is that it suffers from missing and noisy information on the graph. The key insight of KG reasoning embedding methods is to embed these complex queries in the same latent space as the entity embeddings so that all the reasoning steps can be done in the embedding space instead of symbolic graph traversal. In detail, we follow the logical queries defined in **(author?)** [25].

Definition 2.1 (First-order logic queries) *A first-order logic query q consists of a non-variable anchor entity set $\mathcal{V}_q \subseteq \mathcal{V}$, existentially quantified bound variables $V_1, \ldots, V_k$ and a single target variable $V_?$, which provides the query answer. The disjunctive normal form of a logical query q is a disjunction of one or more conjunctions.*

$$q[V_?] = V_? \,.\, \exists V_1, \ldots, V_k : c_1 \vee c_2 \vee ... \vee c_n$$

1. *Each c represents a conjunctive query with one or more literals e. $c_i = e_{i1} \wedge e_{i2} \wedge \cdots \wedge e_{im}$.*
2. *e represents an atomic formula or its negation. $e_{ij} = r(v_a, V)$ or $\neg\, r(v_a, V)$ or $r(V', V)$ or $\neg\, r(V', V)$, where $v_a \in \mathcal{V}_q$, $V \in \{V_?, V_1, \ldots, V_k\}$, $V' \in \{V_1, \ldots, V_k\}$, $V \neq V'$, $r \in \mathcal{R}$.*

In order to reason over and embed such queries, one needs to consider the following operations. KG reasoning methods design neural logical operators that simulate their real counterparts. We refer the readers to [24] for more details.

1. **Relation Projection:** Given a set of entities $S \subseteq \mathcal{V}$ and relation type $r \in \mathcal{R}$, compute adjacent entities $\cup_{v \in S} A_r(v)$ related to S via r: $A_r(v) \equiv \{v' \in \mathcal{V} :\ (v, r, v') \in \mathcal{E}\}$.
2. **Intersection:** Given sets of entities $\{S_1, S_2, \ldots, S_n\}$, compute their intersection $\cap_{i=1}^n S_i$.
3. **Complement/Negation:** Given a set of entities $S \subseteq \mathcal{V}$, compute its complement $\overline{S} \equiv \mathcal{V} \setminus S$.
4. **Union:** Given sets of entities $\{S_1, S_2, \ldots, S_n\}$, compute their union $\cup_{i=1}^n S_i$.

Capturing Relational Context: While KG reasoning queries have been traditionally proposed to answer complex queries in the presence of incomplete KGs, here, we utilize them to learn vector representations of the entities

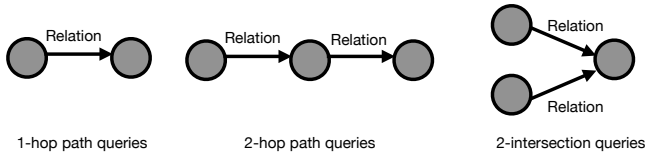


Figure 1: Types of queries we consider to train Query2box.

that are not biased towards one-hop relationships but take into account a richer relational context. We use a set of query templates (see Figure 1) to generate a sample workload of queries that can be answered over the input KG and use that payload to learn robust entity representations as we discuss next. We experimentally show (see Section 6) that this relational bias in the training process leads to entity representations that are more robust and lead to more stable representations (see Section 6).

Training KG Reasoning Embeddings: For a KG, $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$, and a query q, we need to learn an embedding function f_θ that maps from a computation graph of a query to its embedding with the parameterized neural logical operators. Together with the entity and relation embedding matrices (same as the shallow KG embeddings), f_θ also embeds all the nodes on the graph by embedding lookup. In order to measure the similarity/distance between a query q and an entity $v \in \mathcal{V}$, a distance function $\texttt{Dist}(\cdot, \cdot)$ is defined that takes as input the query embedding $f_\theta(q)$ and the entity embedding $f_\theta(v)$ and outputs the distance. The distance function $\texttt{Dist}(\cdot, \cdot)$ is tailored to different embedding space and model design f_θ as in Table 1.

During training, we are given a data sampler $\mathcal{D}$, each sample in $\mathcal{D}$ is a tuple $(q, \mathcal{A}_q, \mathcal{N}_q)$, which represents a query q, its answers $\mathcal{A}_q \subseteq \mathcal{V}$ and the negative samples $\mathcal{N}_q \subseteq \overline{\mathcal{A}_q}$. The training objective is to minimize the distance between the query embedding and its answers $\texttt{Dist}(q, v), v \in \mathcal{A}_q$ while maximizing the distance between the query embedding and the negative samples $\texttt{Dist}(q, v'), v' \in \mathcal{N}_q$, optimizing a contrastive loss term similar to the shallow KG embeddings. As used in most previous KG reasoning embedding works [25, 24], the loss is defined as:

$$\mathcal{L} = -\log \sigma \left(\gamma - \texttt{Dist}(q, v)\right) - \sum_{j=1}^{k} \frac{1}{k} \log \sigma \left(\texttt{Dist}(q, v'_j) - \gamma\right), \tag{47}$$

where γ is a margin hyperparameter and σ is the sigmoid function.

3 Fact Ranking

Here we introduce the task of fact ranking and its corresponding applications that power critical user experiences over KGs. We define a fact on KG as a triple (v_s, r, v_o), where v_s and v_o are entities and r is a relation type from the KG. User queries we target (such as the "What is the occupation of LeBron James?") correspond to queries of the form $q = (v_s, r, ?)$.

3.1 Problem Description

Intelligent assistants rely on entity-centric experiences to answer user queries. For many of these experiences, facts or answers to queries are of different importance/accuracy/uncertainty to users. Besides providing all the answers, one key aspect of service quality is to display relevant facts in a sorted way according to the importance or uncertainty score. For example, the answer to a query "What is the occupation of Selena Gomez?" includes singer and child actor, but for the majority of users, we know the answer singer should rank higher than child actor. The goal of fact ranking is to rank all the answers to a given set of queries that aligns well with user expectation. Fact ranking is ubiquitous in intelligent assistants and the key to improving the user experience.

We define fact ranking as follows: Given a query $q = (v_s, r, ?)$ for a subject v_s and a predicate r we see to find answers $\mathcal{A}_q$ (achieved by graph traversal as discussed in Section 2) for the missing object. The goal of fact ranking is to find a function $\texttt{Rank}(a)$ which can be used to obtain an importance ranking for each answer $a \in \mathcal{A}_q$ and generate the ranking list $[\texttt{Rank}(a_1), \ldots, \texttt{Rank}(a_n)]$, $a_1, \ldots, a_n \in \mathcal{A}_q$. We focus on queries that target facts that cannot be ranked using simple popularity scores, e.g., occupations, genres etc., (an occupation is not necessarily more important than the other).

Unsupervised machine learning (ML) mechanisms are needed to learn a function $\texttt{Rank}(\cdot)$. It is typical that KGs do not associate any importance scores and weights to their edges, which applies to many large KGs including FreeBase [1] and WikiData [32]. Consequently, it is not possible to use simple traversal mechanisms to implement the ranking functions for fact ranking and different mechanisms need to be considered. Such a mechanism may correspond to PageRank-based algorithms which can be used to assign an importance score for each answer a (or fact (q, a) with personalized PageRank), however, they are often not effective on such large-scale heterogeneous graphs where multiple edge types exist [12]. To alleviate these challenges, we consider a setting (see Section 3.2) where unsupervised representation learning is used to learn Function $\texttt{Rank}(\cdot)$.

3.2 A Solution with KG Embeddings

We obtain a solution to the fact ranking problem by leveraging graph embedding models. This solution applies to both shallow KG embeddings and KG reasoning embeddings. The idea is to first train the entity embeddings, relation embeddings, and neural logical operators on the KG using standard training protocols [37, 24] (see Section 2.2). Then, given a fact (v_s, r, v_o), we can use the pre-trained embeddings to efficiently calculate the distance $\texttt{Dist}(v_s, r, v_o)$. The distance plays a crucial role for solving the fact ranking problem since it represents a proxy of plausibility of a fact. This solution is inspired by our prior work on error detection, missing value imputation, and data repairs which showed that all problems correspond to inference tasks over a pre-trained model that learns how to reconstruct the input data [6]. Nonetheless, using the distance obtained by different KG embedding models raises two critical considerations for industrial settings.

For fact ranking, the distance obtained by the KG embedding model can be applied to rank the candidate objects for a specific subject and object configuration. However, different models can learn significantly different geometries and in most cases the distances in these spaces can lead to significant variations in the obtained rankings. In the settings we consider, it is critical that the rankings obtained are stable (i.e., we do not have significant variations in the order of different objects) across training iterations of the embedding model. To this end, we use a post-processing step that verifies the stability of KG embedding models before deployment. This post-processing step utilizes a consistency metric that extends standard ranking comparison methods such as Kendall's Tau to also consider the distance value associated with each query (see Section 4).

4 Consistency in Fact Ranking

We consider different ways to measure the stability of ranking across different training runs. Given a query $(v_s, r, ?)$ and its answers $\mathcal{A}_q = \{a_1, \ldots, a_n\}$, we calculate: $d_i = \texttt{Dist}(v_s, r, a_i), \forall i = 1, \ldots, n$, and create a distance list $\texttt{DistList} = [d_1, \ldots, d_n], d_i \in \mathbb{R}$ and a ranking list of the answers by the distance $\texttt{RankList} = [\texttt{Rank}(a_1), \ldots, \texttt{Rank}(a_n)], \texttt{Rank}(a_i) \in \{1, 2, \ldots, n\}$. We consider training KG embeddings multiple times, and obtain multiple $\texttt{DistList}$ and $\texttt{RankList}$. Our goal is to measure the stability/consistency of the lists across different runs. We assume the KG stays unchanged across different runs/training of the embedding models, hence the items in the list of a query also remain the same.

To measure the stability of ranking, i.e., compare whether two $\texttt{RankList}$ from two runs are consistent, we consider several metrics, including 1) the Kendall rank correlation coefficient, 2) a weighted version of Kendall's Tau, 3) set-based overlap, and 4) rank-biased overlap. Given two distance lists $\texttt{DistList}_1 = [x_1, \ldots, x_n]$ and

Algorithm 1: AdaptiveCluster

Input : A list of distance of answers $\texttt{DistList} = [x_1, \ldots, x_n]$, a scalar threshold δ' (hyperparameter).
Output : A list of cluster IDs `ClusterList`.

$\texttt{DiffList} = []$;
for $i \leftarrow 1$ **to** n **do**
  $\texttt{DiffList}$.append$(x_i - x_{i-1})$;
$\mu = \texttt{DiffList}$.mean(), $\sigma = \texttt{DiffList}$.std();
Threshold $\delta = \min(\mu - 0.2\sigma,\ \delta')$;
$\texttt{ClusterList} = [0]$, $clusterid = 0$;
for $i \leftarrow 0$ **to** $n - 1$ **do**
  if $\texttt{DiffList}[i] > \delta$ **then**
    $clusterid++$;
  $\texttt{ClusterList}$.append$(clusterid)$;
return $\texttt{ClusterList}$;

Algorithm 2: Adaptive Tau

Input : Two lists of distance of answers $\texttt{DistList}_1$, $\texttt{DistList}_2$, a scalar threshold δ' (hyperparameter).
Output : Kendall's Tau coefficient.

$\texttt{ClusterList}_1 = $ AdaptiveCluster$(\texttt{DistList}_1, \delta')$;
$\texttt{ClusterList}_2 = $ AdaptiveCluster$(\texttt{DistList}_2, \delta')$;
return KendallTau$(\texttt{ClusterList}_1, \texttt{ClusterList}_2)$;

$\texttt{DistList}_2 = [y_1, \ldots, y_n]$, the four metrics are calculated as:

1. Kendall's Tau: $\frac{m_c - m_d}{\binom{m}{2}}$, where m_c is the number of concordant pairs between $\texttt{DistList}_1$ and $\texttt{DistList}_2$, and m_d is the number of discordant pairs. A pair of (i, j) is concordant if the sort order of (x_i, x_j) and (y_i, y_j) is the same, otherwise the pair is discordant. Kendall's Tau ranges from -1 to 1.
2. Weighted Tau: It is an extension of Kendall's Tau where each pair also has a weight that is inverse-proportional to the rank, i.e., low ranking objects are not as important as the top ranking objects.
3. Rank-biased overlap (RBO): $(1 - p) \sum_{i=1}^{n} p^{i-1} \cdot A_i$, where i is the depth of the ranking being examined. With $ArgSort$ function, let $\texttt{ASList} = ArgSort(\texttt{DistList})$, we define $A_i = \frac{|\texttt{ASList}_1[:i] \cap \texttt{ASList}_2[:i]|}{i}$. The idea of RBO is to compare the overlap of the two rankings at incrementally increasing depths. It is a weighted metric, which means that the top rank items get higher weights.

However, the downside of the above metrics is that they do not explicitly consider the absolute value of items in `DistList`. One observation is that when two answers have similar distance with the query embedding, a swap in the ranking of the two answers from two runs should not matter as much as a swap in the ranking when the two answers have different distance to the query embedding. Consider the following two scenarios, assume in both scenarios, the length of the `DistList` is 3. In Scenario #1 we have $\texttt{DistList}_1 : [0.20, 0.30, 0.33]$ $\texttt{DistList}_2 : [0.45, 0.61, 0.60]$ and in Scenario #2 we have $\texttt{DistList}_1 : [0.20, 0.30, 0.63]$ $\texttt{DistList}_2 : [0.45, 0.61, 0.50]$.

Although in both scenarios, there exists one discordant pair (the second and third item), yet in Scenario #1, the two items have extremely close distance compared with Scenario #2. So an ideal metric would output a higher consistency score for Scenario #1 than Scenario #2. However, all above metrics give the same results.

To address the above shortcoming, we use an evaluation metric that adaptively considers the margins of different items when measuring the consistency of two `DistList`. In order to identify the items with close

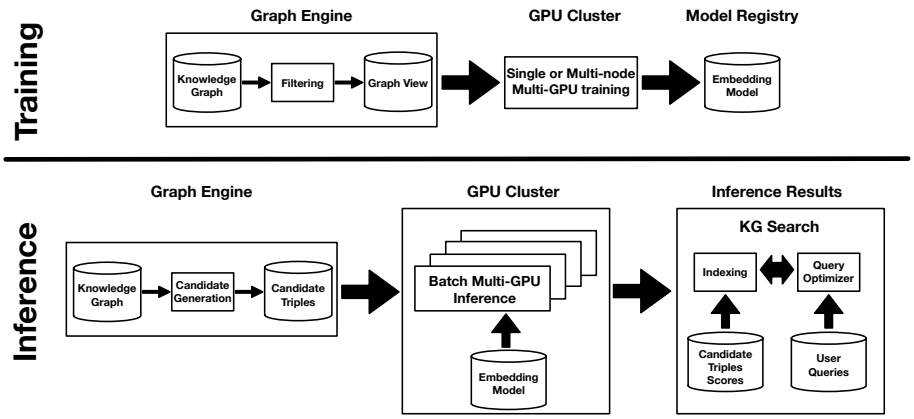


Figure 2: An overview of using Knowledge Graph Embedding models for large-scale fact ranking.

values, we sort the `DistList`, calculate the difference between neighbor items, and measure the average and variance, which we use to set as a threshold. Then, we loop over all the items and aim to cluster the item by checking whether the difference between the current item and the previous item is larger than the threshold. For items in the same cluster, we assign the same value to them such that they will have the same ranking. Finally, we run Kendall's Tau metric over this updated list. The details are shown in Algorithms 1 and 2. We refer to this method as Adaptive Tau since it considers the absolute value of the discordant pairs using clustering with an adaptive threshold. An experimental analysis of the different metrics is shown in Section 6. We find that Adaptive Tau provides a more precise description of the stability and utility of the rankings obtained by embedding models.

5 Scaling to Large KGs

We discuss systems considerations when using KG reasoning embedding models over large-scale KGs. Beyond scalable training, we also require that inference over Query2Box models is scalable and incremental. This requirement is important to enable practical deployments over dynamic billion-scale KGs. We next discuss the main components of the architecture we adopt (see Figure 2).

The multi-hop nature of embedding models such as Query2Box poses unique challenges when training these models over billion-scale KGs. In the case of shallow KG embedding models, graph partitioning is a common method for scaling training [40, 17]. Unfortunately, these methods are not applicable in the case of reasoning-based embeddings. When using Query2Box it is important that we can generate training samples by performing multi-hop traversals of the graph. Such traversals can span multiple partitions. At the same time, it is not always practical to pre-compute such samples in advance. To alleviate this issue, we opt for a single-machine multi-GPU deployment during training and leverage the recently introduced SMORE engine [23] to perform training. SMORE provides a mixed GPU-CPU solution that leverages both the main memory and GPU memory to scale training. In addition, training examples are generated on the fly thus avoiding unnecessary pre-compution. Indicative throughput measurements and scaling of SMORE is shown in Figure 3. A requirement here is that the machine have sufficient main and on-device memory to store the entire graph and thus avoid partitioning. While this requirement is satisfied by modern hardware configurations it is a cost-hungry option. We believe disk-based or distributed training of reasoning-based KG embeddings is an exciting research direction. Once training is complete, the embedding models are archived and then used for inference.

At inference time, we opt for a batch inference setting. We first compute a series of candidate queries that correspond to the set of facts that we want to enable ranking over. We leverage a computation graph engine to materialize all candidate queries (i.e., `(subject, predicate, object` triples) and use the learned Query2Box model to obtain a score for each query. The number of candidate facts can exceed the size of the

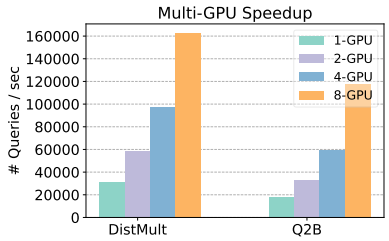


Figure 3: Multi-GPU scaling of SMORE for training the DistMult and Query2box embedding models.

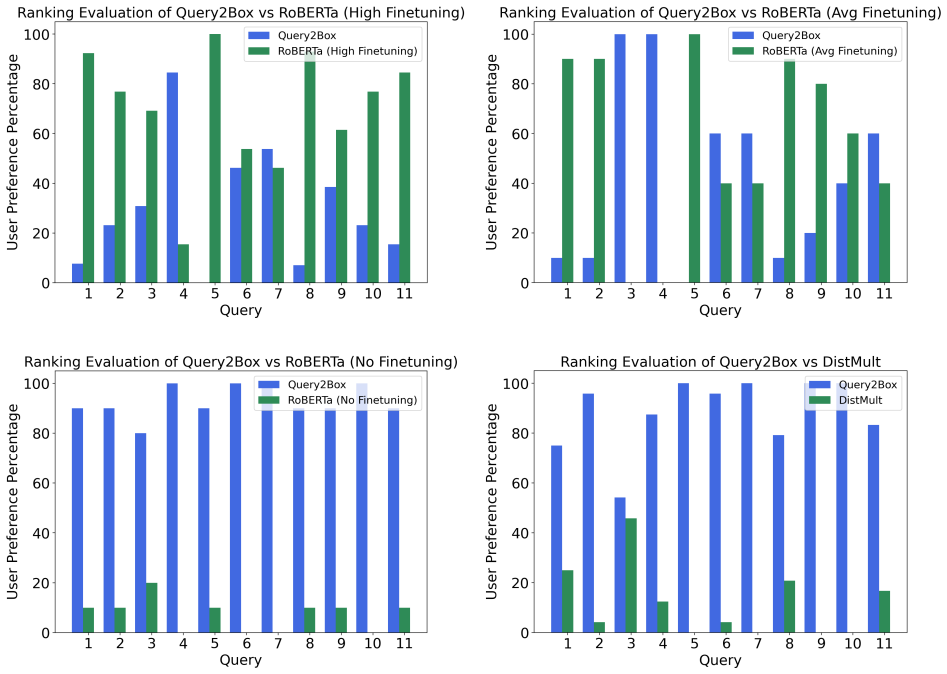


Figure 4: Fact ranking using Query2Box vs. RoBERTa and Query2Box vs. DistMult.

original KG as we consider multiple subject, object configurations that may not appear in the original graph. To deal with the volume of generated queries we opt for a scatter gather-based multi-GPU inference across multiple machines in a GPU cluster. The trained model is loaded in the executor allocated to each machine and the candidate facts are partitioned across machines. The corresponding inference results are gathered into a single relational store and then used for downstream processing. Given the stability of the Query2Box representations (see Section 6), we maintain the fact ranking results via periodic retraining of the Query2Box models followed by batch inference. Inference results for fact ranking are versioned across different training and batch inference runs.

6 Experiments

We evaluate reasoning-based KG embeddings for fact ranking. We focus on fact ranking tasks that align with use cases in industrial deployments. We evaluate the following aspects of our proposed framework: (1) the utility to end users when using KG embeddings for fact ranking, (2) the stability of KG embedding models and hence to what extent they satisfy deployment requirements.

Table 2: Average user preference (based on results in Figures 4) for Q2Box vs other methods for different tasks. For RoBERTa models, (H): high finetuning, (A): average finetuning, (N): no finetuning.

Query2Box	**DistMult**	**RoBERTa (N)**	**RoBERTa (A)**	**RoBERTa (H)**
1- TV Actor	1- Film Actor	1- Singer	1- Film Producer	1- Actor
2- Film Actor	2- Singer	2- Film Producer	2- Film Director	2- Singer
3- Film Director	3- Film Director	3- Film Director	3- Film Actor	3- Film Producer
4- Actor	4- Film Producer	4- Film Actor	4- TV Actor	4- Film Director
5- Film producer	5- Actor	5- Actor	5- Actor	5- Film Actor
6- Singer	6- TV Actor	6- TV Actor	6- Singer	6- TV Actor

Table 3: Average user preference (based on results in Figure 4) for Q2Box vs other methods for different tasks. For RoBERTa models, (H): high finetuning, (A): average finetuning, (N): no finetuning.

Comparison Task	**Competitor**	**Competitor Avg. Percentage**	**Q2Box Avg. Percentage**
DistMult / Q2Box	DistMult	12%	**88%**
RoBERTa (H) / Q2Box	RoBERTa (H)	**70%**	30%
RoBERTa (A) / Q2Box	RoBERTa (A)	**57%**	43%
RoBERTa (N) / Q2Box	RoBERTa (N)	7%	**93%**

Table 4: Stability results of fact ranking. Query2box consistently achieves better performance than DistMult.

	Kendall	Weighted Kendall	Set-based Overlap	Rank-biased Overlap	AdaptiveTau $\delta' = 0.02$	AdaptiveTau $\delta' = 0.05$	AdaptiveTau $\delta' = 0.1$
DistMult	0.380	0.384	0.962	0.990	0.460	0.498	0.484
Query2Box	**0.854**	**0.868**	**0.990**	**0.997**	**0.877**	**0.917**	**0.943**

6.1 Experiment Setup

Queries and Facts We focus on queries of the format "What is the occupation of `[Celeb_Name]`?" which captures the ranking task. We rank possible object completions for the structured query $(v_s, \texttt{OccupationOf}, ?)$, where v_s is the entity of interest. Our dataset contains several million queries obtained by real intelligent assistant user queries.

Knowledge Graph We consider the entire Wikidata KG [32] to validate the proposed framework. The version of Wikidata that we use contains 1,754,058,566 facts defined over 91,900,599 entities and 35,446 relation types. We train our framework on this KG and obtain the answers to the aforementioned set of user queries by finding entities in this KG.

Baselines We evaluate a diverse array of methods, including KG reasoning embeddings *Query2box* [24], shallow KG embeddings *DistMult* [37], and masked language models (MLMs) *RoBERTa* [18].

For KG embedding based methods, both DistMult and Query2box fit in our unified framework. We adopt the standard training procedures for these models (see Section 2). For DistMult, we sample existing edges as positive samples and non-existing edges as negative samples to train the DistMult model with the objective defined in equation 46, where the distance and residue functions are defined in Section 2. For Query2box, similar to [24] and as discussed in Section 2.2, we sample multi-hop queries (Figure 1), and their answers and non-answers to optimize the contrastive objective in equation 47. Besides the entity and relation embedding matrices, we use the neural logic operators in Query2box to embed the complex queries and optimize the query embeddings such that they are close to the answer embedding and pushed far away from the embedding of the sampled non-answers. We use the distance function of the original Query2box model and design the residue function as described in Section 2.

For both DistMult and Query2Box, we use SMORE [23] for training. We train both for 100k iterations with the Adam optimizer [16]. We anneal the learning rate from 0.001 to 0.0001, and adopt a batch size of 8,192 queries with 1,024 negative answers for each query in the batch. We score each candidate answer using the distance functions defined for both models.

For MLMs, we use the RoBERTa model. Given a triple-format query $(v_s, \texttt{OccupationOf}, ?)$, we provide three templates and convert the query into a natural language question. These templates include (1) "v_o is a [mask].", (2) "The occupation of v_o is [mask].", and (3) prompting [34] in which the template is "Barack Obama is a politician, LeBron James is a basketball player, v_o is a [mask].". For different query types, we need to provide different prompts in order to make predictions more effective. As we show later, fine-tuning of the prompt is necessary to obtain competitive results and hence, MLMs are not a universal solution to our task. Given v_o, we score the candidate answer by calculating the likelihood score of v_o in replacement of the [mask] in each of the three templates.

6.2 Fact Ranking Evaluation

Utility We first evaluate the utility of our framework on the fact ranking task. To assess the quality of Query2Box and compare it against other methods, we consider eleven celebrities and their occupations as listed in WikiData. We present our users with four different questionnaires. Each of these questionnaires compares Query2box ranking vs. another baseline ranking obtained using one of the methods we mentioned earlier. Each questionnaire contains eleven questions where each asks users to choose between two different rankings (Query2Box vs. a baseline) of a celebrity occupation.

Figure 4 shows the summary of user preferences between DistMult/RoBERTa and Query2box rankings. In Figure 4, Query2box has outperformed DistMult and Query No. 3 (occupations of Jennifer Lawrence) shows the tightest competition. Table 2 shows the ranking derived from these methods. Jennifer Lawrence is mostly known as a *Film Actor* which is correctly predicted by DistMult. Some users have focused on the first occupation and hence voted for DistMult while other users have considered other occupations and voted for Query2box. In addition, we can notice the flipping of user preferences based on the amount of fine-tuning for RoBERTa models (see also Table 3). As shown in Table 3 which represents the average user preference, Query2Box outperforms DistMult and RoBERTa requires significant fine-tuning of the prompt to outperform the Query2box model.

Stability We now measure the stability of different systems using the metrics we introduced in Section 3.2. We take all triples with OccupationOf as the relation type from the massive KG. Overall the dataset involves 6,566,224 queries of structure $(v_o, \texttt{OccupationOf}, ?)$ for which we measure the stability and consistency of rankings. Here we mainly consider two methods Query2box and DistMult, but not the MLMs due to their necessity of contexts for better ranking utility as discussed. We train both models 5 times and measure the stability of both models using the metrics introduced in Section 4. As shown in Table 4, we find Query2box is more stable and consistent than DistMult since Query2box is trained on more complex multi-hop queries, which better captures the neighborhood structure for each fact. For set-based overlap and rank-biased overlap, both methods achieve extremely high values. This is expected since the occupations of a celebrity are fixed across runs, and the overlap will always be 1 at the last step as we gradually compare the intersection of two sets starting from top-ranking items to the low-ranking ones. Among evaluation metrics, our adaptive method can better characterize a more meaningful measurement of ranking stability than the vanilla Kendall's Tau and rank-biased overlap. As shown in Table 4, Query2box achieves higher performance in AdaptiveTau than the other two metrics. We argue such an adaptive metric is crucial in evaluating ranking stability in production.

7 Use-case: Ranking for Related Entity Search

Recommendation generation is a key component of question answering in entity-centric user experiences, and the task of providing a ranked list of KG entities related to that of users' query can be performed via fact ranking with KGE models. Specifically, given a user query $q = (v_s, r, ?)$, the goal of related entity search is to find a ranking function $\mathsf{Rank}(v_r)$ over a subset KG of entities $v_r \in \mathcal{R} \subseteq \mathcal{V}$ such that the query $q_r = (v_r, r, ?)$ is relevant to the original query, and $\mathsf{Rank}(v_r)$ provides a ranking of each entity v_r based on relatedness to the original query. We leverage the KGE models described in the previous sections for embedding a KG into a vector space and define relatedness between two entities in a KG to be the similarity between their vector representations. Thus, we can use similarity search over KG embeddings to find related entities for a given KG entity. Depending on the specific application of related entity search, we can use different embedding models. As an example, if we are interested in relatedness in the ontology space of a KG, Poincaré embeddings [41] is a suitable model. Otherwise, if we care about relatedness in the whole graph we can use either a shallow (e.g., DistMult [37]) or reasoning-based embedding model.

In addition to the use of KGE-based fact ranking for similarity based relatedness, the task of finding a ranked list of related KG entities for a query requires evaluating additional constraints for aligning answers with users' search intent. For instance, for the query "How tall is LeBron James", the goal is to find other "Person" entities that are related to "Lebron James" and have the corresponding fact for the same predicate "height". Consequently, related entity search use-case goes beyond the traditional vector similarity search and requires batch processing of hybrid queries [42]. Hybrid queries are two part queries consisting of: (i) vector similarity search for retrieving the most similar entities in the embedding space; and (ii) evaluation of conjunction of relational constraints for ensuring the returned results are relevant to search context (e.g., only include "Person" entities). In addition to hybrid query processing, the task of related entity search exhibits following characteristics: (i) hybrid queries are evaluated in a **batch setting** over past user queries, (ii) and relational predicates in industrial KG workloads exhibit filter commonality and filter stability [28], allowing us to customize the system design based on **available prior workload** characteristics. To this end, we employ HQI [42] hybrid vector similarity search system for batch inference over KG embeddings and adopt the following suite of optimizations for *high-throughput batch processing of hybrid queries*:

Workload-aware vector index: Specialized vector indexes that either partition the data or form multi-level indexes over centroids are commonly used in vector databases to speed-up vector similarity search [33]. HQI utilizes the past workload information to guide the partitioning of the vectors in the underlying index in a way that hybrid queries can be answered by accessing as few partitions as possible. By extending the concept of query-data routing trees (qd-trees) [36] to vector databases, HQI considers both vectors and relational predicates from a hybrid query workload when generating physical data layout at data loading time. The resulting data layout partitions the vectors using the distribution of the attributes associated with vectors, the attribute constraints, and similarity of vectors present in the hybrid query workload. We then use the resulting partitioning scheme to generate an index layout that enables processing a batch workload of hybrid queries by accessing vectors from as few partitions as possible.

Batch query optimization: Second, we use HQI's a multi-query optimization technique that (i) batches queries with similar attribute and vector similarity constraints; and (ii) performs batch vector distance computation against a posting list of vectors obtained from a clustering-based index over the vectors. This optimization is motivated by the fact that the set of candidate queries are computed from past user queries and evaluated in a batch setting, which enables computation sharing across queries. Note that this optimization is orthogonal to the workload-aware vector index and is applicable to any clustering-based vector index.

We evaluate the performance improvements of these optimizations for related entity search over KG. We use a subset of KG entity embedding vectors, and we focus on ranking related entities for queries of the format "What is the `[Predicate]` of `[Entity_Name]`?", similar to Section 6. Table 5 compares the performance of our solution against available existing hybrid query processing strategies (see [42] for more details) using a randomly

Table 5: Slowdown for related entity search compared to HQI @ Recall >= .8

	HQI	PreFilter	PostFilter	Range
Slowdown	1×	31×	136×	NA

sampled and aggregated query workload from anonymized, historical queries. HQI and its optimizations provide orders of magnitude performance improvements over best performing baselines for the related entity search task.

8 Conclusion

In this work, we studied fact ranking over large-scale knowledge graphs. We evaluated to what extent modern knowledge graph embedding (KGE) models provide a solution for addressing the problem of fact ranking. We highlighted unique challenges associated with solving this task in industrial settings and evaluated different KGE and text-based embedding models. Our work demonstrated that, in contrast to neural language models or shallow KGE models, multi-hop reasoning models such as Query2Box can better meet user satisfaction.

References

[1] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. SIGMOD, 2008.

[2] A. Bordes, N. Usunier, A. Garcia-Durán, J. Weston, and O. Yakhnenko. Translating Embeddings for Modeling Multi-Relational Data. Neural Information Processing Systems, 2787—2795, 2013.

[3] S. Bouraga, I. Jurerta, S. Faulkner, and C. Herssens. Knowledge-based recommendation systems: a survey. International Journal of Intelligent Information Technologies, 10(2):1–19, 2014.

[4] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Re, and K. Murphy. Machine learning on graphs: A model and comprehensive taxonomy. arXiv preprint, arXiv:2005.03675, 2020.

[5] W. Cohen. TensorLog: A Differentiable Deductive Database. arXiv preprint, arXiv:1605.06523, 2016.

[6] C. De Sa, I. Ilyas, B. Kimelfeld, C. Ré, and T. Rekatsinas. A formal framework for probabilistic unclean databases. arXiv preprint, arXiv:1801.06750, 2018.

[7] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 601–610, 2014.

[8] D. Xin, and T. Rekatsinas. Data Integration and Machine Learning: A Natural Synergy. ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019.

[9] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, K. Murphy, S. Sun, and W. Zhang. From Data Fusion to Knowledge Fusion. VLDB, 7(10):881–892, 2014.

[10] W. Hamilton, P. Bajaj, M. Zitnik, D. Jurafsky, and J. Leskovec. Embedding Logical Queries on Knowledge Graphs. Neural Information Processing Systems, 2018.

[11] A. Heidari, G. Michalopoulos, S. Kushagra, I. Ilyas, and T. Rekatsinas. Record fusion: A learning approach. arXiv preprint, arXiv:2006.10208, 2020.

[12] H. Huang, L. Sun, B. Du, C. Liu, W. Lv, and H. Xiong. Representation Learning on Knowledge Graphs for Node Importance Estimation. ACM SIGKDD Conference on Knowledge Discovery & Data Mining, 646-655, 2021.

[13] I. Ilyas, T. Rekatsinas, V. Konda, J. Pound, X. Qi, and M. Soliman. Saga: A Platform for Continuous Construction and Serving of Knowledge At Scale. ACM SIGMOD International Conference on Management of data, 2022.

[14] I. Ilyas, and X. Chu. Data Cleaning. Morgan & Claypool, 2019.

[15] S. Ji, S. Pan, E. Cambria, P. Marttinen, and S.Y. Philip. A survey on knowledge graphs: Representation, acquisition, and applications. IEEE Transactions on Neural Networks and Learning Systems, 2021.

[16] D. Kingma, and J. Ba. Adam: A method for stochastic optimization. International Conference on Learning Representations (ICLR), 2015.

[17] A. Lerer, L. Wu, J. Shen, T. Lacroix, L. Wehrstedt, A. Bose, and A. Peysakhovich. Pytorch-biggraph: A large-scale graph embedding system. Conference on Machine Learning and Systems (MLSys), 2019.

[18] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. arXiv preprint, arXiv:1907.11692, 2019.

[19] J. Mohoney, R. Waleffe, H. Xu, T. Rekatsinas, S. Venkataraman. Marius: Learning Massive Graph Embeddings on a Single Machine. USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2021.

[20] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor. Industry-Scale Knowledge Graphs: Lessons and Challenges. Queue, 17(2):48–75, 2019.

[21] J. Pujara, H. Miao, L. Getoor, and W. Cohen. Knowledge graph identification. International semantic web conference, 542-557, 2013.

[22] T. Rekatsinas, X. Chu, I. Ilyas, and C. Ré. HoloClean: Holistic Data Repairs with Probabilistic Inference. VLDB Endowment, 10(11):1190–1201, 2017.

[23] H. Ren, H. Dai, B. Dai, X. Chen, D. Zhou, J. Leskovec, and D. Schuurmans. SMORE: Knowledge Graph Completion and Multi-hop Reasoning in Massive Knowledge Graphs. arXiv preprint, arXiv:2110.14890, 2021.

[24] H. Ren, W. Hu, and J. Leskovec. Query2box: Reasoning over Knowledge Graphs in Vector Space using Box Embeddings. International Conference on Learning Representations (ICLR), 2020.

[25] H. Ren, and J. Leskovec. Beta Embeddings for Multi-Hop Logical Reasoning in Knowledge Graphs. Neural Information Processing Systems (NeurIPS), 2020.

[26] A. Rossi, D. Barbosa, D. Firmani, A. Matinata, P. Merialdo. Knowledge graph embedding for link prediction: A comparative analysis. ACM Transactions on Knowledge Discovery from Data, 15(2)1–49, 2021.

[27] Z. Sun, Z. Deng, J. Nie, and J. Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. International Conference on Learning Representations (ICLR), 2019.

[28] L. Sun, M.J. Franklin, S. Krishnan, and R.S. Xin. Fine-grained partitioning for aggressive data skipping, ACM SIGMOD International Conference on Management of Data, 1115-1126, 2014

[29] Z. Sun, S. Vashishth, S. Sanyal, P. Talukdar, and Y. Yang. A re-evaluation of knowledge graph completion methods. arXiv preprint, arXiv:1911.03903, 2019.

[30] P. Tabacof and L. Costabello. Probability Calibration for Knowledge Graph Embedding Models. International Conference on Learning Representations (ICLR), 2020.

[31] T. Trouillon, J. Welbl, S. Riedel, E. Gaussier, G. Bouchard. Complex embeddings for simple link prediction. International Conference on Machine Learning (ICML), 2016.

[32] D. Vrandečić, and M. Krötzsch. Wikidata: A Free Collaborative Knowledgebase. Communications of the ACM, 57(10):78–85, 2014.

[33] J. Wang, X. Yi, R. Guo, H. Jin, P. Xu, S. Li, X. Wang, X. Guo, C. Li, X. Xu, and others. Milvus: A purpose-built vector data management system. ACM SIGMOD International Conference on Management of Data, 2614–2627, 2021.

[34] C. Wei, S.M. Xie, and T. Ma. Why Do Pretrained Language Models Help in Downstream Tasks? An Analysis of Head and Prompt Tuning. Advances in Neural Information Processing Systems, 2021.

[35] G. Weikum. Knowledge graphs 2021: a data odyssey. VLDB Endowment, 14(12):3233–3238, 2021.

[36] Z. Yang, B. Chandramouli, C. Wang, J. Gehrke, Y. Li, U.F. Minhas, P. Larson, D. Kossman, and R. Acharya. Qd-Tree: Learning Data Layouts for Big Data Analytics. ACM SIGMOD International Conference on Management of Data, 193–208, 2020.

[37] B. Yang, W. Yih, X. He, J. Gao, L. Deng. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. International Conference on Learning Representations (ICLR), 2015.

[38] J. You, X. Ma, Y. Ding, M. Kochenderfer, and J. Leskovec. Handling missing data with graph representation learning. Advances in Neural Information Processing Systems (NeurIPS), 2020.

[39] S. Zhang, Y. Tay, L. Yao, Q. Liu. Quaternion knowledge graph embeddings. Advances in Neural Information Processing Systems (NeurIPS), 2019.

[40] Z. Zhu, S. Xu, J. Tang, and M. Qu. Graphvite: A high-performance cpu-gpu hybrid system for node embedding. The World Wide Web Conference (WWW), 2019.

[41] M. Nickel, and D. Kiela. Poincaré embeddings for learning hierarchical representations Advances in neural information processing systems (NeurIPS), 2017.

[42] J. Mohoney, A. Pacaci, S.R. Chowdhury, A. Mousavi, I. Ilyas, U.F. Minhas, J. POund, T. Rekatsinas High-Throughput Vector Similarity Search in Knowledge Graphs ACM SIGMOD/PODS International Conference on Management of

Data (SIGMOD), 2023.

Graph Data Augmentation for Graph Machine Learning: A Survey

Tong Zhao[1,4], Wei Jin[2], Yozen Liu[1], Yingheng Wang[3], Gang Liu[4],
Stephan Günnemann[5], Neil Shah[1], Meng Jiang[4]
[1]Snap Inc., [2]Michigan State University, [3]Cornell University,
[4]University of Notre Dame, [5]Technical University of Munich
[1]{tzhao,yliu2,nshah}@snap.com,[2]jinwei2@msu.edu,[3]yw2349@cornell.edu,
[4]{gliu7,mjiang2}@nd.edu,[5]guennemann@in.tum.de

Abstract

Data augmentation has recently seen increased interest in graph machine learning given its demonstrated ability to improve model performance and generalization by added training data. Despite this recent surge, the area is still relatively under-explored, due to the challenges brought by complex, non-Euclidean structure of graph data, which limits the direct analogizing of traditional augmentation operations on other types of image, video, or text data. Our work aims to give a necessary and timely overview of existing graph data augmentation methods; notably, we present a comprehensive and systematic survey of graph data augmentation approaches, summarizing the literature in a structured manner. We first introduce three different taxonomies for categorizing graph data augmentation methods from the data, task, and learning perspectives, respectively. Next, we introduce recent advances in graph data augmentation, differentiated by their methodologies and applications. We conclude by outlining currently unsolved challenges and directions for future research. Overall, our work aims to clarify the landscape of existing literature in graph data augmentation and motivates additional work in this area, providing a helpful resource for researchers and practitioners in the broader graph machine learning domain. Additionally, we provide a continuously updated reading list at https://github.com/zhao-tong/graph-data-augmentation-papers.

1 Introduction

Data driven inference has received a significant boost in generalization capability and performance improvement in recent years from data augmentation (DA) techniques. DA techniques increase the amount of training data by creating plausible variations of existing data without additional ground-truth labeling efforts, and have seen widespread adoption in fields such as computer vision (CV) [15] and natural language processing (NLP) [26]. These techniques allow machine learning models to learn to generalize across those variations and attend to signal over noise. In recent years, with the rapid development of graph machine learning (GML) methods such as graph neural networks (GNNs) [57, 38], studies have shown that the effectiveness of GML approaches also largely depends on the data quality. Given the dependent nature of graph data and the message-passing design of most GNNs, GML faces unique challenges such as: structural data sparsity brought by power-law degree distributions in most graphs, noisy and even erroneous topology brought by imperfect construction of the graph structure from raw data under other formats, low quality and incomplete node attributes, adversarial attacks on structure and attributes, lack of labelled data due to costly human annotations, and over-smoothing caused by the message passing design in GNNs. As DA allows researchers to alleviate such challenges from a data perspective, there has been increased interest and demand for such techniques on graph data [140], and there has been a growing number of works on graph data augmentation (GDA).

With the irregular and non-Euclidean structure of graph data, it is non-trivial to directly analogize DA techniques from CV and NLP to the graph domain, except for the most basic operations such as random masking/dropping/cropping. To better promote the effectiveness of GML approaches and alleviate the unique challenges in GML, recent literature designed graph-specific augmentation techniques following methodologies such as graph structure learning, graph adversarial training, graph rationalization, etc. Creating a unified taxonomy for all GDA techniques is not intuitive as they can be categorized under different facets. For example, taking the data modelity that the augmentation methods work on, they can be separated into structure augmentations, feature augmentations, and label augmentations. On the other hand, the focusing downstream tasks (i.e., node-level, edge-level, and graph-level tasks) can also categorize the GDA techniques. Moreover, the GDA methods can also be separated by whether the methods involves learning during the augmentation process. That is, whether they are rule-based approaches or learned approaches.

This paper aims to sensitize the GML community towards this growing area of work, as DA has already drawn much attention in CV and NLP [15, 26]. As interest and work on this topic continue to increase, this is an opportune time for a comprehensive work to (i) introduce background and motivation of GDA, (ii) give a bird's-eye view of existing GDA techniques under different taxonomies, (iii) introduce representative GDA techniques with their usage and applications, and (iv) identify key challenges to effectively motivate and orient interest in this area. We hope this survey can serve as a guide for researchers and practitioners who are new to or interested in studying this topic, and also inspire future research in this area.

The text is structured as follows: Section 2 gives background and motivation on GNNs and GDA. It defines GDA and motivates its use in GML tasks. Section 3 categorizes GDA techniques based on three different taxonomies: the operated graph data, the downstream tasks, and whether the method involves learning. Section 4 describes rule-based GDA techniques for GML – which we partition into Data Removal (Section 4.1), Data Addition (Section 4.2), and Data Manipulation (Section 4.3) focuses. Similarly, Section 5 introduces learned GDA techniques, which are further categorized by their methodologies: Graph Structure Learning (Section 5.1), Graph Adversarial Training (Section 5.2), Graph Rationalization (Section 5.3), and Automated Augmentation (Section 5.4). Section 6 introduces GDA techniques that are used under three different self-supervised learning objectives: Contrastive Learning (Section 6.1), Non-contrastive Learning (Section 6.2), and Consistency Training (Section 6.3). Finally, Section 7 discusses challenges and future directions for GDA.

2 Preliminaries

2.1 Notations

Let $G = (\mathcal{V}, \mathcal{E})$ be a graph of N nodes, where $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$ is the set of N nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of links. We denote the adjacency matrix as $\mathbf{A} \in \{0, 1\}^{N \times N}$, where $A_{i,j} = 1$ indicates nodes v_i and v_j are connected and vice versa. We denote the node feature matrix as $\mathbf{X} \in \mathbb{R}^{N \times F}$, where F is the number of raw node features and $\boldsymbol{x}_i$ indicates the feature vector of node v_i (the i-th row of $\mathbf{X}$). We use $\boldsymbol{y}$ to denote the label each sample, which can be node, edge, or graph depending on the task. We use symbol with tilde to denote the data generated by GDA methods. For example, $\tilde{\mathbf{A}}$ for the augmented adjacency matrix, $\tilde{\boldsymbol{x}}_i$ for the augmented feature vector of node v_i, etc.

2.2 Graph Neural Networks

Graph neural networks (GNNs) enjoy widespread use in modern graph-based machine learning due to their flexibility to incorporate node features, custom aggregations, and inductive operation, unlike earlier works which were based on embedding lookups [87, 34]. Following the initial idea of convolution based on spectral graph theory [6], many spectral GNNs have since been developed and improved by [19, 57, 67, 59, 80]. As spectral GNNs generally operate (expensively) on the full adjacency, spatial-based methods which perform

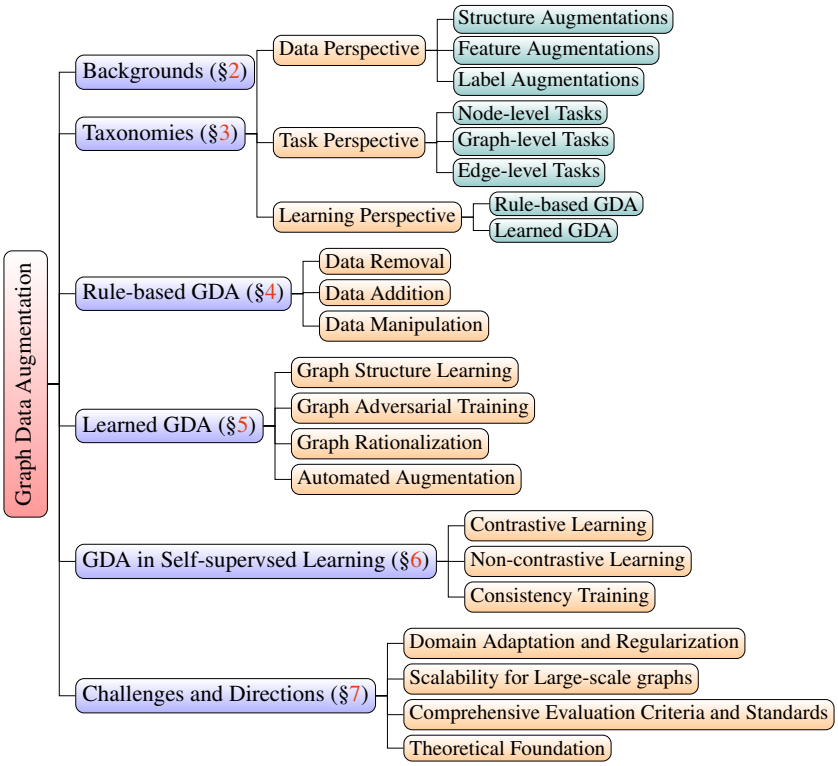


Figure 1: Structure of this survey.

graph convolution with neighborhood aggregation became prominent [38, 107], owing to their scalability and flexibility [128, 121].

Generally, the generic formulation of message passing-based GNNs can be defined by an aggregation function (AGGREGATE) and an update function (UPDATE). In each layer, AGGREGATE aggregates the embeddings from previous layer for each node from all its neighbors, and UPDATE updates each node's embedding by combining its own previous embedding and the aggregated neighbor embeddings [38]. Specifically,

$$\begin{aligned} \boldsymbol{h}_{\mathcal{N}(v)}^{l} &= \text{AGGREGATE}(\{\boldsymbol{h}_{u}^{l-1} | u \in \mathcal{N}(v)\}), \\ \boldsymbol{h}_{v}^{l} &= \text{UPDATE}(\boldsymbol{h}_{v}^{l-1}, \boldsymbol{h}_{\mathcal{N}(v)}^{l}), \end{aligned} \tag{48}$$

where $\boldsymbol{h}_v^l$ denotes the representation of node v at the l-th layer, and $\mathcal{N}(v)$ denotes the set of node v's neighbors.

In implementation, GNNs can usually be implemented with (sparse) matrix multiplications. Without the loss of generality, here we take the most commonly used Graph Convolutional Network (GCN) [57] as an example. One layer of GCN is defined as

$$\mathbf{H}^{l} = \sigma(\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{W}^{l}\mathbf{H}^{l-1}), \tag{49}$$

where $\mathbf{D}$ is the diagonal degree matrix s.t. $\mathbf{D}_{i,i} = \sum_j \mathbf{A}_{i,j}$ (assuming $\mathbf{A}$ contains self-loops), $\sigma(\cdot)$ is the nonlinear activation function such as ReLU, and $\mathbf{W}^l$ denotes the learnable weight matrix at the l-th GNN layer. Furthermore, we use $g_\Theta(\cdot)$ to denote the mapping function of the whole GNN model parameterized by Θ.

2.3 Graph Data Augmentation

The DA area encompasses techniques of increasing/generating training data without directly collecting or labeling more data. Most DA techniques either add slightly modified copies of existing data, or generate synthetic data

based on existing data. The augmented data act as a regularizer and reduce overfitting when training data-driven models [93]. DA techniques has been commonly used in CV [15] and NLP [26], where augmentation operations such as cropping, flipping, and back-translation are usually used in machine learning model training.

In GML, in contrast to regular and Euclidean data such as grids (e.g., images) and sequences (e.g., sentences), the graph structure is encoded by node connectivity, which is non-Euclidean and irregular. Most structured augmentation operations used frequently in CV and NLP cannot be easily analogized to graph data. Therefore, how to design effective augmentations of graph data is less obvious. For example, the data objects for node-level and edge-level tasks are inter-connected and non-i.i.d, meaning that GDA techniques typically modify the entire dataset (graph) instead of a specific data object (nodes or edge) in isolation. Generally, a GDA method can be defined as a transformation function $f : G \rightarrow \tilde{G}$, where the the transformation function f can be either rule-based or learnable, and the augmented graph $\tilde{G}$ contains the augmented adjacency matrix $\tilde{\mathbf{A}}$ and node feature matrix $\tilde{\mathbf{X}}$ (and optionally augmented edge features, node or graph labels). Moreover, the augmentation function f is not necessarily deterministic. That is, the same f may generate multiple different versions of the augmented graph $\tilde{G}$, and the model may use one or multiple of these augmentations as required for training.

2.4 Motivation: Why Augment Graphs?

Graphs are often utilized to model or represent an underlying process of relationships or affinities; for example, "which individuals are friends with one another?" or "which movies do individuals like?" In some cases, these relationships are strictly defined and known, e.g. researchers jointly co-authoring articles, or atoms interacting in a chemical compound. However, in many other cases, an "observed" graph may be misaligned with the true process it intends to model for a variety of reasons [5]. In some cases, like in social interaction graphs, noise may be inadvertently or adversarially introduced by spammers who pollute underlying data about authentic interactions with inauthentic ones for nefarious purposes [90, 65]. In other cases, noise may be inherently created by limited or partial observation (e.g. a movie recommendation system never recommending a certain genre of movies to a group of users) caused by privacy reasons [14, 22], biased recommendation policies [55, 143], or other reasons. Noise can also occur by measurement or thresholding errors (e.g. discretizing continuous signals between brain voxels into discrete ones) [30], or human errors (e.g. a person forgetting to add a known contact to their phone's contact-book). All of these scenarios can introduce gaps between an intended and observed graph. Moreover, even if all relationships a graph intends to capture are observed properly, there is no guarantee that the graph is a particularly useful [5] for a particular downstream learning task, especially when utilized in a GML context, e.g. a graph connecting individuals by similar heights may be unhelpful in regressing income.

GDA methods offer an attractive solution in denoising, imputing, and generally enhancing graph structure to align better with an intended modeling processes, or objectives of a target learning task [140]. Adding or removing nodes and edges can help connect or disconnect a graph to facilitate its use towards targeted objectives. Moreover, utilizing heuristic graph modification strategies to increase model exposure in training may lead to better generalizing, more robust and higher performance models [114, 62, 147]. Both learned and rule-based DA techniques have shown immense potential in other domains like tabular ML (e.g. oversampling [2] and SMOTE [7]), CV (e.g. rotations, flips and translations of images [93] and random erasure [146]) and NLP (e.g. synonym replacement and random token additions/deletions [118] and back-translation [89]); however, as aforementioned, these techniques usually lack clear analogs in the graph domain due to unclear correspondence of their label-preserving transforms. This lack of clarity motivates work into understanding the limitations of graphs, suitable designs for augmentation techniques, and their breadth of impact.

3 Taxonomies

In this section, we introduce three different taxonomies that can be used to categorize GDA techniques. They come from different perspectives of data, task, and learnability, respectively. As these taxonomies are orthogonal to each other, and each of them can in some way categorize all GDA methods, we will only focus on one taxonomy (rule-based vs. learned augmentation) for the later sections.

3.1 Operated Data Modality

As GDA methods all operate on graph data, they can naturally be categorized by the data modality that they aim to manipulate. Therefore, one intuitive taxonomy for GDA methods would be classifying them into one or more of three categories: structure, feature, and label augmentations.

Structure Augmentations are the GDA operations that modify the graph connectivity via adding/removing edges or adding/removing nodes from the graph. The modifications can be either deterministic (e.g., GDC [60] and GAug-M [140] both modify the graph structure and used the modifed graph for training/inferencing) or stochastic (e.g., DropEdge[88] and DropNode [27] randomly drop edges/nodes from the observed training graph). **Feature Augmentations** are the GDA operations that modify or create raw node features. For example, You et al. [129] used Attribute Masking that randomly masked off node features; FLAG [62] augments node features with gradient-based adversarial perturbations. It's worth noting that stucture augmentations and feature augmentations are also sometimes combined in some GDA methods. For example, MoCL [99] substitutes subgraphs in molecular graphs with subgraphs of different functional groups. **Label Augmentations** are the GDA operations that involves modifying the labels. For example, Mixup-based methods [39, 37] interpolate existing training examples and assign new label for the generated example. Counterfactual data augmentation methods (e.g., CFLP [143]) generate counterfactual examples with corresponding new labels.

3.2 Downstream Tasks

Another straightforward taxonomy of categorizing GDA methods is by the downstream tasks that they tackle. Generally, most GML methods can be categorized into three high-level task types: **node-level**, **edge-level**, and **graph-level** tasks. Similarly, many GDA methods are designed toward one of these tasks, and cannot be easily generalized to other tasks. For example, CFLP [143] generates counterfactual links as augmented data specifically for training a neural link predictor, and these counterfactual links are useless to other tasks like node classification as they are counterfactual labels on node pairs under specific treatments. Moreover, certain GDA methods that are designed for molecular graphs (e.g., MoCL [99]) are not opeartable on the large graph datasets used in other tasks as they rely on the domain specific substructures of molecular graphs, e.g., functional groups. Nonetheless, the downside of categorizing by downstream tasks is that a fair number of GDA methods were designed more generically for various tasks; for example, DropEdge [88] simply conducts random edge dropping during training, and the method can naturally be applied on most GML methods.

3.3 Rule-based vs. Learned Augmentations

GDA methods can also be categorized by whether the augmentation process involved learning, namely rule-based GDA approaches and learned GDA approaches. More specifically, **rule-based GDA approaches** refer to the non-learnable methods that modify or manipulate the graph data following pre-defined rules, which can be stochastic, deterministic, or mixture of both. A rule-based GDA method can be as simple as randomly removing a given fraction of edges [88] or randomly cropping out part of the graph [129]; it can also be more complicated such as counterfactual augmentation [143] based on similarity matching rules and graph diffusion methods [60] that follows specific diffusion kernels. We also categorize Mixup-based augmentations [39] as rule-based approaches

since they usually only contain one non-learnable parameter (sampled from pre-defined distributions) when generating new data objects by interpolating two existing data objects.

On the other hand, **learned GDA approaches** refer to the augmentation methods that contains learnable parameters in the process of generating augmented examples. The augmentation module can either be trained independently or in an end-to-end style with the downstream classifier or regressor [140]. For example, graph structure learning methods [152, 50, 140] often assume the observed graph data is noisy, incomplete, or entirely missing, so they first try to learn the "clean" graph structure before using it in the training and inference of GNNs. Graph rationalization methods [120, 71] learn subgraphs that are likely to be causally related with the graph labels and use them for augmentation. Automated augmentation methods [144, 79] utilize reinforcement learning agents to learn the optimal augmentation strategy for the given data automatically.

In Sections 4 and 5, we will introduce GDA approaches in more detail based on this separation as it provides better differentiation of the methodologies and improved readability. Table 1 shows a summary of GDA techniques, categorized following this taxonomy and the methods' methodologies.

4 Rule-based Approaches for GDA

Owing to their simplicity and efficiency, rule-based graph data augmentation methods are the most commonly used augmentation techniques in graph machine learning. The rule-based GDA approaches can generally categorized into three categories, where the first category of methods would remove part of the data (e.g., Stochastic Masking) to create new graph data, the second category of methods augments the graph data by generating new graphs or adding components (e.g., Counterfactual Augmentation, Pseudo-labeling), and the third category includes methods that manipulate the data following rules can involve both removing and adding operations (e.g., Diffusion, etc.) In the following subsections, we summarize the representative approaches in each category and also discuss their applications on different tasks and domains.

4.1 Data Removal

Edge Dropping. Edge dropping methods stochastically remove a certain fraction of edges from the graph data. Aiming to alleviate the known over-smoothing problem of GNNs, Rong et al. [88] first proposed DropEdge which randomly dropped a fixed fraction of edges in each training epoch, resembling Dropout [96]. More specifically, at the beginning of each training epoch, the modified adjacency matrix $\tilde{\mathbf{A}}$ is defined by

$$\tilde{\mathbf{A}} = \mathbf{M} \odot \mathbf{A}, \tag{50}$$

where $\mathbf{M} \in \{0,1\}^{N \times N}$ is a binary mask on the adjacency matrix s.t. $M_{i,j} = Bernoulli(\varepsilon)$, $\varepsilon \in (0,1)$ is the drop rate hyper-parameter, and $\odot$ denotes the Hadamard product.

During GNN training, DropEdge adopts a newly sampled $\tilde{\mathbf{A}}$ instead of the original graph structure $\mathbf{A}$ for message passing (e.g., Equation equation 49) in each training epoch. By showing the GNN models different perturbations of the graph in each training epoch, DropEdge improves the model's generalization and shows significant performance improvements on deeper GNNs, indicating that the strategy mitigates over-smoothing. Several other methods [129, 102, 144] also adopt random edge masking in other learning schemes such as self-supervised learning, which conducts the same operation as DropEdge.

Node Dropping. Similar to edge dropping, node dropping methods stochastically remove nodes from the graph. Node dropping is typically implemented in two ways: removing all features of the target nodes from the feature matrix, or removing the target nodes along with all the edges connected with them from the graph structure. Feng et al. [27] proposed DropNode, which follows the first schema. Concurrently, You et al. [129] proposed NodeDropping following the latter.

Table 1: A summary of GDA techniques, categorized by whether they are learned augmentations and their methodologies.

	Methodology	Representative Works	Task Level			Augmented Data		
			Node	Graph	Edge	Structure	Feature	Label
Rule-based GDA	Stochastic Dropping/Masking	DropEdge [88]	✓			✓		
		DropNode [27]		✓			✓	
		NodeDropping [129]		✓		✓		
		Feature Masking [102]	✓				✓	
		Feature Shuffling [108]	✓				✓	
		DropMessage [23]	✓		✓		✓	
		Subgraph Masking [129]		✓		✓	✓	
	Subgraph Cropping/Substituting	GraphCrop [113]		✓		✓		
		M-Evolve [147]		✓		✓		
		MoCL [99]		✓		✓	✓	
	Virtual Node	Graphormer [127]		✓		✓		
		GNN-CM$^+$/CM [45]			✓	✓		
	Mixup	Graph Mixup [117]	✓	✓				✓
		ifMixup [37]		✓		✓	✓	✓
		Graph Transparent [86]		✓		✓	✓	✓
		G-Mixup [39]		✓		✓	✓	✓
	SMOTE	GraphSMOTE [142]	✓				✓	
		GATSMOTE [76]	✓			✓		
		GNN-CL [70]	✓			✓	✓	
	Diffusion	GDA [60]	✓			✓		
	Counterfactual Augmentation	CFLP [143]			✓	✓		✓
	Attribute Augmentation	LA-GNN [75]	✓				✓	
		SR+DR [94]	✓				✓	
	Pseudo-labeling	Label Propagation [149]	✓					✓
		PTA [21]	✓					✓
Learned GDA	Graph Structure Learning	GAug [140]	✓			✓		
		GLCN [47]	✓			✓		
		LDS [28]	✓			✓		
		ProGNN [50]	✓			✓		
		Eland [141]	✓			✓		
	Graph Adversarial Training	RobustTraining [125]	✓			✓		
		AdvT [18]	✓		✓	✓		
		FLAG [63]	✓	✓	✓		✓	
		GraphVAT [25]	✓				✓	
	Graph Rationalization	GREA [71]		✓		✓	✓	
		AdvCA [97]		✓		✓	✓	
	Automated Augmentation	AutoGDA [144]	✓			✓	✓	
		GraphAug [79]		✓		✓	✓	
		JOAO [130]		✓		✓	✓	
		MolCLE [116]		✓		✓	✓	

Both DropNode [27] and NodeDropping [129] aim to randomly remove a fraction of the nodes from the given graph, assuming that the missing nodes should not affect the semantic meanings of the remaining nodes, or the whole graph G. Feng et al. [27] focused on semi-supervised node classification, where a consistency loss is used on the predicted logits of different augmented versions of the graphs. On the other hand, You et al. [129] focused on self-supervised graph representation learning with contrastive targets.

Feature Masking. Other than the graph structure, i.e., nodes and edges, multiple works also adopted masking augmentations on the node features. For example, graph contrastive learning methods [102, 129, 130, 151] commonly utilize stochastic feature masking as an efficient way of augmenting or corrupting the graph. On top

of randomly masking feature values (i.e., random entries in $\mathbf{X}$) or feature signals (i.e., random columns in $\mathbf{X}$), Velickovic et al. [108] utilized row swapping as an effective way of corrupting the graph. Specifically, Velickovic et al. [108] randomly re-assigned the each node's feature vector to another node in the graph, which can be obtained by row-wise shuffling of $\mathbf{X}$.

More recently, Fang et al. [23] proposed DropMessage, which masks the features aggregated by message passing in GNNs. More specifically, denoting the aggregated neighbor feature of node v by the l-th layer as $\boldsymbol{h}^l_{\mathcal{N}(v)}$ (Equation equation 48), DropMessage randomly applies a binary mask on $\boldsymbol{h}^l_{\mathcal{N}(v)}$ for each node $v \in \mathcal{V}$ in every GNN layer. Similar to other dropping methods, the masks are sampled according to a Bernoulli distribution.

Subgraph Cropping. Another common data removal augmentation approach is cropping out part of the graph data. Such subgraph cropping can usually be achieved by either sampling the remaining subgraph or the subgraph that will be cropped out. For example, You et al. [129] first proposed the Subgraph augmentation, which samples the remaining subgraph via random walk. The method later learns the graph representations by contrasting the sampled subgraphs, with the assumption that the semantics of the whole graph can be preserved in part or its local structure. On the other hand, GraphCrop [113] crops a contiguous subgraph from each of the given graph object. GraphCrop adopts a graph diffusion-based node-centric strategy, performing graph diffusion on the randomly selected seed nodes, to maintain the topology characteristics of original graphs after the cropping.

4.2 Data Addition

Opposite to data removal methods, data addition methods augments the graph data by adding components to the existing/observed graph data or directly generating additional graphs. Note that although edge dropping is one of the most common techniques in data removal, rule-based edge addition is rather uncommon due to the huge search space for potential edge addition candidates (with a complexity of $O(N^2)$). While graph diffusion methods include adding edges, we discuss them later in Section 4.3 as they also include sparsification operations after edge addition.

Virtual Node. For graph classification, creating a virtual node that connect to all nodes in the graph is a commonly used GDA approach [31, 69, 46, 44, 127]. The idea of virtual node is to compute a graph representation in parallel with the node representations during the aggregation process. Therefore, instead of using an additional pooling layer, the virtual node's representation can directly be used as the graph representation, in a way similar to the `[CLS]` token in language modeling. Moreover, as the virtual node connects to all the nodes, it allows feature aggregation between previously unreachable nodes without adding additional GNN layers. Ying et al. [127] further show that it acts similar as self-attention in Transformers. Other than graph-level tasks, Hwang et al. [45] also studied virtual nodes for link prediction. As the graph data for link prediction is usually much larger for link prediction when compared with those in graph-level tasks, Hwang et al. [45] proposed to augment the graph data multiple virtual nodes, each connecting with a subset of all nodes in the graph with assignment decided by clustering methods.

Data Interpolation. With it's simplicity and effectiveness, Mixup [135] has been commonly used in image and language domains for augmenting new data samples. Specifically, Mixup constructs virtual training examples by interpolating two labeled training samples:

$$\begin{aligned} \tilde{\boldsymbol{x}} &= \lambda \boldsymbol{x}_i + (1-\lambda)\boldsymbol{x}_j, \\ \tilde{\boldsymbol{y}} &= \lambda \boldsymbol{y}_i + (1-\lambda)\boldsymbol{y}_j, \end{aligned} \tag{51}$$

where $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ and $(\boldsymbol{x}_j, \boldsymbol{y}_j)$ are two randomly selected labeled training examples, and $\lambda \in [0, 1]$. By linearly interpolating the feature vectors and labels, Mixup incorporates the prior knowledge and extends the training distribution. Similarly, Manifold Mixup [109] performs Mixup on latent intermediate representations instead of raw features of the two training samples.

The direct analog of Mixup on graphs is not obvious, given the inter-dependent and irregular nature of graph data. Verma et al. [110] proposed GraphMix that augmented the training of a GNNs with a Fully-Connected Network, which is trained by interpolating the hidden states and labels. As GraphMix is more of a regularization method than the analog of Mixup on graphs, Wang et al. [117] proposed Graph Mixup, which analogized Manifold Mixup with a two-branch graph convolution module. Given a pair of nodes, Graph Mixup mixes their raw features, passes them into the two-branch GNN layer, and mixes the hidden representations of each layer. Notably, mixing up the nodes on features and hidden states avoids re-assembling the local neighborhoods of the two nodes. Graph Mixup also works for the task of graph classification. To avoid the node matching problem when mixing up two independent graphs, Graph Mixup mixes the latent representations of the pair of graphs.

On the other hand, ifMixup [37] directly applies Mixup on the graph data instead of the latent space for graph-level tasks. As the pair of graphs are irregular and the nodes from two graphs are not generally aligned, ifMixup arbitrarily assigns indices to the nodes in each graph and matches the nodes according to the indices. Empirically, ifMixup shows marginal performance improvements over Graph Mixup on the task of graph classification. Following ifMixup, Graph Transplant [86] also mixes graph in data space , but uses substructures as mixing units to preserve the local structural information. Graph Transplant employs the node saliency information to select one meaningful substructure from each graph, where the saliency information is defined as the l_2 norm of the gradient of the classification loss.

Different from the above Mixup-based methods which operate on instance level, Han et al. [39] proposed G-Mixup that performs Mixup on class-level. Instead of directly interpolating the individual graphs, G-Mixup interpolates the graph generators (graphons) for each class. Specifically, G-Mixup first estimates a graphon for each class of the training graphs, then mixes up the graphons of different classes, and finally generate synthetic graphs with the mixed graphons. Denoting the graphons of classes a and b as W_a and W_b, respectively, G-Mixup can be formulated as

$$\begin{aligned} \tilde{\boldsymbol{x}} \sim W_c, \text{ where } \quad W_c = \lambda W_a + (1-\lambda) W_b, \\ \tilde{\boldsymbol{y}} = \lambda \boldsymbol{y}_a + (1-\lambda)\boldsymbol{y}_b, \end{aligned} \tag{52}$$

where $\boldsymbol{y}_a$ and $\boldsymbol{y}_b$ are corresponding labels for graphs in classes a and b, respectively.

Besides Mixup, SMOTE [7] is also a classical data augmentation method that interpolates data instances. Different from Mixup which interpolates examples from different classes, SMOTE interpolates examples within the minority classes. Hence, SMOTE is especially effective when dealing with imbalanced data. On graph data, GraphSMOTE [142] augments the minority class by over-sampling synthetic nodes and then generating edges for them. GATSMOTE [76] and GNN-CL [70] further utilize attention designs to improve the edge generating process between the synthetic nodes and original nodes in the graph.

Counterfactual Augmentations. Counterfactual augmentation has been relatively under-explored in the field of graph machine learning. Zhao et al. [143] first proposed a counterfactual data augmentation method CFLP for the task of link prediction. To better understand the relationship between observed graph structure and link formation, CFLP asks the counterfactual question of "would the link still exist if the graph structure became different from observation?" To answer the question, Zhao et al. [143] proposed counterfactual links that approximates the unobserved outcome in the question. CFLP then trains a link prediction model with both the given training data and the generated counterfactual links (as augmented data). Similarly, CLBR Zhu et al. [148] proposed counterfactual data augmentation for bundle recommendation. CLBR generates the counterfactual example by answering the counterfactual question "what would a user interact with if the bundle-item affiliation relations change?".

Attribute Augmentation. Besides updating the graph topology, several works were also proposed to augment the graph data by generating additional node attributes. For example, LA-GNN [75] enhances the locality of node representations by generating node features based on the conditional distribution of the local structures and neighbor features. LA-GNN learns the new features of each node by the conditional distribution of its local neighborhood. The generated feature is directly used together with the raw node features as part of the input of

GNNs for both training and inference. Similarly, SR+DR [94] generates topology features with DeepWalk [87], and uses a dual GNN model with topology regularization to jointly train with both raw and topology features.

Pseudo-labeling. The training data in graph tasks is often only partially labeled due to the generally high cost of human labeling. With the large amount of unlabeled data, pseudo-labeling for the unlabeled data is often adopted under semi-supervised learning settings. Label Propagation [150, 149, 21] is one of the most classical methods for generating pseudo labels when only part of the nodes in the graph are labeled. Label propagation assumes that the two nodes are more likely to have the same label if they are connected, so it iteratively propagates node labels along the edges. With the propagated labels on the previously unlabeled nodes, the GNN model can then be trained with more labeled data.

4.3 Data Manipulation

Other than only adding or removing graph data, several rule-based methods also augment the graph data by combining both kind of operations. In order to separate them from the methods that purely conducts data removal or data addition, we introduce such augmentation methods in this subsection.

Diffusion. Klicpera et al. [60] first proposed generalized graph diffusion that modeled a "future" state of the graph where the signals were more spread out. Specifically, the generalized graph diffusion is formulated as

$$\tilde{\mathbf{A}} = \sum_{k=0}^{\infty} \theta_k \mathbf{T}^k, \tag{53}$$

where θ_k denote the global-local coefficient and $\mathbf{T} \in \mathbb{R}^{N \times N}$ represents the transition matrix derived from the adjacency matrix $\mathbf{A}$ (e.g., $\mathbf{A}\mathbf{D}^{-1}$ or $\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$). θ_k is usually pre-defined by specific diffusion variants, e.g., heat kernel [61] ($\theta_k = e^{-t}\frac{t^k}{k!}$) or Personalized PageRank (PPR) [84] ($\theta_k = \alpha(1-\alpha)^k$), where α denotes the teleport probability in a random walk and t is diffusion time. The analytical solution of the heat kernel and PPR diffusions are defined as

$$\tilde{\mathbf{A}}^{\text{heat}} = e^{-(t\mathbf{T}-t)}; \quad \tilde{\mathbf{A}}^{\text{PPR}} = \alpha(\mathbf{I}_N - (1-\alpha)\mathbf{T})^{-1}, \tag{54}$$

where $\mathbf{I}_N$ is the N by N identity matrix. As the obtained adjacency matrix after diffusion $\tilde{\mathbf{A}}$ is often too dense as input for GNNs, graph sparsification is commonly conducted to filter out some trivial edges, e.g., setting a threshold to cut-off edges with small weights.

For (semi-)supervised learning on graphs, $\tilde{\mathbf{A}}$ can be directly used for both training and inferencing with GNNs [60]. While most message passing-based GNNs are only capable of aggregating one-hop information in each layer, the augmented graph after diffusion allows GNNs to learn from multi-hop (global) information without specifically re-designing the GNN models. In self-supervised graph representation learning, $\tilde{\mathbf{A}}$ is often used as the augmented view for self-supervised learning objectives such as contrastive learning [40, 132].

Subgraph Substituting. Several methods also make use of special substructures such as motifs and functional groups during subgraph augmentation. For example, M-Evolve [147] utilizes motifs to augment the graph data. M-Evolve first finds and selects the target motif in the graph, then adds or removes edges within the selected motifs based on a sampling weight calculated with Resource Allocation index. Similarly, MoCL [99] utilizes biomedical domain knowledge to augment the molecular graphs on the substructures such as functional groups. MoCL selects a substructure from each molecular graph and replaces it with another substructure.

4.4 Applications of Rule-based GDA

The rule-based augmentation techniques are mostly designed for improving general graph learning, and usually does not have constrains on specific tasks or domains. For example, although Rong et al. [88] only evaluated DropEdge for node classification task, the usage of it on other tasks is straightforward, and similar for most

stochastic data removal methods discussed in Section 4.1. Nonetheless, some rule-based GDA methods are more suitable for certain domains. For instance, subgraph substituting methods [147, 99] utilizes substructure information or even biomedical domain knowledge to augment the graphs, which makes them naturally more suitable for graph-level tasks on biomedical data. On the other hand, graph diffusion methods [60] are designed based on the spread of information along the relations in the graph, which makes such methods for suitable for larger graphs such as social networks or citation networks. Similarly, designed for exploring the formation of the links, counterfactual augmentation methods [143, 148] are tailored for link prediction or recommendation on larger graphs. We also specify the targeted tasks for each GDA method in Table 1.

Other than supervised graph representation learning schemes, the stochastic data removal methods (Section 4.1) are also commonly used in self-supervised graph representation learning methods as an efficient way of augmenting/corrupting graph data. For example, several methods [108, 129, 130, 102] use one or multiple of the above-mentioned techniques as augmentation methods for generating the augmented views of graph data. We further elaborate on the usage of data removing augmentations for self-supervised learning in Section 6.

5 Learned Approaches for GDA

In the previous section, we introduced rule-based GDA approaches where no learnable parameters are involved during data augmentation. However, these approaches could sometimes be suboptimal since the augmentations do not take advantage of the rich information from downstream tasks, especially in (semi-)supervised training. Indeed, some prior works from the vision [15] and natural language [83] learning domains show the promise of learned augmentation approaches. To address this concern, learned GDA approaches are proposed to learn augmentation strategies in a data-driven manner. The existing methods can be categorized into the following types: (1) structure learning, (2) adversarial training, (3) rationalization, and (4) automated augmentation.

5.1 Graph Structure Learning

In real-world scenarios, given graph structures are often incomplete [28], noisy [50, 78] or manipulated by adversarial attacks [49, 36]. Simply applying rule-based GDA approaches for training (semi-)supervised models on such graphs can lead to suboptimal performances, as they may not necessarily generate better graph structures for downstream tasks. To tackle these issues, several works propose graph structure learning approaches which aim to search for a better graph structure that augments the initial graph structure. Essentially, those methods treat the graph structure as learnable parameters and iteratively refine it while learning the model parameters [140, 50, 28, 12, 78, 145]. Numerous studies have demonstrated the effectiveness of graph structure learning methods in improving model generalization [140, 12] and robustness [50, 145]. In the following, we introduce several representative works that fall into the category of graph structure learning.

Improving Generalization. There are numerous methods for graph structure learning that target improving the generalization performance. Overall, they can be divided into two categories based on the adjacency matrix which they learn: learning continuous structure and learning discrete structure.

Although the original adjacency matrix is usually discrete (or binary), continuous structure methods do not assume the learned adjacency matrix to be discrete, as modeling discrete structure requires additional efforts in optimization. Typically, these methods either model the adjacency matrix as free parameters or use a parameterized neural network to model the structure. For instance, GLCN [47] is an early work which proposes a unified network architecture to learn an optimal graph structure and GNN. It incorporates the similarities of node features to learn a sparse and continuous graph structure. Formally, it defines a graph learning loss $\mathcal{L}_{\text{GL}}$ as:

$$\mathcal{L}_{\text{GL}} = \sum_{i,j=1}^{N} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2 \, \tilde{\mathbf{A}}_{ij} + \gamma \|\tilde{\mathbf{A}}\|_F^2 + \beta \|\tilde{\mathbf{A}} - \mathbf{A}\|_F, \tag{55}$$

where the first two terms control the smoothness and sparsity of the augmented graph, respectively; the third term forces the augmented graph to be close to the original graph; γ and β are the hyper-parameters that balance the three terms. By minimizing $\mathcal{L}_{\text{GL}}$ together with the classification loss, GLCN is able to learn a graph structure that best serves the downstream task. TO-GCN [126] also considers the feature similarity, but it further employs label similarity to refine the graph topology. To handle the inductive learning setting, IDGL [12] casts the graph structure learning problem as similarity metric learning which will be jointly trained with the prediction model dedicated to a downstream task. To encourage learning graph structure invariant to task-irrelevant information, Sun et al. [100] utilized the Information Bottleneck [103] principle to solve the graph structure learning problem. Moreover, SLAPS [24] identifies a supervision starvation problem in previous structure learning approaches and proposes to incorporate additional self-supervision by designing a feature denoising task.

Despite the appeal of the first type of methods, continuous structures typically deviate from the original, sparse and discrete structure evident in many real-world graphs. To address this concern, some works focus on sampling the graph structures from a targeted distribution. For instance, by taking advantage of neural edge predictors like GAE [58], Zhao et al. [140] proposed GAug to generate plausible edge augmentations for an input graph. The output of the edge predictor can be formulated as

$$\mathbf{M} = \sigma_0\left(\mathbf{Z}\mathbf{Z}^T\right), \text{ with } \mathbf{Z} = \mathbf{H}^l, \tag{56}$$

where $\mathbf{M}$ is the edge probabilities matrix and σ_0 is an element-wise sigmoid function. Based on the edge probabilities matrix, two variants GAug-M and GAug-O are proposed to tackle augmentation in settings where edge manipulation is and is not feasible at inference time, respectively. Specifically, GAug-M deterministically adds edges with the highest edge probabilities to the graph at inference time; GAug-O optimizes the graph structure by minimizing the downstream classification loss together with the edge prediction loss and samples the adjacency matrix according to an element-wise Bernoulli distribution. Another representative work is LDS [28], which aims at learning discrete structure between data points while learning GNN parameters. It models the process as learning the edge probability matrix, which parameterizes the element-wise Bernoulli distribution from which the discrete structure is sampled. Then it formulates the learning process as a bi-level problem and updates the structure and model parameters in a differentiable way. Shang et al. [91] improves the efficiency of LDS by converting the bi-level problem to a uni-level problem and extends it to multivariate time series. In addition to Bernoulli distribution, recent studies have investigated other distributions to sample the discrete structure. For example, to account for the underlying generation of graphs, GEN [111] hypothesizes that the estimated graph is drawn from Stochastic Block Model (SBM) [42]. Similarly, BGCN [138] iteratively trains an assortative mixed membership stochastic block model with predictions of GCN to produce multiple denoised graphs, and ensembles results from multiple GCNs. To explicitly guarantee the strength and diversity of graph augmentation, MH-Aug [85] draws augmented graphs from an explicit target distribution through the Metropolis-Hastings algorithm, which can also be viewed as a graph structure learning process.

Instead of drawing discrete structures from targeted distributions, another line of works focus on dropping/adding edges from the original graph which can also lead to a discrete adjacency matrix. For instance, to improve the performance of GNNs under random noise, PTDNet [78] proposes to prune task-irrelevant edges by penalizing the number of edges in the sparsified graph and imposing the low-rank constraint with parameterized networks. Similarly, NeuralSparse [145] learns to drop task-irrelevant edges; it takes node/edge features as parts of input and jointly optimizes graph sparsification from the supervision of downstream task. Moreover, Gao et al. [29] proposed TADropEdge which leverages the graph spectrum to generate edge weights that represent the edges' criticality for the graph connectivity and drops edges by treating their weights as probabilities. Besides node classification tasks, Spinelli et al. [95] proposed FairDrop for the task of fair graph representation learning, which biasedly dropped edges with a sensitive attribute homophily mask to protect against unfairness. Later, Chen et al. [8] proposed AdaEdge, which iteratively adds/removes edges according to the node classification prediction. In each iteration, after the GNN model is sufficiently trained, AdaEdge adds edges between nodes that are predicted to be in the same class with high confidence, and vice versa. AdaEdge iteratively performs

GNN training and graph modification until convergence. Besides, Zhao et al. [141] proposed Eland for the task of anomaly detection on time-stamped user-item bipartite graphs. Eland first transforms the user-item graph into users' action sequences and adopts seq2seq model for future action prediction. The predicted user actions are added back into the graph to yield the augmented graph data. As the augmented graph contains richer user behavior information, Eland enhances the anomaly detection performance and detects anomalies at an early stage. It is worth mentioning that the aforementioned techniques are focused on one specific task such as node classification. To make graph structure learning benefit various downstream tasks, Liu et al. [77] proposed an unsupervised approach to learn graph structures with the aid of self-supervised contrastive learning [153].

While existing methods majorly focus on training-time augmentation, i.e., modifying the training graph data, a new line of work (e.g., GTrans [54]) introduces test-time augmentation by transforming the test graph through optimizing a self-supervised loss. It has been demonstrated to significantly improve the generalization performance of GNNs on out-of-distribution data.

Improving Robustness. Recent studies have demonstrated the vulnerability of GNNs under adversarial attacks, i.e., carefully-crafted small perturbation on the input graph leads GNNs into giving wrong predictions [157, 17, 154, 51]. A series of works are proposed to focus on enhancing the robustness of graph neural networks under adversarial attacks by learning clean graph structure. Jin et al. [50] observed that adversarial attacks violate important graph properties such as sparsity, low-rank, and feature smoothness; it then proposes the ProGNN framework to robustify GNNs by alternatively updating the graph structure by preserving these graph properties by adding penalizing regularization terms and training GNN parameters on the updated graph structure. Specifically, it defines the following graph learning loss:

$$\mathcal{L}_{\text{GL}} = \alpha\|\tilde{\mathbf{A}}\|_1 + \beta\|\tilde{\mathbf{A}}\|_* + \lambda \operatorname{tr}\left(\mathbf{X}^T\tilde{\mathbf{L}}\mathbf{X}\right) + \|\tilde{\mathbf{A}} - \mathbf{A}\|_F^2, \tag{57}$$

where $\|\cdot\|_1$ is the ℓ_1 norm, $\|\cdot\|_*$ is the nuclear norm, and $\tilde{\mathbf{L}}$ is the normalized Laplacian matrix of $\tilde{\mathbf{A}}$. The first three terms in Equation equation 57 force the learned graph to preserve the properties of sparsity, low-rank, and feature smoothness, respectively. Similar to GLCN [47], ProGNN also includes the downstream classification loss in the graph learning process. Despite the robustness of ProGNN, it is computationally expensive with $\mathrm{O}(N^3)$ time complexity and $\mathrm{O}(N^2)$ space complexity. To speed up ProGNN, LRGNN [124] decouples the adjacency matrix into a low-rank component and a sparse one, and learns the graph structure by minimizing the rank of the low-rank component and suppressing the sparse one. Furthermore, as robust GNNs tend to yield unsatisfying performance when trained with limited labeled nodes, Dai et al. [16] took advantage of self-supervision and uses node attributes to predict the links so as to boost robust performance, which also saves computational cost from direct structure learning. Also using a link predictor, DefenseVAE [134] employs variational graph autoencoder [58] to reconstruct graph structure that can reduce the effects of adversarial perturbations and boost the performance of GNNs under adversarial attacks. In addition, utilizing information theory, CoGSL [74] targets at learning the most compact structure relevant to downstream tasks in order to achieve a better balance between robustness and accuracy. Instead of explicitly learning the graph structure, GNNGuard [137] mitigates the negative effects of adversarial attacks by assigning higher weights to edges connecting similar nodes while pruning edges between dissimilar nodes, which can also be considered as implicit graph structure learning. While the aforementioned techniques have shown robustness in some specific settings, one recent work [82] revealed that their robustness decreases significantly under proper evaluation (in particular the adaptive attacks). This suggests that a more powerful and adaptive GSL method is needed for effective defense.

It is worth noting that there are some other graph structure learning works which aim at learning graphs to improve the scalability of graph machine learning models [53, 52, 73]. They do not target improving model performance or robustness of GNNs, and hence are not in the GDA scope tackled in this work.

5.2 Graph Adversarial Training

Adversarial training is a widely used countermeasure for adversarial attacks on computer vision [32], and has also been extended to graph domain [17, 25, 20, 43, 18, 10, 63]. Unlike graph structure learning, graph adversarial training does not seek to find an optimal graph structure. Instead, it augments input graphs with adversarial patterns during model training by perturbing node features or graph structure. The adversarially trained models are expected to tolerate adversarial perturbations in graph data and yield better generalization and robustness performance at test time. At the core of adversarial training is the injection of adversarial examples into the training set, with which the trained model can predict the test adversarial examples properly. Thus, we can adopt this strategy to enhance the robustness of GNNs as follows,

$$\min_{\Theta} \max_{\substack{\Delta_{\mathbf{A}} \in \mathcal{P}_{\mathbf{A}} \\ \Delta_{\mathbf{X}} \in \mathcal{P}_{\mathbf{X}}}} \mathcal{L}_{\text{train}} \left(g_{\Theta}(\mathbf{A} + \Delta_{\mathbf{A}}, \mathbf{X} + \Delta_{\mathbf{X}}) \right), \tag{58}$$

where $\mathcal{L}_{\text{train}}$ denotes the training loss for the downstream task; $\Delta_{\mathbf{A}}$ and $\Delta_{\mathbf{X}}$ stand for the perturbation on $\mathbf{A}, \mathbf{X}$, respectively; $\mathcal{P}_{\mathbf{A}}$ and $\mathcal{P}_{\mathbf{X}}$ denote the perturbation space. From the bi-level optimization problem in Equation equation 58, we can observe that adversarial training generates perturbations that maximize the prediction loss and updates model parameters to minimize the prediction loss. The process of generating perturbations (i.e., $\mathbf{A} + \Delta_{\mathbf{A}}, \mathbf{X} + \Delta_{\mathbf{X}}$) can be viewed as adversarial data augmentation and we can leverage such augmentations to improve the model robustness and generalization.

To augment the adjacency matrix, Dai et al. [17] proposed to randomly drop edges during adversarial training without any optimization on the graph data. While this strategy does not bring significant improvement, such cheap adversarial training still shows some improvement in robust classification accuracy. This finding is also in line with that from Zügner and Günnemann [156]. Instead of randomly dropping edges, Xu et al. [125] leveraged projected gradient descent (PGD) to optimize the bi-level problem and generate perturbations on the discrete structure, which achieves significant improvement in robust performance. Similarly, Chen et al. [9] and Dai et al. [18] also used existing adversarial attacks to modify the input graph structure during adversarial training, designed for network embedding methods. Furthermore, Suresh et al. [101] proposed to generate adversarial graph augmentation by learning to drop edges such that the augmentation can capture the minimal information that is sufficient to classify each graph.

On the other hand, there are some works focusing on perturbing the input features to serve as adversarial examples. For instance, Feng et al. [25] proposed an adversarial training strategy with dynamic regularization, which aims to reconstruct graph smoothness and constrains the divergence between the prediction of the target node and its connected nodes. Deng et al. [20] proposed batch virtual adversarial training to promote the smoothness of GNNs and thus defend against adversarial perturbations. Moreover, Kong et al. [63] proposed FLAG which utilizes adversarial training to iteratively augment the node features with gradient-based adversarial perturbations and improves the performances of GNNs on node classification, link prediction, and graph classification tasks. In addition, Zügner and Günnemann [155] studied certifiable robustness of GNNs w.r.t. perturbations of node attributes and propose a robust training scheme inspired by the certificates. Several other variants of adversarial training on perturbing node features are introduced in [112, 43].

5.3 Rationalization

A rationale is defined as a subset of input features that best represent, guide and support model prediction [71]. In the graph domain, rationales are subgraphs intrinsically learned by graph learning models. Rationales can be viewed as a form of augmented graph data that provide intrinsic explanations to the graph models' predictions, as opposed to the post-hoc explanation methods. Rationalization is commonly applied to graph-level property prediction or classification tasks [119, 71, 129, 13, 81, 68] for drug and material discovery on molecular and polymer datasets, etc.

Rationalization emerged in the graph domain as an approach to enhance both the interpretability and overall performance of graph classification and regression. Yu et al. [131] found similarity to the Information Bottleneck (IB) problem, and proposed the Graph Information bottleneck framework (GIB), which learns to generate the maximally informative and compressed subgraph (IB-graph) by leveraging a bi-level optimization scheme and a novel connectivity loss. Also rooted in the IB paradigm, Miao et al. [81] proposed GSAT to better learn and select task-relevant subgraphs that improve interpretation and prediction by injecting stochasticity into the attention weights in order to constrain information from task-irrelevant components. GREA, another rationalization work proposed by Liu et al. [71], proposed a novel environment replacement augmentation method, which separates the rationale and the environment subgraphs (the remaining and complementary subgraphs to the rationale ones) and optimized the separation (rationalization identification) with data augmentation by replacing the original environment subgraph with a different one in the latent space.

Rationalization models are also effective in addressing data bias and out-of-distribution (OOD) problems for graph property prediction tasks since rationales are both interpretable and generalizable [81]. Wu et al. [120] proposed DIR to generate distribution perturbation on training data with causal intervention. Based on the idea that causal patterns are stable to distribution shift, they created a rationale generator that separates causal and non-causal graphs, applies causal intervention to create perturbed distributions, and then jointly learn both the causal and non-causal representation to minimize invariant risk. Similarly, Chen et al. [13] also took a causal perspective to solve the OOD problem. They proposed CIGA to model the graph generation process and the interactions between invariant and spurious features with Structural Causal Models (SCM). The resulting subgraphs generated by CIGA maximally preserves the invariant intra-class information. Li et al. [68] also proposed to separate invariant and variant graphs. In their framework GIL, they proposed a GNN based subgraph generator to identify potentially invariant subgraphs, then infer latent environment labels for the variant subgraphs, before jointly optimizing all modules. To address the limited environments and unstable causal features in data augmentation methods for graph rationalization, AdvCA [97] was proposed to improve the generalization capacity against covariate shift through adversarial causal augmentation.

5.4 Automated Augmentation

GDA techniques mentioned in Section 4 take rule-based approaches to augment graph data, applying the same augmentation method to subgraphs and graphs which embody different attributes and characteristics like degree distribution and homophily. To tackle this issue, Automated GDA techniques [98, 79, 144, 130, 64, 41, 153] were recently explored to automatically learn tailored augmentations for different subgraphs or graphs. For example, Sun et al. [98] proposed AutoGRL for the task of node classification. Through the training process, AutoGRL learns the best combination of GDA operations, GNN architecture, and hyper-parameters. The searching space of AutoGRL includes four GDA operations implemented by random masking and GAug-M [140]: drop features, drop nodes, add edges, and remove edges.

Since automated GDA objectives are often complex to optimize, some recent works use reinforcement learning approaches as a solution. Zhao et al. [144] framed the AutoGDA as a bi-level optimization problem, aiming to find a different set of augmentation strategies for each community in the graph as they observed various characteristics in each community. They employ an RL-agent to generalize the learning and find localize augmentation strategies for node classification tasks. On graph classification tasks, Luo et al. [79] set out to learn an automated augmentation model with GraphAug, to provide label-invariant augmentations for each graph in the dataset. Applying reinforcement learning, they maximize the estimated label-invariance probability to learn the augmentation category and transformation selection.

Another group of works on automated augmentation focus on graph contrastive learning. You et al. [130] proposed to learn augmentations to replace ad hoc and handpicked augmentations for contrastive learning. They design an augmentation-aware projection head to avoid complicated augmentations, and formulate a bi-level optimization problem to learn both the augmentation strategy and graph representation. Hassani and Khasahmadi

Table 2: Representative self-supervised graph learning works that utilized graph data augmentation techniques. †Although the methods in this category are semi-supervised methods, they used GDA operations with only self-supervised learning objectives (i.e, consistency loss). Therefore, we categorize their GDA techniques as designed for self-supervised learning objectives.

	Representative Works	Task Level			Augmented Data		
		Node	Graph	Edge	Structure	Feature	Label
Contrastive Learning	DGI [108]	✓				✓	
	GRACE [151]	✓			✓	✓	
	MVGRL [40]	✓			✓		
	GraphCL [129]		✓		✓	✓	
	JOAO [130]		✓		✓	✓	
Non-contrastive Learning	CCA-SSG [136]	✓			✓	✓	
	GBT [3]	✓			✓	✓	
	BGRL [102]	✓			✓	✓	
	T-BGRL [92]	✓			✓	✓	
Consistency Training†	GRAND [27]	✓			✓	✓	
	NodeAug [114]	✓			✓	✓	
	MV-GCN [132]	✓			✓		
	NASA [4]	✓			✓		

[41] learned a probabilistic policy that contains a set of distributions over different augmentation operations in their method LG2AR, and samples an augmentation strategy from the policy in each training epoch. Zhu et al. [153] proposed GCA, which proposes adaptive augmentations based on node centrality measures. Unlike the aforementioned methods which find the best augmentation strategy for the dataset, GCA adaptively augments different nodes according to their importance. Wang et al. [116] proposed to use a generative probabilistic model and a learnable feature selector to automatically parameterize topological and attribute augmentations, which can also provide explanations for underlying patterns in molecular graphs. Lastly, Kose and Shen [64] proposed FairAug which utilizes adaptive augmentations for fair graph representation learning.

6 GDA for Self-supervised Learning

Other than directly using the augmented graph data in supervised learning, the most common use case for GDA is under self-supervised learning (SSL) schemes, e.g. contrastive learning. Self-supervised objectives learn representations that are robust to noise and perturbations by maximizing the (dis)agreements of learned representations. Therefore, unlike most of the previously mentioned learned GDA techniques (Section 5) which aim to enhance the task-relevant information in the data, most of the GDA techniques for self-supervised learning are rule-based augmentations (Section 4) which aim to corrupt or perturb the given graph data. Moreover, most self-supervised graph representation learning methods tend to use a combination of several simple GDA operations. In this section, we introduce three commonly used self-supervised graph learning schemes as well as the GDA approaches they utilize.

6.1 Contrastive Learning

In recent years, with the rapid development of contrastive learning in CV [11], many contrastive learning methods [151, 129, 105, 122, 71, 56] have been proposed for applications on graph data. Typically, a graph contrastive learning framework includes three main components: a GDA module that generates different views of the given graph data, a GNN-based encoder to compute the representations, and a contrastive learning objective to train the model. For each data example (nodes for node-level tasks and graphs for graph-level tasks), these methods consider augmented views or variants of itself as associated positive samples and other data examples in

the same batch as associated negative samples. Contrastive learning objectives then maximize the (dis)agreements of the representations between each data example with their (negative) positive examples.

To efficiently generate different augmented data for graph contrastive learning, rule-based data removal operations (Section 4.1) are the most commonly used GDA techniques, as they are fast and easy to apply. For example, multiple methods (GRACE [151], GraphCL [129], etc.) adopt stochastic edge dropping and/or feature masking due to their simplicity. DGI [108] adopts feature corruption by conducting a row-wise shuffling on the raw node feature matrix $\mathbf{X}$. In general, graph contrastive learning methods usually adopt a combination of multiple augmentation techniques to generate different augmented views. GraphCL [129] and InfoGCL [123] adopt four GDA operations: node dropping which randomly removes nodes along with its edges, edge perturbation which randomly adds or drops edges, attribute masking which randomly masks off certain node attributes, and subgraph sampling which samples connected subgraphs. SUBG-CON [48] utilizes a subgraph sampler to extract the context subgraph as a proxy of data augmentation. GRACE [151] uses only the basic random edge dropping and attribute masking for creating different views of the graph.

Other than data removal augmentations, graph diffusion is also commonly used in contrastive learning as it can naturally create a "future view" of the given graph where the information are more spread out. MVGRL [40] adopts the diffusion graph proposed by GDC [60] as the second view. Interestingly, Hassani and Khasahmadi [40] showed that using three views (original graph, PPR diffusion graph and heat kernel diffusion graph) would not result with better performance than using two views (original graph and one diffusion graph), and concluded "increasing the number of views does not improve the performance." However, Yuan et al. [132] later proposed MV-CGC which adopted a similar contrastive learning framework with three views: the original graph, diffusion graph, and a proposed feature similarity view. Empirically, the node representations learned by MV-CGC outperformed those learned by MVGRL on node classification, suggesting that additional well-designed GDA methods or views may be helpful to graph contrastive learning approaches.

More recently, several studies [101, 106, 72] pointed out that stochastic rule-based GDA operations may suffer from failing to induce useful task-relevant invariance on common benchmark datasets. Specifically, Trivedi et al. [106] analyzed that the generalization error of graph contrastive learning can be bounded under the assumptions of invariance to relevant augmentations, recoverability, and separability, which refer to *data-centric properties*, by instantiating rule-based GDA as a composition of graph edit operations. Such bound demonstrates conditions with low separability and recoverability during the usage of rule-based GDA, which motivates the necessity of inducing task-relevant invariance. Following the theoretical analysis, Zhang et al. [139] proposed a covariance-preserving feature augmentation technique, in which the augmented feature has bounded variance. Wang et al. [115] proposed to use different levels in hierarchical graphs as augmented views.

6.2 Non-contrastive Learning

While showing promising performance on various tasks, contrastive learning methods rely heavily on disagreement between data examples and their associated negative examples to avoid model collapse [33]. As sampling high quality negative examples is often costly, and random negative sampling usually requires large batch sizes, several works [33, 133, 1] propose non-contrastive self-supervised learning methods to learn representations in a self-supervised manner without needing negative examples. Instead of comparing across different samples, non-contrastive self-supervised methods compare only between different views of the same sample and use designs such as prediction heads and stop gradient to avoid model collapsing [33], or measure the cross-correlation matrix between the representations learned form different views [133].

As the non-contrastive methods are designed for more efficient self-supervised learning than the contrastive methods, the GDA techniques they adopt are all the most basic, stochastic ones (Section 4.1). Specifically, all the non-contrastive self-supervised graph representation learning methods (CCA-SSG [136], GBT [3], BGRL [102], and T-BGRL [92]) utilized only random edge dropping and node feature masking as the augmentation strategies. While the first three methods generates two augmented views for comparison, to further improve the performance

on link prediction under inductive settings, T-BGRL [92] also used the same augmentation strategies but with higher masking probability as an efficient corruption to create an third "negative" view to mitigate collapse, which is later used in a triplet loss.

6.3 Consistency Training

In real GML applications, semi-supervised learning usually plays an important role as only a small fraction of training data are labeled in most of the cases [121]. Due to such label scarcity, consistency training is commonly used to leverage the unlabeled data to improve the model quality. Similar to contrastive learning, consistency training itself is a self-supervised learning objective that aims to maximizes the agreement of representations learned from different views of the data. However, unlike (non-)contrastive learning that compares between data objects, the consistency loss compares the distributions of a batch of representations via metrics like KL-divergence. Therefore, the consistency loss is rarely used itself, but often used along with supervised losses in the semi-supervised learning settings. The final learning objective is usually a linear combination of the supervised loss (e.g., cross entropy for classification tasks) and the consistency loss.

NodeAug [114] uses three local structure-based augmentation operations: replacing attributes, removing and adding edges. NodeAug minimizes the KL-divergence between the node representations learned from the original graph and augmented graph. GRAND [27] creates multiple different augmented graphs with node dropping and feature masking. The consistency loss then minimizes the distances of the representations learned from the augmented graphs. NASA [4] proposes Neighbor Replace augmentation to randomly replace the 1-hop neighbors with 2-hop neighbors, and then use a neighbor-constrained consistency regularization during training. To further utilize the information given by different graph diffusions, MV-GCN [132] generates two complementary views with PPR and heat kernel and learns from both created views and the original graph. Then, it feeds three views of the graph into three GCNs, and uses a consistency regularization loss to reduce the distribution distance of the representations learned across the views, and derives the final node representations as a combination of the three.

7 Challenges and Directions

Despite substantial progress has been achieved in graph data augmentation research, several open problems remain to solve. In this section, we summarize several promising yet under-explored research directions.

7.1 Domain Adaptation and Regularization

Given the rapid development of GDA techniques in recent years, automated GDA methods have been proposed to automatically tune the augmentation strategy for different datasets and tasks. Nonetheless, the existing automated GDA methods for graph data (as introduced in Section 5.4) mainly focus on specific datasets and downstream tasks. Ideally, automated augmentation solutions should be transferable. That is, domain adaptation is a desired characteristic for automated GDA techniques. When the automated augmentation method trained on one dataset could only be used on that dataset, the method may be equivalent to automating the hyperparameter tuning process and lose the generalizability across datasets [144]. Therefore, for an ideal automated GDA method, it should be able to be trained on one dataset and used for many, ideally cross domain or under OOD settings. While OOD benchmarks are already available in the GML community [35], automated GDA methods that can be transferable across domains are still missing in the literature. Moreover, on certain types of graph data such as molecule graphs, most commonly used GDA operations would change the underlying semantics of the graph. For example, dropping a carbon atom from the phenyl ring of aspirin breaks the aromatic system and results in a alkene chain [66], which is an entirely different chemical compound. This motivates a need for domain-based regularization methods for such tasks. So far, only Sun et al. [99] proposed MoCL that

considers the semantic information brought by local substructures when augmenting the molecule graphs, leaving domain-based regularization GDA methods rather under-explored.

7.2 Scalability for Large-Scale Graphs

GDA techniques add additional complexity on top of the existing GNNs, and many GDA techniques use global information during the augmentation process, which might not be able to easily scale. For example, GAug-M [140] involves selecting the top K out of $O(N^2)$ logits for node pairs when selecting edges to add. Such high complexity operations can cause scalability issues in actual applications where the graph size can be very large, e.g., at billion scale. While complex GDA techniques bring significant performance improvements, the scalability of these methods are still worthy of attention. For example, in order to enable end-to-end training, GAug-O [140] requires back-propagating on the entire learned adjacency matrix, creating massive memory overheads. To improve the performance of DropEdge [88], TADropEdge [29] required the pre-calculation of a score for each edge in the graph prior to the training of GNNs. Therefore, to be applicable in practical applications, efficiency is also a necessity for GDA techniques. As mentioned in the previous subsections, automated solution which combine the fast and simple augmentation operations may be a promising direction. Nonetheless, how to design a scalable and efficient automated GDA framework is still an open line of research.

7.3 Comprehensive Evaluation Criteria and Standards

Similar to the DA research in other domains, a general concern for GDA research is that the evaluation only focuses on the prediction performance on specific datasets. Although this is likely the most important metric, other metrics such as additional time and resource consumption, transferability, or scalability are also important for researchers to more comprehensively understand the methods. For example, as aforementioned, while graph structure learning methods such as GAug [140] shows promising performances for node classification, the method's design inherently limits its ability to generalize on large-scale graphs. Furthermore, only few works discuss the additional time and resource requirement needed for applying their proposed GDA methods, especially for the learned augmentations which may require training of additional modules. Therefore, a set of comprehensive evaluation criteria and standards is desired for better understanding the benefits and costs of the newly proposed GDA methods. Ideally, such a benchmark could contain multiple datasets in different scales and domains, enabling researchers to better evaluate transferability and scalability tradeoffs.

7.4 Theoretical Foundation

GDA is a powerful technology to improve the performance of data-driven inference on graphs without the need of extra labeling effort or complex models. Empirically, GDA methods are also shown to improve the generalization of GML methods and alleviate the over-smoothing problem encountered by GNNs. Yet, there is little rigorous understanding of how and why GDA achieves those benefits, especially for (semi-)supervised learning. Although several works [140, 8] have analyzed the relation between graph homophily and classification performance or the over-smoothing problem, there is limited work showcasing rigorous proofs or theoretical bounds on these relationships.

Recently, several works provided theoretical insights of DA in the CV domain. For example, Wu et al. [119] theoretically analyzed the generalization effect of data augmentation on images. They interpreted the effect of data augmentation from a bias-variance perspective, where data augmentation adds new information to model training while also serving as a regularizer. Due to the irregular characteristics of graph data, these theoretical analysis cannot be directly applied for the GDA context. Besides the generalization perspective, several recent works have studied the certified robustness of GNNs [156]. Improved robustness bounds would be a desired property of GDA techniques. Recent studies [104] on the topology bottleneck and over-squashing of GNNs

provide theoretical guides for edge-based GDA techniques. Counterfactual augmentation methods on graphs such as CFLP [143] can also bring insights for analyzing GDA from the perspective of causality.

7.5 Data Augmentation for Complex Graph Types

Existing GDA approaches are mainly designed for homogeneous graphs, while not all of them can be easily generalized to other complex types of graphs such as heterogeneous graphs, dynamic graphs, hypergraphs, etc. These complex graphs have broader applications with their ability of modeling more complex relationship, nonetheless, the complexity of the data requires more sophisticated design of GDA methods. Taking heterogeneous graphs as an example, even the simplest edge dropping would require a drop rate hyper-parameter for each of the edge types in the graph, which could introduce significant computational overhead for hyper-parameter searching. Additionally, beyond direct analogous of GDA methods for homogeneous graph for complex graph types, specially designed GDA methods for different graph types could better utilize the rich information contained in them. Therefore, a comprehensive evaluation of the existing GDA methods on complex graphs is needed by the community to better understand the effectiveness of existing GDA methods and also better design principled augmentation approaches for each graph types.

8 Conclusion

Our work presents a comprehensive and structured survey of data augmentation techniques for graph machine learning (GML). We categorized existing graph data augmentation (GDA) techniques three taxonomies from different perspectives, introduced recent GDA approaches based on their core methodology, and introduced their applications in self-supervised learning. Finally, we outlined current challenges as well as directions for future research explorations in the GDA domain. We hope this survey serves as a guide for GML researchers and practitioners to study and use GDA techniques, and inspires additional interest and work on this topic.

References

[1] R. Balestriero and Y. LeCun. Contrastive and non-contrastive self-supervised learning recover global and local spectral embedding methods. arXiv:2205.11508, 2022.

[2] R. Barandela, R. M. Valdovinos, J. S. Sánchez, and F. J. Ferri. The imbalanced training sample problem: Under or over sampling? In Joint IAPR international workshops on SPR and SSPR, 2004.

[3] P. Bielak, T. Kajdanowicz, and N. V. Chawla. Graph barlow twins: A self-supervised representation learning framework for graphs. Knowledge-Based Systems, 2022.

[4] D. Bo, B. Hu, X. Wang, Z. Zhang, C. Shi, and J. Zhou. Regularizing graph neural networks via consistency-diversity graph augmentations. In AAAI, 2022.

[5] I. Brugere, B. Gallagher, and T. Y. Berger-Wolf. Network structure inference, a survey: Motivations, methods, and applications. CSUR, 2018.

[6] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. arXiv:1312.6203, 2013.

[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. JAIR, 2002.

[8] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In AAAI, 2020.

[9] J. Chen, Y. Wu, X. Lin, and Q. Xuan. Can adversarial network attack be defended? arXiv:1903.05994, 2019.

[10] J. Chen, X. Lin, H. Xiong, Y. Wu, H. Zheng, and Q. Xuan. Smoothing adversarial training for gnn. IEEE Transactions on Computational Social Systems, 2020.

[11] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In ICML, 2020.

[12] Y. Chen, L. Wu, and M. Zaki. Iterative deep graph learning for graph neural networks: Better and robust node embeddings. In NeurIPS, 2020.
[13] Y. Chen, Y. Zhang, Y. Bian, H. Yang, M. KAILI, B. Xie, T. Liu, B. Han, and J. Cheng. Learning causally invariant representations for out-of-distribution generalization on graphs. In NeurIPS, 2022.
[14] F. Chierichetti, A. Epasto, R. Kumar, S. Lattanzi, and V. Mirrokni. Efficient algorithms for public-private social networks. In KDD, 2015.
[15] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation strategies from data. In CVPR, 2019.
[16] E. Dai, W. Jin, H. Liu, and S. Wang. Towards robust graph neural networks for noisy graphs with sparse labels. In WSDM, 2022.
[17] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song. Adversarial attack on graph structured data. In ICML, 2018.
[18] Q. Dai, X. Shen, L. Zhang, Q. Li, and D. Wang. Adversarial training methods for network embedding. In TheWebConf, 2019.
[19] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In NeurIPS, 2016.
[20] Z. Deng, Y. Dong, and J. Zhu. Batch virtual adversarial training for graph convolutional networks. arXiv:1902.09192, 2019.
[21] H. Dong, J. Chen, F. Feng, X. He, S. Bi, Z. Ding, and P. Cui. On the equivalence of decoupled graph convolution network and label propagation. In TheWebConf, 2021.
[22] Q. Duong, M. P. Wellman, and S. Singh. Modeling information diffusion in networks with unobserved links. In IEEE PASSAT/SocialCom, 2011.
[23] T. Fang, Z. Xiao, C. Wang, J. Xu, X. Yang, and Y. Yang. Dropmessage: Unifying random dropping for graph neural networks. arXiv:2204.10037, 2022.
[24] B. Fatemi, L. El Asri, and S. M. Kazemi. Slaps: Self-supervision improves structure learning for graph neural networks. NeurIPS, 2021.
[25] F. Feng, X. He, J. Tang, and T.-S. Chua. Graph adversarial training: Dynamically regularizing based on graph structure. TKDE, 2019.
[26] S. Y. Feng, V. Gangal, J. Wei, S. Chandar, S. Vosoughi, T. Mitamura, and E. Hovy. A survey of data augmentation approaches for nlp. arXiv:2105.03075, 2021.
[27] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang. Graph random neural networks for semi-supervised learning on graphs. NeurIPS, 2020.
[28] L. Franceschi, M. Niepert, M. Pontil, and X. He. Learning discrete structures for graph neural networks. In ICML, 2019.
[29] Z. Gao, S. Bhattacharya, L. Zhang, R. S. Blum, A. Ribeiro, and B. M. Sadler. Training robust graph neural networks with topology adaptive edge dropping. arXiv:2106.02892, 2021.
[30] K. A. Garrison, D. Scheinost, E. S. Finn, X. Shen, and R. T. Constable. The (in) stability of functional brain network measures across thresholds. Neuroimage, 2015.
[31] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. Neural message passing for quantum chemistry. In ICML, 2017.
[32] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. arXiv:1412.6572, 2014.
[33] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. NeurIPS, 2020.
[34] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In KDD, 2016.
[35] S. Gui, X. Li, L. Wang, and S. Ji. Good: A graph out-of-distribution benchmark. NeurIPS, 2022.
[36] S. Günnemann. Graph neural networks: Adversarial robustness. In Graph Neural Networks: Foundations, Frontiers, and Applications. 2022.
[37] H. Guo and Y. Mao. Intrusion-free graph mixup. arXiv:2110.09344, 2021.
[38] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In NeurIPS, 2017.
[39] X. Han, Z. Jiang, N. Liu, and X. Hu. G-mixup: Graph data augmentation for graph classification. In ICML, 2022.
[40] K. Hassani and A. H. Khasahmadi. Contrastive multi-view representation learning on graphs. In ICML, 2020.

[41] K. Hassani and A. H. Khasahmadi. Learning graph augmentations to learn graph representations. arXiv:2201.09830, 2022.
[42] P. W. Holland, K. B. Laskey, and S. Leinhardt. Stochastic blockmodels: First steps. Social networks, 1983.
[43] W. Hu, C. Chen, Y. Chang, Z. Zheng, and Y. Du. Robust graph convolutional networks with directional graph adversarial training. Applied Intelligence, 2021.
[44] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. NeurIPS, 2021.
[45] E. Hwang, V. Thost, S. S. Dasgupta, and T. Ma. Revisiting virtual nodes in graph neural networks for link prediction. 2021.
[46] K. Ishiguro, S.-i. Maeda, and M. Koyama. Graph warp module: an auxiliary module for boosting the power of graph neural networks in molecular graph analysis. arXiv preprint arXiv:1902.01020, 2019.
[47] B. Jiang, Z. Zhang, D. Lin, J. Tang, and B. Luo. Semi-supervised learning with graph learning-convolutional networks. In CVPR, 2019.
[48] Y. Jiao, Y. Xiong, J. Zhang, Y. Zhang, T. Zhang, and Y. Zhu. Sub-graph contrast for scalable self-supervised graph representation learning. In ICDM, 2020.
[49] W. Jin, Y. Li, H. Xu, Y. Wang, S. Ji, C. Aggarwal, and J. Tang. Adversarial attacks and defenses on graphs: A review, a tool and empirical studies. arXiv:2003.00653, 2020.
[50] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang. Graph structure learning for robust graph neural networks. In KDD, 2020.
[51] W. Jin, Y. Li, H. Xu, Y. Wang, S. Ji, C. Aggarwal, and J. Tang. Adversarial attacks and defenses on graphs. SIGKDD Explorations, 2021.
[52] W. Jin, X. Tang, H. Jiang, Z. Li, D. Zhang, J. Tang, and B. Yin. Condensing graphs via one-step gradient matching. In KDD, 2022.
[53] W. Jin, L. Zhao, S. Zhang, Y. Liu, J. Tang, and N. Shah. Graph condensation for graph neural networks. In ICLR, 2022.
[54] W. Jin, T. Zhao, J. Ding, Y. Liu, J. Tang, and N. Shah. Empowering graph representation learning with test-time graph transformation. ICLR, 2023.
[55] T. Joachims and A. Swaminathan. Counterfactual evaluation and learning for search, recommendation and ad placement. In SIGIR, 2016.
[56] M. Ju, T. Zhao, Q. Wen, W. Yu, N. Shah, Y. Ye, and C. Zhang. Multi-task self-supervised graph neural networks enable stronger task generalization. ICLR, 2023.
[57] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. arXiv:1609.02907, 2016.
[58] T. N. Kipf and M. Welling. Variational graph auto-encoders. arXiv:1611.07308, 2016.
[59] J. Klicpera, A. Bojchevski, and S. Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. arXiv:1810.05997, 2018.
[60] J. Klicpera, S. Weißenberger, and S. Günnemann. Diffusion improves graph learning. NeurIPS, 2019.
[61] R. I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete structures. In ICML, 2002.
[62] K. Kong, G. Li, M. Ding, Z. Wu, C. Zhu, B. Ghanem, G. Taylor, and T. Goldstein. Flag: Adversarial data augmentation for graph neural networks. arXiv:2010.09891, 2020.
[63] K. Kong, G. Li, M. Ding, Z. Wu, C. Zhu, B. Ghanem, G. Taylor, and T. Goldstein. Robust optimization as data augmentation for large-scale graphs. In CVPR, 2022.
[64] O. D. Kose and Y. Shen. Fair node representation learning via adaptive data augmentation. arXiv:2201.08549, 2022.
[65] S. Kumar and N. Shah. False information on web and social media: A survey. arXiv:1804.08559, 2018.
[66] N. Lee, J. Lee, and C. Park. Augmentation-free self-supervised learning on graphs. In AAAI, 2021.
[67] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. IEEE Transactions on Signal Processing, 2018.
[68] H. Li, Z. Zhang, X. Wang, and W. Zhu. Learning invariant graph representations for out-of-distribution generalization. In NeurIPS, 2022.
[69] J. Li, D. Cai, and X. He. Learning graph-level representation for drug discovery. arXiv preprint arXiv:1709.03741, 2017.

[70] X. Li, L. Wen, Y. Deng, F. Feng, X. Hu, L. Wang, and Z. Fan. Graph neural network with curriculum learning for imbalanced node classification. arXiv:2202.02529, 2022.

[71] G. Liu, T. Zhao, J. Xu, T. Luo, and M. Jiang. Graph rationalization with environment-based augmentations. In KDD, 2022.

[72] G. Liu, E. Inae, T. Zhao, J. Xu, T. Luo, and M. Jiang. Data-centric learning from unlabeled graphs with diffusion model. arXiv preprint arXiv:2303.10108, 2023.

[73] M. Liu, S. Li, X. Chen, and L. Song. Graph condensation via receptive field distribution matching. arXiv:2206.13697, 2022.

[74] N. Liu, X. Wang, L. Wu, Y. Chen, X. Guo, and C. Shi. Compact graph structure learning via mutual information compression. In TheWebConf, 2022.

[75] S. Liu, H. Dong, L. Li, T. Xu, Y. Rong, P. Zhao, J. Huang, and D. Wu. Local augmentation for graph neural networks. arXiv:2109.03856, 2021.

[76] Y. Liu, Z. Zhang, Y. Liu, and Y. Zhu. Gatsmote: Improving imbalanced node classification on graphs via attention and homophily. Mathematics, 2022.

[77] Y. Liu, Y. Zheng, D. Zhang, H. Chen, H. Peng, and S. Pan. Towards unsupervised deep graph structure learning. In TheWebConf, 2022.

[78] D. Luo, W. Cheng, W. Yu, B. Zong, J. Ni, H. Chen, and X. Zhang. Learning to drop: Robust graph neural network via topological denoising. In WSDM, 2021.

[79] Y. Luo, M. McThrow, W. Y. Au, T. Komikado, K. Uchino, K. Maruhash, and S. Ji. Automated data augmentations for graph classification. arXiv:2202.13248, 2022.

[80] Y. Ma, X. Liu, T. Zhao, Y. Liu, J. Tang, and N. Shah. A unified view on graph neural networks as graph signal denoising. In CIKM, 2021.

[81] S. Miao, M. Liu, and P. Li. Interpretable and generalizable graph learning via stochastic attention mechanism. In ICML, 2022.

[82] F. Mujkanovic, S. Geisler, S. Günnemann, and A. Bojchevski. Are defenses for graph neural networks robust? NeurIPS, 2022.

[83] T. Niu and M. Bansal. Automatically learning data augmentation policies for dialogue tasks. arXiv:1909.12868, 2019.

[84] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[85] H. Park, S. Lee, S. Kim, J. Park, J. Jeong, K.-M. Kim, J.-W. Ha, and H. J. Kim. Metropolis-hastings data augmentation for graph neural networks. NeurIPS, 2021.

[86] J. Park, H. Shim, and E. Yang. Graph transplant: Node saliency-guided graph mixup with local structure preservation. arXiv:2111.05639, 2021.

[87] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In KDD, 2014.

[88] Y. Rong, W. Huang, T. Xu, and J. Huang. Dropedge: Towards deep graph convolutional networks on node classification. arXiv:1907.10903, 2019.

[89] R. Sennrich, B. Haddow, and A. Birch. Improving neural machine translation models with monolingual data. arXiv:1511.06709, 2015.

[90] N. Shah, A. Beutel, B. Gallagher, and C. Faloutsos. Spotting suspicious link behavior with fbox: An adversarial perspective. In ICDM, 2014.

[91] C. Shang, J. Chen, and J. Bi. Discrete graph structure learning for forecasting multiple time series. In ICLR, 2021.

[92] W. Shiao, Z. Guo, T. Zhao, E. E. Papalexakis, Y. Liu, and N. Shah. Link prediction with non-contrastive learning. arXiv:2211.14394, 2022.

[93] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. Journal of Big Data, 2019.

[94] R. Song, F. Giunchiglia, K. Zhao, and H. Xu. Topological regularization for graph neural networks augmentation. arXiv:2104.02478, 2021.

[95] I. Spinelli, S. Scardapane, A. Hussain, and A. Uncini. Fairdrop: Biased edge dropout for enhancing fairness in graph representation learning. TAI, 2021.

[96] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. JMLR, 2014.

[97] Y. Sui, X. Wang, J. Wu, A. Zhang, and X. He. Adversarial causal augmentation for graph covariate shift. arXiv preprint arXiv:2211.02843, 2022.

[98] J. Sun, B. Wang, and B. Wu. Automated graph representation learning for node classification. In IJCNN, 2021.

[99] M. Sun, J. Xing, H. Wang, B. Chen, and J. Zhou. Mocl: data-driven molecular fingerprint via knowledge-aware contrastive learning from molecular graph. In KDD, 2021.

[100] Q. Sun, J. Li, H. Peng, J. Wu, X. Fu, C. Ji, and S. Y. Philip. Graph structure learning with variational information bottleneck. In AAAI, 2022.

[101] S. Suresh, P. Li, C. Hao, and J. Neville. Adversarial graph augmentation to improve graph contrastive learning. NeurIPS, 2021.

[102] S. Thakoor, C. Tallec, M. G. Azar, M. Azabou, E. L. Dyer, R. Munos, P. Veličković, and M. Valko. Large-scale representation learning on graphs via bootstrapping. In ICLR, 2022.

[103] N. Tishby, F. C. Pereira, and W. Bialek. The information bottleneck method. arXiv, 2000.

[104] J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In ICLR, 2022.

[105] P. Trivedi, E. S. Lubana, Y. Yan, Y. Yang, and D. Koutra. Augmentations in graph contrastive learning: Current methodological flaws & towards better practices. arXiv:2111.03220, 2021.

[106] P. Trivedi, E. S. Lubana, M. Heimann, D. Koutra, and J. J. Thiagarajan. Analyzing data-centric properties for graph contrastive learning. In NeurIPS, 2022.

[107] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. arXiv:1710.10903, 2017.

[108] P. Velickovic, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm. Deep graph infomax. ICLR, 2019.

[109] V. Verma, A. Lamb, C. Beckham, A. Najafi, I. Mitliagkas, D. Lopez-Paz, and Y. Bengio. Manifold mixup: Better representations by interpolating hidden states. In ICML, 2019.

[110] V. Verma, M. Qu, A. Lamb, Y. Bengio, J. Kannala, and J. Tang. Graphmix: Regularized training of graph neural networks for semi-supervised learning. arXiv:1909.11715, 2019.

[111] R. Wang, S. Mou, X. Wang, W. Xiao, Q. Ju, C. Shi, and X. Xie. Graph structure estimation neural networks. In TheWebConf, 2021.

[112] X. Wang, X. Liu, and C.-J. Hsieh. Graphdefense: Towards robust graph convolutional networks, 2019.

[113] Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi. Graphcrop: Subgraph cropping for graph classification. arXiv:2009.10564, 2020.

[114] Y. Wang, W. Wang, Y. Liang, Y. Cai, J. Liu, and B. Hooi. Nodeaug: Semi-supervised node classification with data augmentation. In KDD, 2020.

[115] Y. Wang, Y. Min, X. Chen, and J. Wu. Multi-view graph contrastive representation learning for drug-drug interaction prediction. In TheWebConf, 2021.

[116] Y. Wang, Y. Min, E. Shao, and J. Wu. Molecular graph contrastive learning with parameterized explainable augmentations. In BIBM, 2021.

[117] Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi. Mixup for node and graph classification. In TheWebConf, 2021.

[118] J. Wei and K. Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. arXiv:1901.11196, 2019.

[119] S. Wu, H. Zhang, G. Valiant, and C. Ré. On the generalization effects of linear transformations in data augmentation. In ICML, 2020.

[120] Y. Wu, X. Wang, A. Zhang, X. He, and T.-S. Chua. Discovering invariant rationales for graph neural networks. In ICLR, 2021.

[121] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. TNNLS, 2020.

[122] Y. Xie, Z. Xu, J. Zhang, Z. Wang, and S. Ji. Self-supervised learning of graph neural networks: A unified review. TPAMI, 2022.

[123] D. Xu, W. Cheng, D. Luo, H. Chen, and X. Zhang. Infogcl: Information-aware graph contrastive learning. NeurIPS, 2021.

[124] H. Xu, L. Xiang, J. Yu, A. Cao, and X. Wang. Speedup robust graph structure learning with low-rank information. In CIKM, 2021.

[125] K. Xu, H. Chen, S. Liu, P.-Y. Chen, T.-W. Weng, M. Hong, and X. Lin. Topology attack and defense for graph neural networks: An optimization perspective. arXiv:1906.04214, 2019.

[126] L. Yang, Z. Kang, X. Cao, D. Jin, B. Yang, and Y. Guo. Topology optimization based graph convolutional network. In IJCAI, 2019.

[127] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu. Do transformers really perform badly for graph representation? NeurIPS, 2021.

[128] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In KDD, 2018.

[129] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen. Graph contrastive learning with augmentations. NeurIPS, 2020.

[130] Y. You, T. Chen, Y. Shen, and Z. Wang. Graph contrastive learning automated. In ICML, 2021.

[131] J. Yu, T. Xu, Y. Rong, Y. Bian, J. Huang, and R. He. Graph information bottleneck for subgraph recognition. arXiv:2010.05563, 2020.

[132] J. Yuan, H. Yu, M. Cao, M. Xu, J. Xie, and C. Wang. Semi-supervised and self-supervised classification with multi-view graph neural networks. In CIKM, 2021.

[133] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny. Barlow twins: Self-supervised learning via redundancy reduction. In ICML, 2021.

[134] A. Zhang and J. Ma. Defensevgae: Defending against adversarial attacks on graph data via a variational graph autoencoder. arXiv:2006.08900, 2020.

[135] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. arXiv:1710.09412, 2017.

[136] H. Zhang, Q. Wu, J. Yan, D. Wipf, and P. S. Yu. From canonical correlation analysis to self-supervised graph neural networks. NeurIPS, 2021.

[137] X. Zhang and M. Zitnik. Gnnguard: Defending graph neural networks against adversarial attacks. NeurIPS, 2020.

[138] Y. Zhang, S. Pal, M. Coates, and D. Ustebay. Bayesian graph convolutional neural networks for semi-supervised classification. In AAAI, 2019.

[139] Y. Zhang, H. Zhu, Z. Song, P. Koniusz, and I. King. Costa: Covariance-preserving feature augmentation for graph contrastive learning. In KDD, 2022.

[140] T. Zhao, Y. Liu, L. Neves, O. Woodford, M. Jiang, and N. Shah. Data augmentation for graph neural networks. In AAAI, 2021.

[141] T. Zhao, B. Ni, W. Yu, Z. Guo, N. Shah, and M. Jiang. Action sequence augmentation for early graph-based anomaly detection. In CIKM, 2021.

[142] T. Zhao, X. Zhang, and S. Wang. Graphsmote: Imbalanced node classification on graphs with graph neural networks. In WSDM, 2021.

[143] T. Zhao, G. Liu, D. Wang, W. Yu, and M. Jiang. Learning from counterfactual links for link prediction. In ICML, 2022.

[144] T. Zhao, X. Tang, D. Zhang, H. Jiang, N. Rao, Y. Song, P. Agrawal, K. Subbian, B. Yin, and M. Jiang. Autogda: Automated graph data augmentation for node classification. In LoG, 2022.

[145] C. Zheng, B. Zong, W. Cheng, D. Song, J. Ni, W. Yu, H. Chen, and W. Wang. Robust graph representation learning via neural sparsification. In ICML, 2020.

[146] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang. Random erasing data augmentation. In AAAI, 2020.

[147] J. Zhou, J. Shen, and Q. Xuan. Data augmentation for graph classification. In CIKM, 2020.

[148] S. Zhu, Q. Shen, Y. Zhang, Y. Pang, and Z. Wei. Data-augmented counterfactual learning for bundle recommendation. arXiv:2210.10555, 2022.

[149] X. Zhu. Semi-supervised learning with graphs. Carnegie Mellon University, 2005.

[150] X. Zhu and Z. Ghahramani. Learning from labeled and unlabeled data with label propagation. 2002.

[151] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang. Deep graph contrastive representation learning. arXiv:2006.04131, 2020.

[152] Y. Zhu, W. Xu, J. Zhang, Y. Du, J. Zhang, Q. Liu, C. Yang, and S. Wu. A survey on graph structure learning: Progress and opportunities. arXiv, 2021.

[153] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang. Graph contrastive learning with adaptive augmentation. In

TheWebConf, 2021.

[154] D. Zügner and S. Günnemann. Adversarial attacks on graph neural networks via meta learning. arXiv preprint arXiv:1902.08412, 2019.

[155] D. Zügner and S. Günnemann. Certifiable robustness and robust training for graph convolutional networks. In KDD, 2019.

[156] D. Zügner and S. Günnemann. Certifiable robustness of graph convolutional networks under structure perturbations. In KDD, 2020.

[157] D. Zügner, A. Akbarnejad, and S. Günnemann. Adversarial attacks on neural networks for graph data. In KDD, 2018.

It's FREE to join!

TCDE

tab.computer.org/tcde/

The Technical Committee on Data Engineering (TCDE) of the IEEE Computer Society is concerned with the role of data in the design, development, management and utilization of information systems.

- Data Management Systems and Modern Hardware/Software Platforms
- Data Models, Data Integration, Semantics and Data Quality
- Spatial, Temporal, Graph, Scientific, Statistical and Multimedia Databases
- Data Mining, Data Warehousing, and OLAP
- Big Data, Streams and Clouds
- Information Management, Distribution, Mobility, and the WWW
- Data Security, Privacy and Trust
- Performance, Experiments, and Analysis of Data Systems

The TCDE sponsors the International Conference on Data Engineering (ICDE). It publishes a quarterly newsletter, the Data Engineering Bulletin. If you are a member of the IEEE Computer Society, you may join the TCDE and receive copies of the Data Engineering Bulletin without cost. There are approximately 1000 members of the TCDE.

Join TCDE via Online or Fax

ONLINE: Follow the instructions on this page:

www.computer.org/portal/web/tandc/joinatc

FAX: Complete your details and fax this form to **+61-7-3365 3248**

Name _____

IEEE Member # _____

Mailing Address _____

_____

Country _____

Email _____

Phone _____

TCDE Mailing List

TCDE will occasionally email announcements, and other opportunities available for members. This mailing list will be used only for this purpose.

Membership Questions?

Xiaoyong Du
Key Laboratory of Data Engineering and Knowledge Engineering
Renmin University of China
Beijing 100872, China
duyong@ruc.edu.cn

TCDE Chair

Xiaofang Zhou
School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, QLD 4072, Australia
zxf@uq.edu.au

IEEE Computer Society
10662 Los Vaqueros Circle
Los Alamitos, CA 90720-1314

Non-profit Org.
U.S. Postage
PAID
Los Alamitos, CA
Permit 1398