Deep Generative Models for Subgraph Prediction

by

Erfaneh Mahmoudzadeh

B.Sc., Shahid Beheshti University, 2020

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Summer 2024

Declaration of Committee

Name:	**Erfaneh Mahmoudzadeh**
Degree:	**Master of Science**
Thesis title:	**Deep Generative Models for Subgraph Prediction**
Committee:	**Chair:** Mo Chen Assistant Professor, Computing Science
	Oliver Schulte Supervisor Professor, Computing Science
	Martin Ester Committee Member Professor, Computing Science
	Maxwell Libbrecht Examiner Associate Professor, Computing Science

Abstract

Graph Neural Networks (GNNs) are crucial across various domains due to their ability to model complex relational data. This work introduces subgraph queries as a new task in deep graph learning. Unlike traditional tasks like link prediction or node classification, subgraph queries predict components of a target subgraph based on evidence from an observed subgraph. For instance, they can predict a set of target links and node labels. In this work, I have introduced VGAE+ to answer these queries, using a probabilistic deep Graph Generative Model (GGM): an inductively trained Variational Graph Auto-Encoder (VGAE), enhanced to represent a joint distribution over links, node features, and labels. Bayesian optimization tunes the weighting of links, node features, and labels. I developed deterministic and sampling-based inference methods for estimating subgraph probabilities from the VGAE distribution. Evaluation on six benchmark datasets shows VGAE+ surpasses baselines, with AUC improvements up to 0.2 points.

Keywords: Subgraph Query; Inductive Link Prediction; Inductive Node Classification; Variational Graph Auto-Encoder; Probabilistic Graph Query; Bayesian optimization

Dedication

To my parents, who are my roots no matter how far I am from them.
To Amineh, Aghileh, and Ali, who I love the most and are always there for me.
To Saba, Hosna, and Suren, who remind me there is always hope for a brighter future.
To all the young people of my country, whose chance to experience a day like this was unjustly taken from them.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr. Oliver Schulte, for his unwavering support, insightful guidance, and constant encouragement throughout my master's journey. I am truly thankful for your mentorship.

A special thanks to Parmis Naddaf and Kiarash Zahirnia. Your endless support has made all the difference in my research. Your friendship and guidance have been invaluable, and I am deeply grateful.

I also want to extend my appreciation to Dr. Max Libbrecht for fulfilling the role of my examiner, Dr. Martin Ester for serving as a member of my committee, and Dr. Mo Chen for chairing my thesis defense. Their contributions and support are greatly appreciated.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and Overview

In this chapter, I review terms and definition on graphs, graph query tasks, and Graph Generative Models.

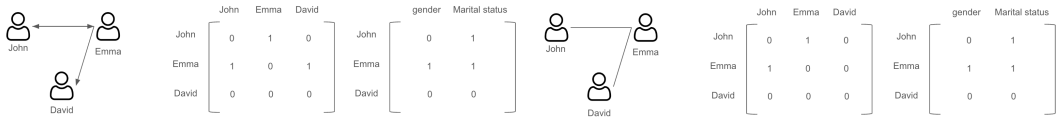
1.1 Background and Definitions

In this section, I provide an overview of the foundational concepts and terminologies that are essential for understanding the subsequent discussions on graphs and graph neural networks in this thesis.

1.1.1 Graphs

Attributed graphs are mathematical structures used to represent relationships between pairs of objects. In graph theory, a graph G is defined as an ordered pair (V, E), where V is a set of nodes and E is a set of edges or links. Each node u in a graph can have a set of features. Each edge (u, v) in a graph connects node u to node v. In a directed graph, the order of nodes u, v is important, while in an undirected graph it does not matter. An undirected link between nodes refers to symmetric relationships like the connections on the LinkedIn website, while directed links refer to the relations that are asymmetric, for example, the connection on the Twitter website is directed, you may follow some while they do not follow you. Figures 1.1a and 1.1b show examples of a directed and an undirected graph, along with their adjacency matrix and feature matrix.
A graph G, with N nodes and k features, can be represented with one $N \times N$ adjacency matrix, $\boldsymbol{A}$, where $\boldsymbol{A}[u, v]$ shows the connection between node u and v, and one $N \times k$ feature matrix, $\boldsymbol{X}$, where $\boldsymbol{X}[u]$ shows the features of node u.

Homogeneous Graph A homogeneous graph is a graph with one type of nodes and one type of edges. In these graphs all nodes or edges have the same type. For example, in a citation network, all nodes are papers. Also all edges are identical, which means we have one type of relation between entities. For example, in the citation network, each edge(u, v)



(a) A directed graph with 3 nodes with 2 features and 3 links. Since links are directed in this graph, we have an asymmetric adjacency matrix

(b) A directed graph with 3 nodes with 2 features and 2 links. Since links are undirected in this graph, we have a symmetric adjacency matrix

Figure 1.1: Example of a directed and undirected graph and corresponding adjacency and feature matrices.

means that paper u has cited paper v. Homogeneous graphs are often used to model networks where the entities and their relationships are uniform.

Heterogeneous Graph A heterogeneous graph is a graph with different types of nodes and edges. This means that, in these graphs, each node can represent different entities with different features, and each edge can represent different types of relations between entities. For example, in a movie database network, each node can represent a movie, a director, or an actor, and each edge can represent different relations like "played in" or "directed". This complexity of heterogeneous graphs allows them to better model real-world systems, where interactions are not uniform.

1.1.2 Inductive, Semi-inductive and Transductive Training Methods

Inductive, Semi-inductive and transductive prediction methods are fundamental approaches for training graph learning models, distinguished by whether the model is trained to generalize to unseen nodes (inductive) or to make predictions only within the confines of a fixed set of nodes present during training (transductive).

Inductive and Semi-inductive Prediction Inductive and Semi-inductive queries in graph analysis involve using learned patterns and structures from a training graph to infer unseen parts of the graph or entirely new graphs. This approach allows models to generalize beyond the specific instances they were trained on. In practice, these queries can be applied to various tasks, such as node classification, where the model predicts the labels of nodes in a new graph, and link prediction, where the model predicts the existence of edges between nodes in new or partially observed graphs [8].

Transductive Prediction Transductive learning in graph queries focuses on predicting labels for a specific set of nodes within the graph, leveraging both the labeled and unlabeled data available during the training phase. Unlike inductive learning, which aims to infer a generalizable model applicable to any new data point, transductive learning targets only

those nodes present in the training set. This approach is particularly useful in scenarios where the task is to classify or label a known set of nodes in a graph, utilizing the graph's structure and relationships between nodes to improve prediction accuracy [43, 45].

1.1.3 Graph Queries

The term *graph query* refers to prediction tasks for graph data. Graph query tasks are used for extracting information from graph-structured data. Here, I review some common graph queries.

Link Prediction

In graph queries, *link prediction* involves estimating the likelihood of the existence of a link between two nodes in a graph. The goal of this task is to identify missing links in the graph and determine whether two nodes should be connected. Several methods use knowledge from the given graph to calculate this likelihood. Link prediction is a common task in network analysis and can be used in various applications such as social network analysis, recommendation systems, and bioinformatics. Here are some of the most common methods:

1. Heuristic-Based Methods: This method involves identifying nodes with higher mutual structural features. These methods include measures like Common Neighbors: where two nodes are more likely to be connected if they share many mutual neighbors, and the Jaccard Coefficient, which calculates the similarity between nodes by dividing the number of common neighbors by the total number of unique neighbors [19].

2. Matrix Factorization: This method involves using techniques like Singular Value Decomposition (SVD) or Non-negative Matrix Factorization (NMF) on the adjacency matrix. The resulting latent features enable the prediction of missing links and the identification of potential connections within the target nodes in the graph [22].

3. Graph Embedding Methods: These methods use a combination of structural and feature information to generate embeddings for each node. Based on these node embeddings, they predict the probability of a target link between them [45, 7].

Node Classification

Node classification is a task in graph queries where the goal is to predict the labels of target nodes using both the node features and the graph's structural information. Some nodes in the graph may have known labels, and the task is to use these labeled nodes to predict the labels for the unlabeled target nodes. This task is common in various applications, such as social network analysis (predicting user interests), fraud detection (classifying fraudulent transactions), and biological network analysis (identifying protein functions). The most common approach is to compute latent node feature vectors, also known as *node embeddings*,

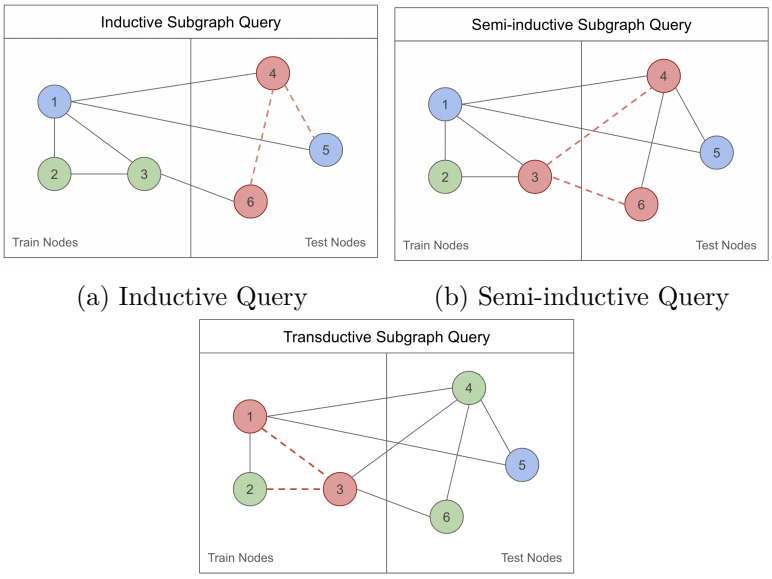
which can then be used as inputs to standard machine learning classifiers to predict node labels. Here are some methods for node classification:

1. Feature Propagation: In this method, features of nodes are iteratively spread across the graph to enhance the classification task. Initially, nodes possess their own features, and some nodes may be labeled. During propagation, each node updates its features by aggregating information from its neighbors until the feature vectors stabilize. The final feature matrix is the embedding for each node [39].

2. Graph-based Machine Learning Models: These models iteratively update node representations by incorporating information from their local neighborhoods, allowing them to capture the complex relationships and dependencies within the graph, ultimately facilitating accurate node classification. For example, Graph Convolutional Networks (GCNs) [45], Graph Attention Networks (GATs) [33], and GraphSAGE [8], work by leveraging the structure and features of a graph to do node classification.

3. Matrix Factorization: These methods involve decomposing the graph's adjacency matrix $\boldsymbol{A}$ into lower-dimensional matrices to uncover latent features that represent the nodes. These latent features capture the underlying structure and relationships within the graph, making them useful for predicting node labels [46].

Subgraph Queries



(a) Inductive Query (b) Semi-inductive Query

(c) Transductive Query

Figure 1.2: Examples of inductive, semi-inductive and transductive subgraph queries.

Subgraph Queries (SQs) specify a set of links and labels for a set of query nodes. The specification can involve a language as complex as first-order logic [3]. In this work, I present

a straightforward SQ syntax, primarily consisting of combinations of links and/or node attributes and labels. The syntax is adequate to represent the link prediction and node classification tasks previously studied in graph learning. (See Section 2.) Figure 1.2 shows examples for inductive, semi-inductive and transductive subgraph queries. In Figure 1.2a target query nodes are nodes 4 and 6 and our subgraph query is to predict the labels for those nodes and predict the existence of links (6, 4) and (5, 4). Since nodes 4 and 6 were unseen during the training, this is an inductive SQ. Figure 1.2b shows examples for inductive, semi-inductive and transductive subgraph queries. In Figure 1.2a target query nodes are nodes 4 and 6 and our subgraph query is to predict the labels for nodes 4, 6 and predict the existence of links (3,4) and (6,4). Since nodes 4 and 6 were unseen during the training and node 3 was seen during the training time, this is an inductive SQ. In Figure 1.2b target query nodes are nodes 1, 4, and 6 and our subgraph query is to predict the labels for nodes 3, 4, 6 and predict the existence of links (3,4) and (6,4). Since nodes 4 and 6 were unseen during the training and node 3 was seen during the training time, this is a semi-inductive SQ. Also Figure 1.2c shows an example for transductive SQ, where target query nodes are nodes 1 and 3 and our subgraph query is to predict the labels for those nodes and predict the existence of links (1,3) and (2,3). Since nodes 1, 2, and 3 were used during the training, this is a transductive SQ.

1.1.4 Variational Auto-Encoders

This method involves identifying nodes with higher mutual structural features [15]. These models are probabilistic models that learn to encode data to a lower-dimensional space (latent space) and decode data from the latent space to the original dimensions. During training, the VAE aims to minimize the reconstruction error between the input data and the data generated by the decoder, making them similar. During testing, it tries to generate new data points similar to the input data on which it was trained. Also, it minimizes the Kullback-Leibler Divergence (KLD) between the distribution of latent representations and a predefined prior distribution.

Variational Graph Auto-Encoders

Variational Graph Auto-Encoders (VGAEs) are extensions of Variational Auto-Encoders (VAEs) designed to work with graph-structured data [16]. VGAEs combine the strengths of GNNs with the probabilistic nature of VAEs to learn robust representations of graphs. The architecture of a VGAE typically consists of two main components: the encoder and the decoder. The encoder maps the input graph to a latent space. It uses a GNN to encode the node features and the graph structure into a latent representation. This latent representation is characterized by a mean vector μ and a variance vector σ for each node. Similar to VAEs,

VGAEs sample latent vectors $\boldsymbol{z}$ from generated distribution by the encoder.

$$q(\boldsymbol{z}[u]|\boldsymbol{E} = \boldsymbol{e}, \boldsymbol{Y} = \boldsymbol{y}) \sim N(\mu[u], \sigma[u]). \tag{1.1}$$

The decoder reconstructs the graph structure from the latent space. It predicts the probability of the existence of edges between pairs of nodes based on their latent representations.

1.2 Overview of Thesis

This thesis discusses the motivation for subgraph queries and my approach to answering them.

1.2.1 Motivation

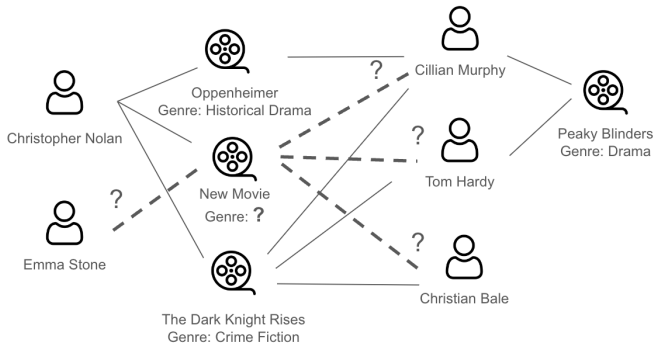


Figure 1.3: Example of joint link prediction and node classification: predicting potential actors for Christopher Nolan's new movie and its genre. Target links are dashed lines marked with "?". Solid links are specified as evidence.

Due to the importance of graph data, deep graph learning is a growing field with many applications. Typical graph learning tasks include link prediction and node classification. Most research in this area has provided models that perform either link prediction or node classification tasks. This research introduces *subgraph prediction*, a novel task in graph learning that simultaneously performs link prediction and/or node classification within a specific subgraph. In this section, I provide some examples to illustrate the power of subgraph prediction, describe use cases, and clarify its relationship to link prediction and node classification. Figure 1.3 shows a scenario with a new film directed by Christopher Nolan. Previously, he collaborated with Cillian Murphy, Christian Bale, and Tom Hardy in "The Dark Knight Rises," and with Cillian Murphy in "Oppenheimer." Suppose our evidence stipulates that Cillian Murphy is cast in this new movie, and we want to predict which previous collaborators will join him and what type of movie it will be. The subgraph prediction

task is to jointly predict: (i) the movie's genre and (ii) which additional actors will join the cast.

Predicting the genre of the new movie is an instance of single node classification, an extensively studied task [38, 2]. Predicting whether Tom Hardy will be in the new movie (independent of other actors) is an instance of single link prediction, an equally extensively studied task [9]. Predicting whether Tom Hardy, Cillian Murphy, and Christian Bale will all join the movie, is an instance of *joint link prediction*, a recently introduced task [23] that generalizes single link prediction. This example query illustrates how subgraph prediction generalizes both joint link prediction and node classification.

For another example, consider fraud detection in financial networks. A node classification approach is to label a network node as "suspicious" or "normal" [31]. However, recent attack types involve collusion among many network nodes controlled by the attackers [17]. Detecting collusion can be achieved by a more powerful approach based on joint node classification. Moreover, attacking nodes exchange messages at a substantially higher frequency than normal nodes [34]. Subgraph prediction can capture joint association patterns among node labels and links to model both which nodes are suspicious and how suspicious nodes interact with each other.

Subgraph queries are significantly more expressive than works that focus on single tasks. They allow users to choose a *set* of links and a *set* of node features or labels rather than assuming a fixed prediction target type (e.g., single link, single label). Second, users can provide different kinds of evidence, depending on what is known, and use a single system for answering the query. In contrast, previous researchers have developed a separate customized method for different evidence types [29]. Answering general relational queries, including subgraph queries, is a major use case for non-neural statistical-relational models, such as Markov Logic networks [3], and Probabilistic Soft Logic [13], which motivates my goal of expanding the prediction tasks to which neural models can be applied.

Recent generative AI models have shown the usefulness of a single multi-task system for many users compared to separate systems tailored for individual tasks. For deploying graph learning, a single query answering system is important in a production environment where it is not known in advance which graph queries will be important, and users may not have the resources to build a customized machine learning solution for different query types.

1.2.2 Approach

Figure 1.4 shows our system design. Our approach is a form of *domain-specific pre-training*: a probabilistic Graph Generative Model (GGM) is learned from data in a deployment domain. The model is used to answer queries with no further learning required. The deployment domain can be large (e.g., 1M nodes), whereas queries are typically small (e.g., involving 10-100 nodes). Specifically, I train a generative Variational Graph Auto-Encoder (VGAE) [16, 9] inductively so that the VGAE can be applied to query targets of different sizes that may

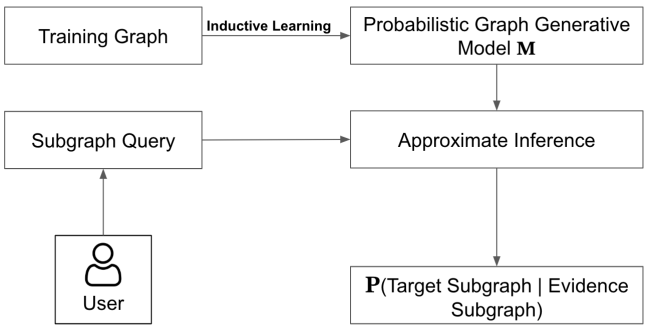


Figure 1.4: After training a single GGM, the approximate inference methods described in this thesis can answer a user query.

involve unseen nodes. The input for VGAE is a graph. To use the graph as input of the model, the links in the graph are represented by an adjacency matrix $\boldsymbol{A}$, and node features are represented by using a feature matrix $\boldsymbol{X}$. The generative probabilities over graphs (implicitly) define a conditional probability $P(target\_subgraph|evidence\_subgraph)$ for every subgraph query. The VGAE utilizes an encoder-decoder architecture with a GNN as an encoder, *augmented* to generate node labels or features as well as links. While VGAEs are not SOTA for generating realistic graphs, they are useful for generating node embeddings for downstream prediction tasks, such as link prediction [23]. I describe and evaluate approximate inference methods for estimating conditional subgraph probabilities from a trained VGAE. Ease of use and wide applicability make inference from a single model useful, even if, on some queries, inference from a model is less accurate than a system custom-built for that query.

1.2.3 Evaluation

I evaluate a variety of query types depending on whether 1) the target subgraph involves a single neighbor or multiple neighbors of a target node, and 2) the target comprises only links among testing nodes (inductive), or links that connect training and test nodes (semi-inductive). Our main baseline is to estimate link probabilities and node labels independently using state of the art node classification/link prediction methods. For subgraph queries, I find that the VGAE+ model achieves superior or competitive performance on all datasets and settings, especially inductive inference. For node classification/link prediction problems, I find that the inference from the augmented VGAE model, VGAE+, is competitive with custom methods, even though our subgraph query system supports a much wider range of queries.

1.2.4 Contributions

Our main contributions are as follows.

- Introducing the task of conditional subgraph prediction.
- Approximate inference methods, including sampling, for answering subgraph queries from a single trained GGM.
- An augmented Variational Graph Auto Encoder model for jointly modelling links and node features or labels. A new training objective for the VGAE+ that utilizes Bayesian optimization to weight link prediction and node classification.

Chapter 2

Related Works

I review the work most related to the topics addressed in this thesis.

2.1 Graph Queries

Powerful (non-probabilistic) graph query languages have been developed, such as Cypher and SPARQL [5, 26, 6]. Graph queries are typically used to retrieve data that is stored in a graph data model. Such graph queries typically return a set of nodes or links that satisfy a complex condition [6]. In contrast, subgraph queries return a probability for a target subgraph, conditional on other subgraphs. Below I often refer to probabilistic graph queries as "graph queries" for short, although the concept is different from that used in graph query languages.

2.2 Inductive Graph Training

Inductive graph learning has been a major topic in recent research [8, 42, 27]. Unlike traditional transductive learning, which operates on a fixed set of nodes and edges, inductive graph training focuses on making predictions for nodes or graphs that were not part of the training set. In this approach, models use GNNs to work with new, unseen data. This makes it a strong approach for real-world tasks where new information is always emerging. Heterogeneous Graph Transformer (HGT) [12] addresses the problem of inductive link prediction in heterogeneous networks. The encoder in the HGT uses meta relations to parameterize attention and message passing mechanisms, ensuring nodes and edges maintain type-specific representations. The decoder then aggregates these messages, using attention weights to update the target node's representation, ultimately providing contextualized embeddings for downstream tasks. GIN [40] is used for inductive graph classification task. It learns to classify entire graphs, rather than individual nodes. To support inductive link prediction or node classification training, our query answering approach can be used with any inductive encoder-decoder graph neural network architecture, trained with the variational ELBO ob-

jective, modified to generate attributes as well as links. To our knowledge, this is the first use of the variational ELBO objective to support inductive subgraph prediction.

2.3 Graph Generative Models

Graph Generative Models (GGMs) are machine learning models designed to generate new graphs that are similar to a given set of training graphs. GraphRNN [18] is an autoregressive model that generates graphs node by node and edge by edge using a Recurrent Neural Network (RNN) to capture the sequential dependencies in graph structure. GraphVAEs [30] encode entire graphs into a latent space and then decode this latent representation to generate new graphs. These models are particularly effective for generating graphs of fixed size and structure by learning a probabilistic representation of the graph's underlying distribution. Another example for GGMs are Graph Generative Adversarial Networks (GraphGANs) [35]. GraphGANs generate realistic graph structures by learning from existing graph data. They consist of two main components: (1) a generator which tries to create graph structures that mimic the properties of real graphs and (2) a discriminator which evaluates the authenticity of these generated graphs by distinguishing them from real graph samples.

Inference from a model requires the following properties of a GGM.

1. Supports the computation/estimation of explicit graph probabilities.
2. Admits a conditional variant, that supports conditioning on subgraphs.
3. Applies to graphs of different sizes.

VGAEs meet these requirements, so I base our experiments on them. Specifically, I employ the most recent VGAE designed for link prediction [23]. Another advantage of the VGAE is that it is designed for a single large dataset, as are most methods for link prediction and node classification, so I can employ the same benchmark datasets. In contrast, other deep GGMs are usually trained on datasets with many disjoint graphs [4] (e.g., molecules). I believe that developing deep GGMs so that they support answering subgraph queries is a fruitful new direction for GNNs.

2.4 Link Prediction

For deep graph models, the most common setting is *single link completion*: predict a single target link given a set of known links, which typically include a large set of training links. Other recent variants include the following single link tasks.

1. Condition on the attributes of the two target nodes only.
   (a) DEAL [10] is a model for inductive link prediction in attributed graphs, that uses dual encoders for attribute and structure embeddings along with an alignment mechanism.

2. Condition on links among training and test nodes.

   (a) SEAL is a framework designed for transductive single link prediction [32]. SEAL generates embeddings by extracting local enclosing subgraphs around target links to learn graph structure features from these subgraphs. Nodes within these subgraphs are labeled based on their distances to the target link, and these labeled subgraphs are input to the GNN, which processes them to produce link prediction scores.

   (b) IGMC generates embeddings by first extracting a local enclosing subgraph around each user-item pair and labeling the nodes based on their roles and distances [44]. This labeled subgraph is then processed by a GNN to learn and aggregate feature representations. Finally, the target user and item representations are concatenated and passed through a Multi-Layer Perceptron (MLP) to predict the rating.

3. Condition on links among other test nodes only.

   (a) The GraIL model is the inductive version of SEAL [32]. GraIL generates embeddings by extracting local subgraphs around target nodes, labeling the nodes based on their distances to the target link, and then using a GNN to iteratively aggregate and update node representations. The final subgraph representation and node embeddings are combined with the target relation embedding to score the likelihood of the given relation.

These single link tasks are generalized by the recently introduced joint link prediction task [23]. In joint link prediction, the target is a set of links to be predicted jointly, given another set of links and node features as evidence. Joint link prediction is the closest predecessor to our work in that it aims to answer a large class of graph queries from a single model. However, joint link prediction does not provide the capability of predicting node labels or features, which is important in many applications. Subgraph prediction requires training a generative model to support the dual task of joint link prediction and joint node classification, which I address with Bayesian optimization in this thesis.

2.5 Node Classification

Node classification is one of the most common tasks in graph analysis [9]. The goal is to predict a class for each unlabeled node in the graph based on available graph evidence [14], which usually includes all links and node features. In this research I have considered two types of node classification:

1. Semi-inductive node classification frameworks

(a) DeepWalk is a transductive model that generates latent representations of nodes in a graph by using local information from truncated random walks [25]. The method uses optimization techniques to build representations that capture shared similarities in local graph structure between nodes. These representations are then used as input features for classification methods, such as logistic regression, to do node classification.

2. Inductive node classification frameworks

   (a) MVGRL is a self-supervised approach for learning representations of nodes and graphs by contrasting different structural views of graphs [11]. MVGRL transforms a sample graph into two correlated structural views: an adjacency matrix for local structure and a diffusion matrix for global structure. This approach captures both local and global information. Two separate GNNs process each view. These GNNs generate node representations, which are then passed through a shared MLP projection head. The node representations are pooled to obtain graph-level representations and processed again through the MLP projection head. A discriminator scores the agreement between these representations.

These models differ in that they only perform node classification, whereas a subgraph query also requires predicting links.

2.6 Two-Task GNNs

There is a recent trend towards two-task GNNs that support both node classification and link prediction [37]. I have considered three models that can perform both node classification and link prediction tasks. However, while these models are designed to handle both node classification and link prediction, they require separate training for each task. In contrast, I train a single model for the large set of diverse tasks that can be represented as subgraph queries. To our knowledge, inference from a generative model to estimate subgraph probabilities is a new application of GGMs.

1. The Generalizable Graph Masked Auto-Encoder (GiGaMAE) is a self-supervised generative model designed to enhance the generalization capabilities of node representations in graphs [29]. This model uses a graph masked auto-encoder framework where edges and features of the graph are masked, and the resulting masked graph is encoded by a GNN to generate latent node representations. Instead of directly reconstructing the original graph's edges or features, GiGaMAE reconstructs embeddings from Node2Vec for structural information and PCA for attribute information, as targets. For node classification and link prediction, GiGaMAE leverages the learned node embeddings, which are optimized through a mutual information-based reconstruction

loss that balances shared and distinct information across multiple targets. Although this model can perform both node classification and link prediction, it requires separate training for each task. Consequently, the generated node embeddings are different and specifically customized for each individual task.

2. GraphSAGE is an inductive model designed to generate low-dimensional embeddings for nodes within large graphs [8]. GraphSAGE learns embeddings for the new nodes in a graph by aggregating information from their local neighborhood and tries to capture the structural properties of the graph by looking at the nodes and links surrounding each node. In this study, I employed GraphSAGE for both link prediction and node classification tasks. For node classification, I utilized the supervised method outlined in the original paper [8]. For link prediction, GraphSAGE was utilized as an encoder, trained in the unsupervised mode. I trained a separate link predictor that takes as input two node embeddings and outputs a probability indicating whether a link exists between the two nodes.

3. Graph Attention Networks (GATs) are inductive models that use attention mechanisms to generate node embeddings in graphs [33]. Each node's features are initially transformed linearly, and the importance of neighboring nodes is determined through learnable attention coefficients, which are computed using a shared attentional mechanism and then normalized via the softmax function. The node embeddings are generated by aggregating the features of the neighboring nodes, weighted by these normalized attention coefficients. This method allows GATs to dynamically focus on the most relevant parts of the neighborhood for each node. For link prediction, I trained a separate link decoder that takes as input two GAT node embeddings and outputs a probability, indicating whether a link exists between the two nodes. And for node classification, I followed the original paper [33].

2.7 Subgraph Classification

Subgraph classification is a task in graph learning that focuses on identifying and categorizing substructures within a larger graph, with significant challenges related to feature extraction and computational efficiency. SUBGNN is a subgraph neural network that learns disentangled subgraph representations [1]. Like our approach, their query targets and evidence are subgraphs. However, their system addresses only *classifying* the target subgraph with a single label. In contrast, I assign subgraph probabilities.

Chapter 3

Method and Approach

This chapter defines probabilistic subgraph queries, categorizes different kinds of graph queries, and proposes a method to answer them using VGAE models.

3.1 Problem Statement

Subgraph queries (SQs) specify a set of links and labels for a set of query nodes. SQs allow users to query over nodes and links in graphs and return probabilistic answers to those queries. In this research, I introduce a simple subgraph query syntax that uses combinations of links, node attributes, and labels to represent prediction tasks commonly studied in graph learning. (Chapter 2).

Data Format. An attributed labelled graph is a pair $G = (V, E)$ comprising a finite set of nodes V with features and labels, and links E, which may be positive (present) or negative (absent). Each node is assigned a k-dimensional attribute $\boldsymbol{x}_i$ with $k > 0$ and a node label $\boldsymbol{l}_i$. A graph with N nodes can be represented by the following objects.

- An $N \times N$ adjacency matrix $\boldsymbol{A}$ with $\{0, 1\}$ Boolean entries.
- An $N \times k$ node feature matrix $\boldsymbol{X}$ with $\{0, 1\}$ Boolean entries.
- An $N \times l$ node label matrix Ł with a one-hot encoding of l discrete labels for each node.

A target subgraph of graph $G = (V, E)$ is a graph $G^{\boldsymbol{Y}} = (V^{\boldsymbol{Y}}, E^{\boldsymbol{Y}})$ where $V^{\boldsymbol{Y}} \subseteq V$ and $E^{\boldsymbol{Y}} \subseteq E$. Similarly, an evidence subgraph of graph G is a graph $G^{\boldsymbol{E}} = (V^{\boldsymbol{E}}, E^{\boldsymbol{E}})$ where $E^{\boldsymbol{E}} \subseteq E$ and $E^{\boldsymbol{E}} \cap E^{\boldsymbol{Y}} = \emptyset$. We refer to the nodes $V^{\boldsymbol{E}}$ that appear in the evidence as **evidence nodes**, to the nodes $V^{\boldsymbol{Y}}$ that appear in the target as **target nodes**, and to their union as **query nodes** (i.e., $V = V^{\boldsymbol{E}} \cup V^{\boldsymbol{Y}}$).

Definition of Subgraph Queries. A relational random variable corresponds to either a node attribute or an adjacency. Let $\mathcal{A} = \{\boldsymbol{A}[u, v] : u \in V, v \in V\}$ be the set of **link variables**, and $\mathcal{L} = \{\boldsymbol{L}[u] : u \in V\}$ be the set of **node label variables**, and $\mathcal{X} = \{\boldsymbol{X}[u] : u \in V\}$ be the set of **feature variables**.

A **subgraph query** $\mathbb{P}(\textit{target}|\textit{evidence})$ is of the form

$$\mathbb{P}(\mathcal{A}^{\boldsymbol{Y}} = \boldsymbol{a}^{\boldsymbol{Y}}, \mathcal{L}^{\boldsymbol{Y}} = \boldsymbol{l}^{\boldsymbol{Y}}, \mathcal{X}^{\boldsymbol{Y}} = \boldsymbol{x}^{\boldsymbol{Y}} | \mathcal{A}^{\boldsymbol{E}} = \boldsymbol{a}^{\boldsymbol{E}}, \mathcal{L}^{\boldsymbol{E}} = \boldsymbol{l}^{\boldsymbol{E}}, \mathcal{X}^{\boldsymbol{E}} = \boldsymbol{x}^{\boldsymbol{E}})$$

where

- $\mathcal{A}^{\boldsymbol{Y}} = \{\boldsymbol{A}[u_i, v_i] : i = 1, \ldots, |\mathcal{A}^{\boldsymbol{Y}}|, \boldsymbol{A}[u_i, v_i] \in E^{\boldsymbol{Y}}\}$ is the list of binary target links, each assigned a value $a_i \in \{0, 1\}$. Similarly, $\mathcal{A}^{\boldsymbol{E}} = \{\boldsymbol{A}[u_i, v_i] : i = 1, \ldots, |\mathcal{A}^{\boldsymbol{E}}|, \boldsymbol{A}[u_i, v_i] \in E^{\boldsymbol{E}}\}$ is the list of binary evidence links.
- $\mathcal{L}^{\boldsymbol{Y}} = \{\boldsymbol{L}[u_i] : i = 1, \ldots, |\mathcal{L}^{\boldsymbol{Y}}|, u_i \in V^{\boldsymbol{Y}}\}$ is the list of target node labels, each assigned a one-hot encoding of the l class labels $l_i \in \{0, 1\}^l$. Similarly, $\mathcal{L}^{\boldsymbol{E}} = \{\boldsymbol{L}[u_i] : i = 1, \ldots, |\mathcal{L}^{\boldsymbol{E}}|, u_i \in V^{\boldsymbol{E}}\}$ is the list of evidence node labels.
- $\mathcal{X}^{\boldsymbol{Y}} = \{\boldsymbol{X}[u_i] : i = 1, \ldots, |\mathcal{X}^{\boldsymbol{Y}}|, u_i \in V^{\boldsymbol{Y}}\}$ is the list of target node features, each assigned a feature vector $\boldsymbol{x}_i \in \mathbb{R}^{1 \times k}$. Similarly, $\mathcal{X}^{\boldsymbol{E}} = \{\boldsymbol{X}[u_i] : i = 1, \ldots, |\mathcal{X}^{\boldsymbol{E}}|, u_i \in V^{\boldsymbol{E}}\}$ is the list of evidence node features.

The indices i index elements in a query not nodes in a graph. Given a partition of nodes into observed training nodes and unobserved test nodes, a query is **inductive** if a test node appears in the target nodes.

Note that the target and evidence specifications can be and typically are *partial* in that some graph components are unspecified. For example, for two evidence nodes u and v, the evidence may specify their features, but not whether there exists a link between them or not.

Our definition of subgraph query treats a node feature $\boldsymbol{x}[u]$ as a group, in that either all or no features of node u are specified in a query. This restriction is not essential; we use it mainly to simplify notation.

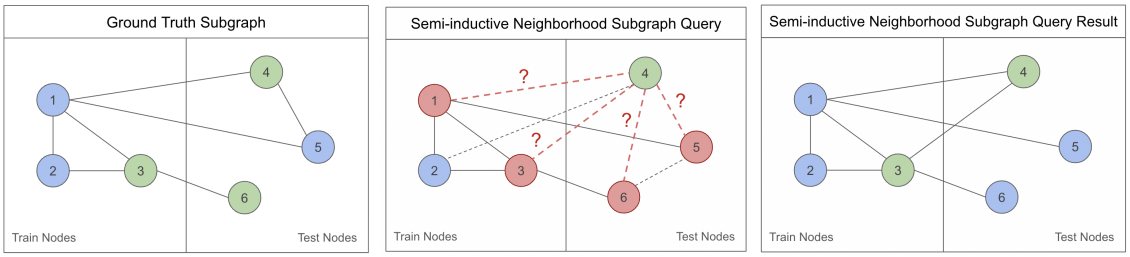
3.1.1 Example of Semi-Inductive Subgraph Queries

Figure 3.1b shows an example of Semi-inductive Neighborhood Query, the most complex SQ type I consider in this research (Section 4.3), and Figure 3.2b shows an example of Semi-inductive Single Neighbor Query (Section 4.3). The query nodes are 1–6, nodes 1–3 are training nodes, and nodes 4–6 are testing nodes. Figures 3.1a and 3.2a show the ground truth graph with a partition of nodes. Node labels are green and blue. Figures 3.1b and 3.2b show examples for neighborhood and single neighbor queries, respectively. In these figures, target labels and links are colored red. Black dashed links are unspecified links in

evidence. Figures 3.1c and 3.2c show the output of the model for neighborhood query and single neighbor query, where the model has specified the existence of red target links and the label of the target nodes.

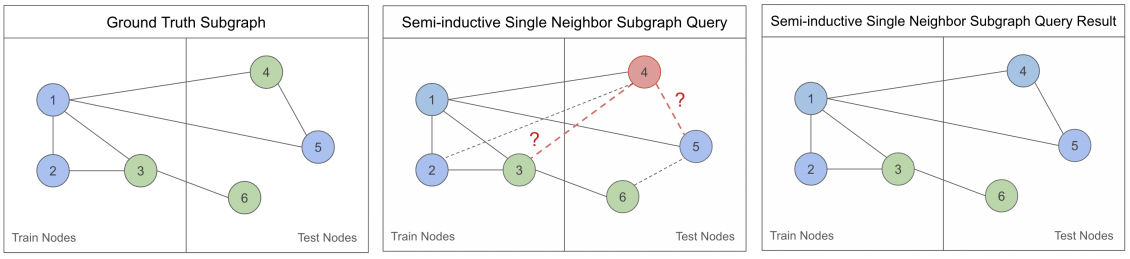
For Figure 3.1, the query target is to predict the labels of nodes 1, 3, 5, and 6, and the target links. The positive target links comprise the pairs $(1, 4)$ and $(5, 4)$, and the negative target links comprise the pairs $(3, 4)$ and $(6, 4)$. The query evidence comprises all 6 node feature vectors (not shown). Positive evidence links are $(1, 2), (1, 3), (2, 3), (3, 6)$, and $(1, 5)$. Negative evidence links are $(3, 5), (2, 6), (2, 5)$, and $(1, 6)$. The links (2, 4) and (5, 6) are represented by black dashed links and are unspecified. As you can see in Figure 3.1c, the model has predicted the correct labels for nodes $1, 3$, and 5 and wrong label for node 6. Additionally, for link prediction, we have correct predictions for links $(1, 4), (3, 4)$, and $(6, 4)$ and wrong prediction for link $(5, 4)$.

For Figure 3.2, the query target is to predict the label of node 4, and the target links. The positive target link is $(6, 4)$, and the negative target link is $(3, 4)$. The query evidence comprises all 6 node feature vectors (not shown). Positive evidence links are $(1, 2), (1, 3), (2, 3), (3, 6), (1, 5)$, and $(1, 4)$. Negative evidence links are $(3, 5), (2, 6), (2, 5)$, and $(1, 6)$. The links (2, 4) and (5, 6) are represented by black dashed links and are unspecified. As you can see in Figure 3.2c, the model has predicted the wrong label for node 4. Also, for link prediction, we have correct predictions for links $(3, 4)$ and $(6, 4)$.



(a) Input graph (b) Neighborhood query (c) The output of the model.

Figure 3.1: An example of Semi-inductive Neighborhood Query



(a) Input graph (b) Single Neighbor query. (c) The output of the model.

Figure 3.2: An example of Semi-inductive Single Neighbor Query

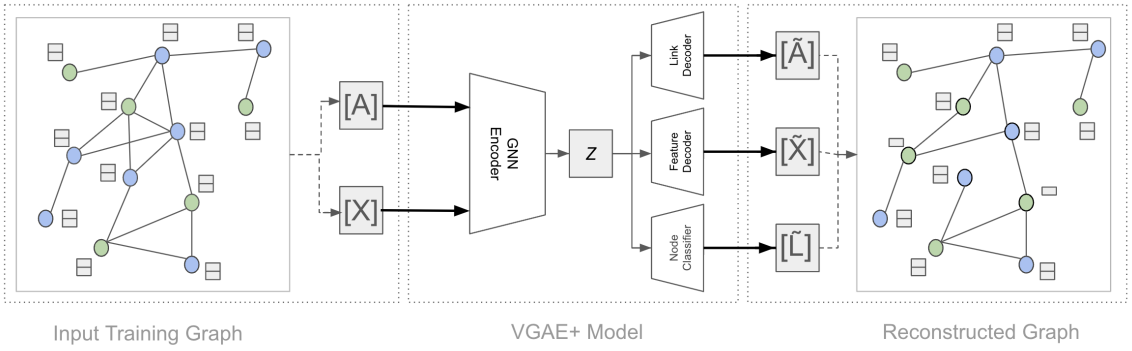


Figure 3.3: Model Architecture: First, the input graph is represented by an adjacency matrix $\boldsymbol{A}$ and a feature matrix $\boldsymbol{X}$. These two matrices are the input of the model. Then, the encoder generates latent node embeddings $\boldsymbol{Z}$. Finally, the decoders generate reconstructed adjacency matrix $\tilde{\boldsymbol{A}}$, reconstructed feature matrix $\tilde{\boldsymbol{X}}$, and reconstructed node labels matrix $\tilde{\boldsymbol{L}}$. Then the reconstructed graph is represented using $\tilde{\boldsymbol{A}}$, $\tilde{\boldsymbol{X}}$, and $\tilde{\boldsymbol{L}}$

In this work, I mask the $\boldsymbol{A^Y}$ in the original matrix with zeroes to reconstruct these values by using the encoder's embeddings and our decoders. Since my system performs unsupervised node classification and my model does not use node labels as input, I do not use labels as training input.

3.2 Model Definition

In this thesis, I take the Variational Graph Auto-Encoder (VGAE) [16] as the basic generative model for answering SQs. This section describes VGAE+, a VGAE augmented with feature and label reconstruction along with training and implementation details. Figure 3.3 shows the VGAE model architecture. Given a training graph $G = (V, \boldsymbol{A}, \boldsymbol{X}, \boldsymbol{L})$, the graph encoder takes as input the adjacency matrix $\boldsymbol{A}$ and then the feature matrix $\boldsymbol{X}$ and outputs node embeddings $\boldsymbol{Z}$. Then the model reconstructs graph G by reconstructing $\boldsymbol{A}$, $\boldsymbol{X}$, and node label matrix $\boldsymbol{L}$ and outputs graph $G' = (V, \boldsymbol{A}', \boldsymbol{X}', \boldsymbol{L}')$.

3.2.1 Augmented VGAE Generative Model

In the VGAE model, links are generated independently, given node embeddings. Following the GraphVAE approach [30], I generate node classes and node features independently as well, given node embeddings. Let $\boldsymbol{Z}$ be an $N \times d$ matrix that represents latent node embeddings. I utilize three decoder models (see Figure 3.3):

$$
\begin{aligned}
p_\theta(\boldsymbol{A}|\boldsymbol{Z}) &= \prod_{u,v} p_\theta(\boldsymbol{A}[u,v]|\boldsymbol{z}[u], \boldsymbol{z}[v]) \\
p_\psi(\boldsymbol{X}|\boldsymbol{Z}) &= \prod_{u} p_\psi(\boldsymbol{X}[u]|\boldsymbol{z}[u]) \\
p_\phi(\boldsymbol{L}|\boldsymbol{Z}) &= \prod_{u} p_\phi(\boldsymbol{L}[u]|\boldsymbol{z}[u])
\end{aligned}
\tag{3.1}
$$

where $p_\theta : \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$ is a trainable **link decoder**, and p_ψ resp. p_ϕ denotes a trainable **feature decoder** resp. **label decoder**.

The graph **encoder** $q_\phi(\boldsymbol{z}|\boldsymbol{X}, \boldsymbol{A})$ is implemented by a GNN that takes as input an attribute graph and returns latent node embeddings. For compatibility with baseline methods, the encoder does not receive node labels as input, but adding them is straightforward. A VGAE+ is trained using the variational ELBO objective [16, 9] as follows:

$$\begin{aligned}\mathcal{L}(\theta, \phi) = -E_{\boldsymbol{Z}\sim q_\phi(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A})}[\alpha \times \ln p_\theta(\boldsymbol{A}|\boldsymbol{Z}) \\ + \beta \times \ln p_\psi(\boldsymbol{X}|\boldsymbol{Z}) + \gamma \times \ln p_\phi(L|\boldsymbol{Z})] \\ + KL\big(q_\phi(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{A})||p(\boldsymbol{Z})\big).\end{aligned} \tag{3.2}$$

Where $KL(.||.)$ is Kullback-Leibler divergence which measures difference between two probability distributions. $p(\boldsymbol{Z})$ is prior distribution of $\boldsymbol{Z}$ which is usually taken as a standard Gaussian distribution. The approximate posterior $q_\varphi(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{A})$ is the graph encoder. The reconstruction likelihood $p_\phi(L|\boldsymbol{Z})$, $p_\psi(\boldsymbol{X}|\boldsymbol{Z})$, and $p_\theta(\boldsymbol{A}|\boldsymbol{Z})$ are computed by the decoders of the VGAE+. Here α, β and γ are hyperparameters that weight the importance of different reconstruction tasks. The higher the values of these hyperparameters, the greater the effects they have on the loss function. Equation (3.2) is the training objective used in our experiments.

Data Splitting

In order to answer inductive queries involving unseen nodes, I train the VGAE+ without node IDs. The **input graph** is one large graph $G_{in} = (V, E)$. Suppose the nodes in the graph are divided into three categories: training nodes V_{tr}, inductive test nodes V_{te}, and validation nodes V_{va}. I randomly partition the nodes in the input graph as follows: 70% training nodes V_{tr}, 20% inductive test nodes V_{te}, and 10% validation nodes V_{va}. To train an inductive model, the training graph is the input subgraph induced by V_{tr}, which excludes test nodes. To train the model properly for node classification, I make sure that the training nodes V_{tr} include nodes from all the classes in each dataset.

Bayesian Optimization

The weight hyperparameters α, β, γ are found by Bayesian optimization [24]. Bayesian optimization is a sequential model-based optimization technique used for optimization. It is an algorithm for finding the best set of parameters for a given objective function. Bayesian optimization efficiently searches the parameter space and often finds optimal or near-optimal solutions with fewer function evaluations compared to grid search or random search methods by intelligently selecting points to evaluate.

The optimizer optimizes these parameters over the interval [0,1] so that they minimize the value of the reconstruction loss:

$$min_{\alpha,\beta,\gamma} E_{\boldsymbol{Z}\sim q_\phi(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{A})}[\ln p_\theta(\boldsymbol{A}|\boldsymbol{Z}) + \ln p_\psi(\boldsymbol{X}|\boldsymbol{Z}) + \ln p_\phi(L|\boldsymbol{Z})] \tag{3.3}$$

which is the model score on the validation graph. The validation graph comprises all validation nodes, their neighbors in the input graph, and all links that involve at least one validation node. $CE(Label)$ and $BCE(Features)$ are computed over the labels of all validation nodes. $BCE(Link)$ is computed over all validation graph links.

3.2.2 Implementation

This section details the specific implementation of VGAE+, focusing on the encoder, link decoder, feature decoder, and node classifier.

Encoder

The graph **encoder** $q_\varphi(\boldsymbol{z}|\boldsymbol{X},\boldsymbol{A})$ is implemented by a GNN that takes an attributed labeled graph as input and returns node embeddings $\boldsymbol{Z}$. The node embeddings are independent, and for each node, represent a conditional Gaussian distribution, such that

$$q(\boldsymbol{z}[u]|\boldsymbol{X},\boldsymbol{A}) \sim N(\mu[u],\sigma[u]) \tag{3.4}$$

with mean and covariance $\mu[u],\sigma[u]$ for node u.

Link Decoder

As a strong link decoder, I utilize a Stochastic Block Model (SBM) [21], which is defined as:

$$p_\theta(\boldsymbol{A}[u,v]|\boldsymbol{z}[u],\boldsymbol{z}[v]) = \boldsymbol{z}[u]^\top \Lambda \boldsymbol{z}[v] \tag{3.5}$$

where $\Lambda \in R^{d\times d}$ is the trainable d-block matrix in the SBM.

Feature Decoder

Node features are reconstructed independently, given node embeddings. Our benchmark datasets comprise discrete features, so our *feature decoder* is of the form $p_\psi : \mathbb{R}^d \to \{0,1\}^k$

$$p_\psi(\boldsymbol{X}|\boldsymbol{Z}) = \prod_{u=1}^{N}\prod_{d=1}^{k} p_\psi(\boldsymbol{X}_d[u]|\boldsymbol{z}[u]) = \prod_{u=1}^{N}\prod_{d=1}^{k} \sigma(\tilde{x}_d[u]), \tag{3.6}$$

where ψ are the parameters of a fully connected neural net feature decoder that maps a node embedding $\boldsymbol{z}[u]$ to a k-dimensional node feature reconstruction $\tilde{x}[u]$.

Node Classifier

The *node classifier* $p_\phi : \mathbb{R}^d \to \{0,1\}^l$, classifies the node labels independently under a one-hot encoding, given the same node embeddings:

$$p_\phi(\boldsymbol{L}|\boldsymbol{Z}) = \prod_{u=1}^{N} p_\phi(\boldsymbol{L}[u]|\boldsymbol{z}[u]) = \prod_{u=1}^{N} softmax(\tilde{l}[u]) \tag{3.7}$$

where ϕ are the parameters of a fully connected node classifier that maps a node embedding $\boldsymbol{z}[u]$ to a l-dimensional vector $\tilde{l}[u]$.

3.3 Subgraph Inference from a VGAE Model

Consider a subgraph query Equation 3.1 with n query nodes:

$$\mathbb{P}(\mathcal{A}^{\boldsymbol{Y}} = \boldsymbol{a}^{\boldsymbol{Y}}, \mathcal{L}^{\boldsymbol{Y}} = \boldsymbol{l}^{\boldsymbol{Y}}, \mathcal{X}^{\boldsymbol{Y}} = \boldsymbol{x}^{\boldsymbol{Y}} | \boldsymbol{E} = \boldsymbol{e}) \tag{3.8}$$

where $\boldsymbol{E}$ is a list of evidence variables. (See Figures 3.2b and 3.1b) I define two subgraph inference models, deterministic (Det) and Monte Carlo (MC), for the VGAE+ generative model. Essentially, the deterministic method uses node embeddings computed deterministically from the evidence, whereas the MC method samples node embeddings conditional on the evidence.

3.3.1 Inference Models

For a fixed set of n query node embeddings $\boldsymbol{z}$, the target probability is given by

$$\mathbb{P}_{\boldsymbol{\eta}}(\mathcal{A}^{\boldsymbol{Y}} = \boldsymbol{a}^{\boldsymbol{Y}}, \mathcal{L}^{\boldsymbol{Y}} = \boldsymbol{l}^{\boldsymbol{Y}}, \mathcal{X}^{\boldsymbol{Y}} = \boldsymbol{x}^{\boldsymbol{Y}} | \boldsymbol{z}) \tag{3.9}$$

and can be computed by multiplying the independent decoder probabilities (Equation 3.5, 3.6, and 3.7) parameterized by $\boldsymbol{\eta}$. The **posterior distribution** over the n node embeddings is a conditional Gaussian distribution, such that $p(\boldsymbol{z}[u]|\boldsymbol{E} = \boldsymbol{e}) \sim N(\mu[u], \sigma[u])$ with mean and covariance $\mu[u], \sigma[u]$ for node u. Equation 3.12 below shows how to approximate the posterior using the GNN encoder. Let $\boldsymbol{\mu_E}$ be the mean node embeddings from the posterior distribution.

Deterministic Inference This inference method computes the mean of node embeddings using the prior network, then passes the computed mean as final embedding to each decoder to compute a target probability. If we have m targets in $\boldsymbol{Y}$, then the query probability is computed as follows:

$$\mathbb{P}(\mathcal{A}^{\boldsymbol{Y}}, \mathcal{L}^{\boldsymbol{Y}}, \mathcal{X}^{\boldsymbol{Y}} | \boldsymbol{E} = \boldsymbol{e}) \approx \mathbb{P}(\mathcal{A}^{\boldsymbol{Y}}, \mathcal{L}^{\boldsymbol{Y}}, \mathcal{X}^{\boldsymbol{Y}} | \boldsymbol{z}^{\mu_{\boldsymbol{E}}}). \tag{3.10}$$

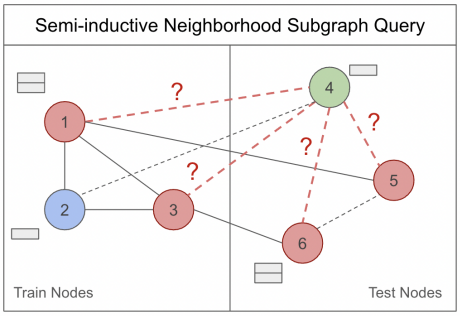


Figure 3.4: An example of a semi-inductive subgraph query: Red dashed links are target links, and black dashed links are unspecified links. Red colored nodes are target nodes for node classification. The squares beside each node represent features for that node.

where $\boldsymbol{z}^{\mu_{\boldsymbol{E}}} = \boldsymbol{\mu}_{\boldsymbol{E}}$.

Monte Carlo Inference This inference method samples S node embeddings from the posterior distribution, and averages the subgraph probabilities:

$$\mathbb{P}(\mathcal{A}^{\boldsymbol{Y}}, \mathcal{L}^{\boldsymbol{Y}}, \mathcal{X}^{\boldsymbol{Y}} | \boldsymbol{E} = \boldsymbol{e}) \approx \frac{1}{S} \sum_{s=1}^{S} \mathbb{P}(\mathcal{A}^{\boldsymbol{Y}}, \mathcal{L}^{\boldsymbol{Y}}, \mathcal{X}^{\boldsymbol{Y}} | \boldsymbol{z}^s) \tag{3.11}$$

where $\boldsymbol{Z}^s \sim \prod_{u=1}^{n} p(\boldsymbol{z}[u] | \boldsymbol{E} = \boldsymbol{e})$.

Posterior Distribution $p(\boldsymbol{Z} | \boldsymbol{E} = \boldsymbol{e})$

Training a VGAE+ model provides decoder models $p_{\boldsymbol{\eta}}$ and an encoder q_ϕ. The challenge is that the encoder assumes as input an adjacency matrix, or equivalently, a complete subgraph induced by the query nodes. I bridge the gap between a partially specified subgraph and a complete induced subgraph by imputing graph components missing from the evidence with 0 as a default value. For missing links, using the 0 default is the approach taken in previous work [23, 16]. As discussed by [23], 0 default is appropriate for links: First, a message-passing encoder treats 0 links as uninformative, which is appropriate for unspecified links. Second, because of graph sparsity, the mode of the true link posterior given the evidence is close to 0. For node features and labels, the 0 default is also appropriate given our one-hot encoding, since the VGAE+ encoder is trained not to propagate information from 0-valued features or labels.

Formally, to obtain **evidence embeddings**, I approximate the posterior distribution using the trained encoder q_ϕ:

$$p(\boldsymbol{Z} | \boldsymbol{E} = \boldsymbol{e}) \approx q_\phi(\boldsymbol{Z} | \boldsymbol{A}^{\boldsymbol{E},\boldsymbol{0}}, \boldsymbol{X}^{\boldsymbol{E},\boldsymbol{0}}) \tag{3.12}$$

where the *evidence matrices* $\boldsymbol{A}^{\boldsymbol{E},\boldsymbol{0}}$ and $\boldsymbol{X}^{\boldsymbol{E},\boldsymbol{0}}$ are defined as follows: 1) The dimension of $\boldsymbol{A}^{\boldsymbol{E},\boldsymbol{0}}$ is $n \times n$. If $\mathcal{A}^{\boldsymbol{E}}$ specifies a link assignment $\mathcal{A}[u, v] = e_i$, then $\boldsymbol{A}^{\boldsymbol{E},\boldsymbol{0}}[u, v] := e_i$;

otherwise $\boldsymbol{A}^{\boldsymbol{E},\mathbf{0}}[u,v] := 0$. 2) The dimension of $\boldsymbol{X}^{\boldsymbol{E},\mathbf{0}}$ is $n \times k$. If $\mathcal{X}^{\boldsymbol{E}}$ specifies a feature vector $\mathcal{X}[u] = \boldsymbol{x}_i$, then $\boldsymbol{X}^{\boldsymbol{E},\mathbf{0}}[u] := \boldsymbol{x}_i$; otherwise $\boldsymbol{X}^{\boldsymbol{E},\mathbf{0}}[u] := \mathbf{0}$. Similarly, we can assign a default value of 0 to unspecified node labels, but our experiments do not utilize queries that include node labels as evidence. Note that while the number of training nodes may be very large, the number of query nodes is typically small (on the order of 10-100). Applying the same encoder to subgraphs of different sizes is possible because I train the VGAE+ model inductively.

To illustrate, consider Figure 3.4. Since the link between nodes 1 and 2 is specified to exist, the evidence adjacency matrix assigns $\boldsymbol{A}^{\boldsymbol{E},\mathbf{0}}[1,2] := 1$. The link between nodes 3 and 5 is specified not to exist, so the evidence matrix assigns $\boldsymbol{A}^{\boldsymbol{E},\mathbf{0}}[3,5] := 0$. Since the link between nodes 5 and 6 is unspecified, the evidence matrix assigns $\boldsymbol{A}^{\boldsymbol{E},\mathbf{0}}[5,6] := 0$. The feature vector for node 3 is unspecified, the evidence matrix assigns the zero feature vector $\boldsymbol{X}^{\boldsymbol{E},\mathbf{0}}[2] := \mathbf{0}$. The following matrices demonstrate $\mathcal{A}^{\boldsymbol{E}}$, $\boldsymbol{A}^{\boldsymbol{E},\mathbf{0}}$, $\mathcal{X}^{\boldsymbol{E}}$, and $\boldsymbol{X}^{\boldsymbol{E},\mathbf{0}}$ for the example query in Figure 3.4, where links (2, 4) and (5, 6) and features for nodes 3, 5 are unspecified. Unspecified values are shown with "?":

- $\mathcal{A}^{\boldsymbol{E}} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & ? & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & ? & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & ? \\ 0 & 0 & 1 & 0 & ? & 1 \end{bmatrix}$
- $\boldsymbol{A}^{\boldsymbol{E},\mathbf{0}} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$
- $\mathcal{X}^{\boldsymbol{E}} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ ? & ? \\ 1 & 0 \\ ? & ? \\ 1 & 1 \end{bmatrix}$

- $\boldsymbol{X}^{\boldsymbol{E},\mathbf{0}} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$

Chapter 4

Experiments

I describe our benchmark datasets, the design of the test queries, and how the evaluation metrics are computed.

4.1 Datasets

To evaluate all the methods, I utilize datasets from previous studies of GGMs [16, 41, 10].

- **Cora** is a citation dataset that consists of nodes that represent machine learning papers divided into seven classes. In this dataset nodes represent papers and links represent citations among them [28].
- **ACM** is a citation dataset. It has three types of nodes (author, paper, subject) and two types of edges (paper-author, paper-subject) [41].
- **IMDb** is a movie dataset with three types of nodes (movie, actor, director) and two types of edges (movie-actor, movie-director). In this dataset labels are the genres of movies [41].
- **CiteSeer** is also a citation dataset which consists of nodes that represent machine learning papers divided into six classes. In this dataset nodes represent papers and links represent citations among them [28].
- **Photo & Computers** are datasets from the Amazon co-purchase graph [20]. In these datasets, nodes represent goods, links indicate that two goods are frequently bought together, node features are bag-of-words encoded product reviews, and class labels are given by the product category [10].

Table 4.1 presents statistics for all 6 benchmark datasets. In this table *edge density* refers to the number of links in a graph divided by the total number of possible links in that graph. This metric measures the connectedness or compactness of a graph. A lower number indicates a sparser graph, meaning there are fewer connections between the nodes.

Dataset	Nodes	Links	Edge Density	Average Node Degree
Cora [16]	2,708	5,429	0.00074	3.8
ACM [41]	8,993	18,929	0.00093	2.2
IMDb [41]	12,772	19,120	0.00046	2.9
CiteSeer [16]	3,327	4,732	0.00171	2.7
Photo [10]	7,650	238,162	0.01629	36.7
Computers [10]	13,752	491,722	0.01041	36.7

Table 4.1: Statistics of the datasets.

The *average node degree* in a graph is the average of the degrees of all the nodes in the graph.

4.2 Data Preprocessing

Following previous link prediction studies [16], I add self-loops and make all links undirected (i.e., if the training data contains an adjacency, $v \to u$, it also contains $u \to v$). Cora, CiteSeer, Photo, and Computers are homogeneous datasets, whereas ACM and IMDb are heterogeneous datasets. Since our comparison methods use homogeneous GNNs, I homogenize different edge types so that every edge in the adjacency matrix is represented by 0 for no link and 1 for link existence.

4.3 Test Query Design

In all queries, the evidence $\boldsymbol{E}$ does not contain node labels, but specifies: (1) the node feature vector for each query node, (2) the non-target links from the input graph. (e.g. in Section 3.1.1.) To define query targets, I randomly select a set of 100 test nodes as target nodes from the test nodes V_{te}. Evidence links vary depending on the inductive or semi-inductive query type. (See Section 3.2.1). Target links and nodes depend on the query type as follows.

Single Neighbor Queries For each target node $u \in V_{te}$, I randomly select two test links, one positive pair (u, v_+) from the neighborhood of u, and one negative pair (u, v_-) from outside the neighborhood. The resulting query is of the form $P(\boldsymbol{L}[u], \boldsymbol{A}[u, v_+], \boldsymbol{A}[u, v_-]|\boldsymbol{E})$: Which of the two links is true and what is the label of the target node (Figure **??**.)?

Neighborhood Queries For each target node $u \in V_{te}$ with at least one neighbor, suppose it has $deg(u)$ neighbors. Let $v_1^+, \ldots, v_{deg(u)}^+$ enumerate the nodes in the neighborhood of u. I randomly select $|deg(u)|$ negative test nodes $v_1^-, \ldots, v_{deg(u)}^-$. The resulting query is of the form $\mathbb{P}(\{\boldsymbol{L}[v_i^+], \boldsymbol{A}[u, v_i^+], \boldsymbol{L}[v_i^-], \boldsymbol{A}[u, v_i^-] : i = 1, \ldots, deg(u)\}|\boldsymbol{E})$: Which nodes are neighbors and what are their labels (Figure A.1b.)?

4.4 Metrics

For single-neighbor queries, I compute the mean of each metric across all test queries. For neighborhood queries, the metric is calculated separately for each query, over all the components of the query (e.g., all the neighbors to be predicted). Then, I report the mean of each metric across all neighborhood queries. I score link prediction and node classification separately using different metrics. Separate scoring for each downstream task is supported by the VGAE+ inference model (3.9), but in fact, favors the single-task prediction baselines over our joint prediction model. The reason for independent scoring for one task is that our baseline methods are not designed for subgraph queries. Therefore, I use them only to obtain scores for individual tasks. The joint predictive performance for a subgraph query is measured by the average AUC score and AP score over predicted links and node labels. I have evaluated the performance of our model on all three tasks of link prediction, node classification, and subgraph queries, using three sets of metrics:

4.4.1 Metrics for Subgraph Queries

- **ROC-AUC** measures the area under the Receiver Operating Characteristic (ROC) curve. This metric measures the performance of the model across all possible classification thresholds.
- **Average Precision (AP)** measures the weighted mean of precision over all possible classification thresholds.

4.4.2 Metrics for Link Queries

- **ROC-AUC**
- **AP**
- **Hit-Rate (HR)** is computed as follows: (1) For each method, find the set T of the top 20% links as ranked by the method. (2) Find the number of ground-truth links in T. (3) Divide by the total number of links in T.

4.4.3 Metrics for Node Queries

- **ROC-AUC**
- **AP**
- **F1-score macro** is a metric used to evaluate the performance of a classification model, especially in situations where the dataset is imbalanced. It is the harmonic mean of precision and recall, calculated separately for each class, and then averaged. This ensures that each class is given equal weight, regardless of its prevalence in the

dataset. F1-score macro is particularly useful because it highlights the performance of the model across all classes, preventing high performance on a dominant class from overshadowing poor performance on minority classes. This makes it a valuable metric for assessing models in multi-class and imbalanced data scenarios.

4.5 Experimental Setup

I built VGAE+ on the VGAE implementation for joint link prediction [23], following the authors' recommendation of using MC inference for link prediction and a graph isomorphism network (GIN) encoder [40] with 2 layers and embedding dimension 128. I extended their VGAE architecture to include the reconstruction of node features and labels, as described in Section 3.2. Node labels are predicted using the node classifier (Equation 3.7), trained end-to-end. I give results for deterministic prediction (Equation 3.10) and MC prediction (Equation3.11) with 30 samples. (For the GitHub repository for VGAE+, see Section A.1)

4.6 Baselines

I select baselines that are well established and represent a variety of approaches to node classification and single link prediction. Node classification methods perform joint label prediction (known as semi-supervised node classification [38]) similar to our VGAE+ model. I apply single link prediction methods to a set of target links separately to obtain a probability for each target link (cf. [23].) I organize methods according to whether they were originally evaluated on both link prediction and node classification, or just one or the other.

4.6.1 Two-Task Methods

These frameworks train two separate link prediction and node classification models using the same GNN architecture.

GiGaMAE Generalizable Graph Masked Auto-Encoder [29] is a self-supervised generative model that employs a graph masked auto-encoder framework. GiGaMAE reconstructs embeddings from Node2Vec for structural information and PCA for attribute information based on the downstream task (see Sections 2). I use the authors' code to obtain node label and link prediction probabilities. For link prediction, GiGaMAE utilizes a decoder that assigns a score to each node pair based on the similarity between their embeddings. For node classification, the decoder uses logistic regression.

GraphSAGE [8] is an inductive model. I have used the DGL implementation (SAGEConv) [36], with the supervised training mode, for node classification, where the decoder uses the node labels. For node classification, it uses a simple feedforward neural network architecture as a node classifier. For link prediction, I train an SBM link decoder end-to-end using their unsupervised training.

GAT are a kind of GNN that computes different weights for different nodes in a neighborhood [33]. I have used the DGL implementation (GAT) [36], with the supervised training mode, for node classification, where the decoder sees the node labels. GAT embedding system trained with the same micro-F_1 score as in the original paper [33]. For link prediction, I train an SBM link decoder end-to-end using their unsupervised training.

4.6.2 Link Prediction Baselines

These are the baselines for link prediction tasks:

SEAL is a well-known method for transductive link prediction [43]. I train SEAL inductively by omitting node IDs (cf. [32, 23]). This is similar to the approach of GraIL, here is used for homogeneous graphs rather than knowledge graphs. The basic idea of SEAL is to embed the subgraph around a target link.

DEAL was designed for both transductive and inductive cold-start link prediction tasks [10]. The distinguishing feature of the DEAL approach is that at test time, it uses only the node attributes for link prediction. This makes it a strong baseline for independent prediction since it is not sensitive to the presence of evidence links.

4.6.3 Node Classification Baselines

These are the baselines for node classification tasks:

MVGRL is an inductive self-supervised approach for learning representations of nodes and graphs by contrasting different structural views of graphs [11]. I use the authors' code to train a node classification model.

DeepWalk is a transductive graph embedding method that learns node representations by treating random walks on the graph as sentences and applying word embedding techniques [25]. It captures structural information of the graph by mapping nodes to low-dimensional vectors in a continuous space. The reported results are based on the DGL implementation and pertain only to the transductive setting, as DeepWalk cannot be adapted for the inductive setting. I trained a logistic regression model for node classification [25].

4.6.4 Baseline Setup

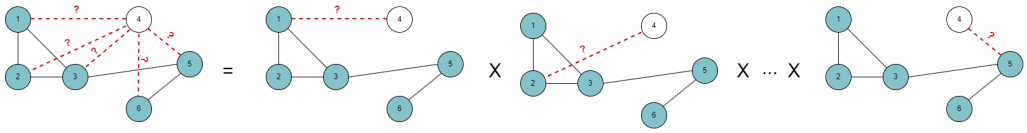


Figure 4.1: Predicting multiple links by computing independent link probabilities for each target link.

For all hyperparameters, I apply the same settings as reported in the original papers. I also use the same split for better evaluation of training, validation, and test nodes in all experiments.

Joint Link Queries I extend the single link prediction methods to predict multiple links by computing independent link probabilities for each target link given the evidence, then multiplying them to approximate the multi link joint probability. Figure 4.1 shows this process; in symbols:

$$\mathbb{P}(\boldsymbol{A}[u, v_1], \ldots, \boldsymbol{A}[u, v_m] | \boldsymbol{E}) \approx \prod_{i=1}^{m} \mathbb{P}(\boldsymbol{A}[u, v_i] | \boldsymbol{E}). \tag{4.1}$$

where u is a target node and $v_1 \ldots v_m$ are all the possible neighbors of u.

To compute ranking metrics for neighborhood queries, we require a score to each potential neighbor u of a test node v. For the baseline methods, I use the probability of the link $\boldsymbol{A}[u, v]$, computed either by independent prediction. For the VGAE sampling methods, I use the link probability averaged over all samples.

Subgraph Queries Since our baselines can only perform node classification or link prediction, to evaluate the performance of VGAE+ in comparison to them, we have evaluated GAT, GraphSAGE, and GiGaMAE separately on these tasks. The evaluation metrics for these tasks include the AUC and AP.

The AUC and AP for subgraph queries are computed as the average of the AUC and AP obtained from node classification and link prediction tasks. This averaging provides a measure of the model's ability to handle the more complex task of subgraph prediction, which inherently involves both predicting links and classifying nodes within the subgraph.

Chapter 5

Results

In this section, I evaluate VGAE+ performance on six benchmark datasets. First, I discuss the results for subgraph queries. Next, I break down the subgraph AUC scores into their link prediction and node label components. This allows us to understand the results in more detail and to compare with more single-task baselines.

5.1 Subgraph Queries

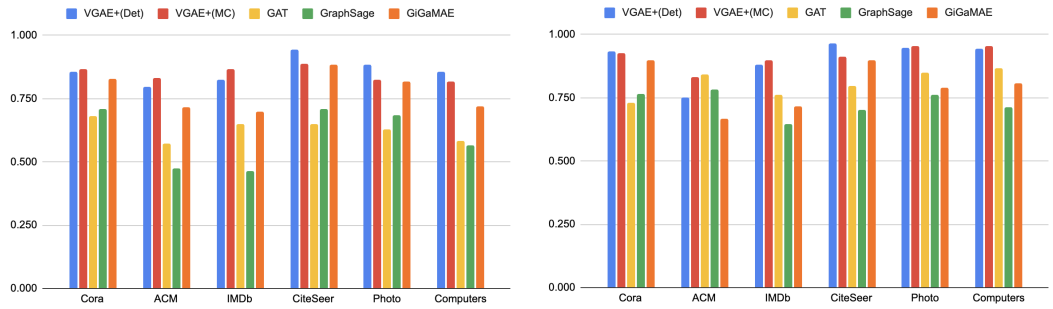
		Single Neighbor Queries						**Neighborhood Queries**					
		Cora	ACM	IMDb	CiteSeer	Photo	Computers	Cora	ACM	IMDb	CiteSeer	Photo	Computers
Semi-inductive	VGAE+(Det)	0.937	0.837	0.863	**0.949**	**0.960**	0.910	0.868	**0.815**	**0.764**	0.935	0.854	0.789
	VGAE+(MC)	**0.949**	0.828	0.900	0.941	0.951	**0.960**	**0.933**	0.716	0.735	**0.951**	**0.950**	**0.919**
	GAT	0.862	0.909	0.843	0.910	0.935	0.922	0.829	0.726	0.733	0.900	0.881	0.840
	GraphSAGE	0.797	0.723	0.748	0.750	0.860	0.858	0.701	0.689	0.677	0.786	0.846	0.724
	GiGaMAE	0.913	**0.886**	**0.920**	0.896	0.956	0.929	0.917	0.782	0.746	0.896	0.900	0.703
Inductive	VGAE+(Det)	**0.932**	0.751	0.880	**0.963**	0.947	0.945	0.858	0.797	0.824	**0.943**	**0.885**	**0.886**
	VGAE+(MC)	0.926	**0.832**	**0.900**	0.912	**0.954**	**0.955**	**0.866**	**0.831**	**0.866**	0.889	0.825	0.819
	GAT	0.728	0.841	0.760	0.798	0.849	0.866	0.681	0.573	0.650	0.648	0.630	0.582
	GraphSAGE	0.766	0.783	0.644	0.702	0.761	0.712	0.710	0.475	0.463	0.709	0.684	0.566
	GiGaMAE	0.896	0.668	0.715	0.896	0.788	0.805	0.828	0.716	0.700	0.885	0.816	0.718

Table 5.1: ROC-AUC results for subgraph queries in semi-inductive and inductive settings.

		Single Neighbor Queries						**Neighborhood Queries**					
		Cora	ACM	IMDb	CiteSeer	Photo	Computers	Cora	ACM	IMDb	CiteSeer	Photo	Computers
Semi-inductive	VGAE+(Det)	0.844	**0.836**	**0.830**	**0.889**	0.908	0.884	0.878	**0.826**	**0.829**	**0.902**	0.862	**0.869**
	VGAE+(MC)	0.787	0.796	0.745	0.858	0.918	**0.901**	0.709	0.727	0.786	0.855	**0.903**	0.855
	GAT	0.751	0.734	0.747	0.760	0.783	0.621	0.743	0.681	0.659	0.724	0.699	0.606
	GraphSAGE	0.592	0.755	0.645	0.820	0.688	0.650	0.515	0.528	0.473	0.711	0.492	0.572
	GiGaMAE	**0.902**	0.739	0.790	0.885	**0.938**	0.843	**0.916**	0.669	0.740	**0.902**	0.879	0.750
Inductive	VGAE+(Det)	0.829	0.816	**0.870**	**0.920**	0.919	0.945	**0.845**	0.776	**0.846**	**0.944**	0.901	0.886
	VGAE+(MC)	0.887	**0.859**	0.821	0.882	**0.929**	**0.958**	0.808	**0.857**	0.801	0.841	**0.929**	**0.950**
	GAT	0.610	0.648	0.668	0.708	0.740	0.586	0.554	0.586	0.631	0.600	0.591	0.500
	GraphSAGE	0.578	0.522	0.466	0.657	0.558	0.565	0.566	0.574	0.561	0.723	0.576	0.593
	GiGaMAE	**0.896**	0.710	0.731	0.907	0.915	0.833	0.820	0.612	0.747	0.937	0.854	0.741

Table 5.2: AP results for subgraph queries in semi-inductive and inductive settings.

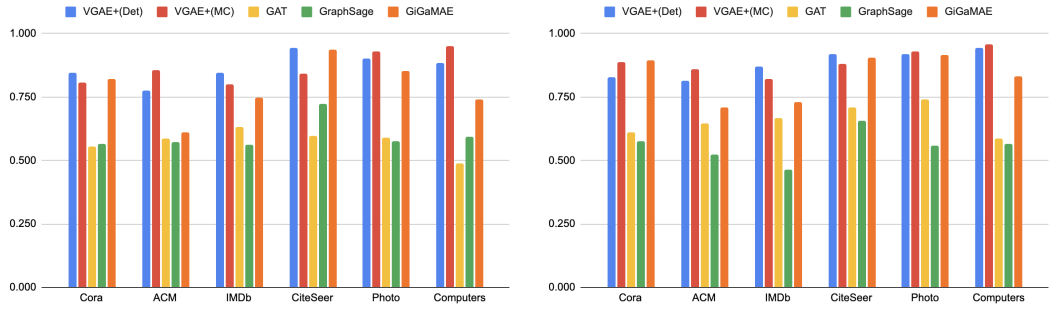
Tables 5.1 and 5.2 and Figures 5.1 and 5.2 show the ROC-AUC and AP results for subgraph queries, which predict *both* links and node labels. In the inductive setting, target nodes, and potentially some of their neighbors, are unseen during training.



(a) ROC-AUC results for Inductive Neighborhood queries.

(b) ROC-AUC results for Inductive Single Neighbor queries.

Figure 5.1: Bar chart for ROC-AUC results for Inductive Neighborhood and Inductive Single Neighbor queries



(a) AP results for Inductive Neighborhood queries.

(b) AP results for Inductive Single Neighbor queries.

Figure 5.2: Bar chart for AP results for Inductive Neighborhood and Inductive Single Neighbor queries

5.1.1 Single Neighbor Queries

The task is to predict the link and node label for a single neighbor of a target node (See Figure **??**).

ROC-AUC

VGAE+ scores higher than all three baselines **in the inductive setting**. The improvement over the next best baseline is substantive on 6 out of 6 datasets; for instance, on IMDb, the improvement is almost 0.2 AUC points.

GiGaMAE was designed for the **transductive setting** and is generally competitive with VGAE+(MC) and VGAE+(Det) on single neighbor semi-inductive queries. VGAE+ scores substantially higher than GraphSAGE and GAT on all datasets and is usually better than GiGaMAE and performs better on 4 out of 6 datasets; for instance, on CiteSeer by almost 0.05 AUC points. For other 2 datasets, ACM and IMDb it is also very competitive.

AP

For this metric, VGAE+(MC) scores higher than all three baselines **in the inductive setting**. The improvement over the next best baseline is substantive on 5 out of 6 datasets; for instance on ACM, the improvement is almost 0.15 AP points.

For the **semi-inductive setting**, VGAE+(MC) scores substantially higher than GraphSAGE and GAT, and is competitive with GiGaMAE; for instance on ACM by almost 0.1 AP points.

5.1.2 Neighborhood Queries

In these queries the task is to predict the links that connect a target node to its neighbors and the neighbors' labels. (See Figure **??**.)

ROC-AUC

Considering the **inductive** setting, where some target nodes are unseen during training, *both VGAE+ methods score higher than all three two-task baselines on all datasets.* The biggest improvement for VGAE+(MC) is observed on the Computers dataset (0.28 AUC points). The strong performance on inductive queries illustrates how a generative model can generalize to data that may be incomplete or new, by inferring unseen information.

In the **semi-inductive** setting, where all target nodes are observed during training, VGAE+ achieves the highest score on all datasets. The VGAE+(MC) top scores are substantially higher than the baselines; for instance, on Computers it outperforms by more than 0.2 AUC points.

AP

For the **inductive** setting, *both VGAE+ methods again score higher than all three two-task baselines on all datasets.* The biggest improvement for VGAE+(MC) is observed on the Computers dataset (0.21 AP points).

In the **semi-inductive** setting, VGAE+ achieves the highest score on 5 out of 6 datasets. The VGAE+(Det) top scores are higher than the baselines; for instance, on ACM it outperforms by more than 0.16 AP points.

Our results show that for *some datasets, inference through sampling from an approximate posterior, Monte Carlo sampling, yields substantive accuracy improvements.* However, sampling takes longer to produce query results. To give a sense of the difference, for the biggest *Computers* dataset, sampling 30 node embeddings vs. using a single deterministic one takes about 4 times as long (39.02s/query for VGAE+(MC) vs. 11.951s/query for VGAE+(Det)), using a single NVIDIA A40 GPU.

In conclusion, *inference from a VGAE model offers the best performance for predicting jointly both links and node labels.*

5.2 Link Prediction Queries

Tables 5.3, 5.4, and 5.5 show the AUC, AP, and Hit-Rate@20% scores for link prediction queries.

		Single Link Queries						**Joint Link Queries**					
		Cora	ACM	IMDb	CiteSeer	Photo	Computers	Cora	ACM	IMDb	CiteSeer	Photo	Computers
Semi-inductive	VGAE+(Det)	0.898	0.959	0.937	0.941	0.947	0.871	0.759	0.962	0.894	0.910	0.798	0.753
	VGAE+(MC)	**0.912**	0.954	0.957	**0.952**	**0.956**	**0.935**	0.886	0.960	0.901	**0.941**	**0.940**	**0.928**
	SEAL	0.910	0.904	0.906	0.948	0.935	0.908	0.741	0.500	0.620	0.752	0.684	0.701
	GAT	0.862	0.943	0.874	0.913	0.902	0.892	0.764	0.925	0.845	0.904	0.872	0.814
	GraphSAGE	0.854	0.969	0.917	0.589	0.858	0.913	0.725	0.962	0.893	0.666	0.853	0.823
	GiGaMAE	0.907	0.943	0.950	0.950	0.949	0.917	**0.902**	0.959	**0.963**	0.940	0.930	0.906
	DEAL	0.800	**0.986**	**0.981**	0.914	0.832	0.823	0.676	**0.979**	0.962	0.910	0.857	0.826
Inductive	VGAE+(Det)	**0.891**	0.963	0.939	**0.950**	**0.936**	0.925	0.754	0.954	0.867	**0.925**	**0.857**	**0.821**
	VGAE+(MC)	0.865	**0.970**	0.942	0.900	0.931	0.922	**0.778**	**0.968**	**0.935**	0.856	0.760	0.752
	SEAL	0.756	0.666	0.854	0.627	0.924	**0.972**	0.693	0.681	0.830	0.679	0.506	0.450
	GAT	0.667	0.870	0.796	0.731	0.796	0.835	0.491	0.699	0.632	0.470	0.453	0.380
	GraphSAGE	0.561	0.593	0.492	0.504	0.552	0.444	0.543	0.535	0.467	0.512	0.528	0.507
	GiGaMAE	0.887	0.732	0.929	0.943	0.798	0.764	0.725	0.920	0.918	0.921	0.854	0.706
	DEAL	0.780	0.902	**0.957**	0.861	0.852	0.844	0.678	0.953	**0.935**	0.837	0.759	0.757

Table 5.3: ROC-AUC results for link prediction in semi-inductive and inductive settings.

		Single Link Queries						**Joint Link Queries**					
		Cora	ACM	IMDb	CiteSeer	Photo	Computers	Cora	ACM	IMDb	CiteSeer	Photo	Computers
Semi-inductive	VGAE+(Det)	87.5	**100**	**100**	95.9	97.8	92.5	83.9	84.2	84.7	91.6	84.6	85.3
	VGAE+(MC)	**100**	**100**	98.2	**100**	**100**	**100**	**100**	**100**	98.5	97.3	**100**	**100**
	GAT	92.5	**100**	**100**	95.0	94.2	91.5	76.9	82.8	78.9	92.0	77.0	72.8
	GraphSAGE	90.0	**100**	**100**	92.5	97.5	92.5	73.8	98.2	90.0	94.7	83.1	85.5
	GiGaMAE	96.5	71.8	78.9	97.5	85.0	84.7	94.6	69.9	72.5	95.8	76.7	84.1
	DEAL	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
Inductive	VGAE+(Det)	87.5	**100**	95.0	95.3	92.6	88.2	63.9	82.7	80.4	90.5	86.7	85.3
	VGAE+(MC)	**100**	**100**	**100**	**100**	**100**	98.5	**100**	**100**	**100**	98.6	**100**	**100**
	SEAL	97.9	84.3	90.2	97.4	98.1	99.5	52.0	11.5	38.0	48.0	4.00	**100**
	GAT	89.5	**100**	98.8	92.1	96.4	90.0	56.8	72.6	64.5	81.0	68.5	65.8
	GraphSAGE	87.8	**100**	**100**	94.7	93.7	96.1	68.8	94.5	87.8	90.5	85.1	78.5
	GiGaMAE	95.0	65.3	54.2	97.5	85.0	85.0	94.2	54.1	30.6	96.8	70.6	87.1
	DEAL	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**

Table 5.4: Hit-Rate @20% results for link prediction in semi-inductive and inductive settings.

5.2.1 Single Link Queries

In these queries, the task is to predict one positive link and one negative link. (See Figure **??**.)

ROC-AUC

For **single link inductive queries**, VGAE+ outperforms the strongest baselines GiGaMAE and DEAL on 4 out of 6 datasets (Cora, ACM, Photo, and Computers) by 0.11 AUC points. On the IMDb dataset, the VGAE+(MC) score is competitive. VGAE+ inference beats the other single link prediction baselines. For **single link semi-inductive queries**, the VGAE+(MC) scores are the highest on 4 out of 6 datasets, but similar in magnitude to the GiGaMAE and DEAL baselines. A strong baseline performance is expected because semi-inductive single link prediction is the most extensively researched link prediction task.

		Single Link Queries						Joint Link Queries					
		Cora	ACM	IMDb	CiteSeer	Photo	Computers	Cora	ACM	IMDb	CiteSeer	Photo	Computers
Semi-inductive	VGAE+(Det)	0.880	0.965	0.976	0.931	0.940	0.956	0.872	0.934	0.859	0.939	0.940	0.935
	VGAE+(MC)	0.862	0.942	0.876	0.912	0.944	0.962	0.765	0.940	0.886	0.920	0.960	0.920
	SEAL	0.925	0.907	0.934	0.926	0.914	0.908	0.655	0.322	0.641	0.738	0.790	0.720
	GAT	0.874	0.878	0.889	0.834	0.846	0.796	0.806	0.859	0.832	0.786	0.754	0.712
	GraphSAGE	0.738	0.895	0.805	0.860	0.900	0.820	0.500	0.535	0.513	0.631	0.500	0.678
	GiGaMAE	**0.948**	0.863	0.893	**0.985**	**0.992**	**0.987**	**0.984**	0.789	0.867	**0.987**	**0.992**	**0.998**
	DEAL	0.824	**0.987**	**0.984**	0.928	0.823	0.818	0.647	**0.985**	**0.969**	0.927	0.894	0.862
Inductive	VGAE+(Det)	0.796	0.835	**0.988**	0.948	0.944	0.969	0.756	0.796	0.894	0.969	**0.978**	0.952
	VGAE+(MC)	0.892	0.955	0.887	0.898	0.972	0.970	0.832	**0.968**	0.856	0.907	0.965	**0.987**
	SEAL	0.787	0.701	0.647	0.697	0.897	0.959	0.748	0.545	0.577	0.544	0.741	0.775
	GAT	0.534	0.781	0.667	0.695	0.799	0.757	0.628	0.737	0.743	0.655	0.583	0.534
	GraphSAGE	0.486	0.554	0.471	0.532	0.627	0.549	0.723	0.678	0.659	0.707	0.622	0.616
	GiGaMAE	**0.981**	0.824	0.856	**0.998**	**0.979**	**0.985**	**0.879**	0.723	0.843	**0.984**	0.957	0.982
	DEAL	0.780	**0.967**	0.957	0.861	0.852	0.844	0.780	0.967	**0.957**	0.861	0.852	0.844

Table 5.5: AP results for link prediction in semi-inductive and inductive settings.

HR@20%

For the HR@20% metric, on **single link inductive queries**, VGAE+(MC) outperforms or is equal to the strong baseline DEAL on 5 out of 6 datasets. On the Computers dataset, the VGAE+(MC) score is competitive. VGAE+ inference beats the other single link prediction baselines. For **single link semi-inductive queries**, VGAE+(MC) outperforms or is equal to the strong baseline DEAL on 5 out of 6 datasets. On the IMDb dataset, the VGAE+(MC) score is competitive.

AP

For the AP metric, although VGAE+ does not usually outperform the strong baselines GiGaMAE and DEAL, it is very competitive. Also, as I discussed in Section 5.1 in the joint node classification and link prediction setting it performs better than GiGaMAE. In **single link inductive queries**, VGAE+(Det) outperform on IMDb datasets by 0.03 AP points. On Cora, ACM, Photo, and Computers datasets, VGAE+(MC) has the second best results. For **single link semi-inductive queries**, VGAE+(Det) achieves the second best results on 3 out of 6 datasets (ACM, IMDb, and CiteSeer), and the reported AP results for both VGAE+(MC) and VGAE+(Det) are competitive with DEAL and GiGaMAE.

5.2.2 Joint Link Queries

In these queries, the task is to predict m positive links and m randomly chosen negative links that are connected to the target node. (See Figure **??**.)

ROC-AUC

For **inductive joint link prediction**, VGAE+ achieves top scores on all datasets compared to the baselines. VGAE+(MC) performs the best on Cora, ACM, and IMDb, and VGAE+(Det) outperforms on CiteSeer, Photo, and Computers. For **semi-inductive joint link prediction**, VGAE+(MC) outperforms DEAL and GiGaMAE on 3 out of 6 datasets

by 0.13 AUC points. On Cora and IMDb, GiGaMAE performs slightly better by only 0.06 AUC points on Cora, and DEAL only outperforms on ACM by 0.12 AUC points.

HR@20%

For the HR@20% metric, on **joint link inductive queries**, VGAE+(MC) outperforms or is equal to the strong baseline DEAL on 5 out of 6 datasets. On the CiteSeer dataset, the VGAE+(MC) score is competitive only 1% less than DEAL. VGAE+ inference beats the other single link prediction baselines. For **joint link semi-inductive queries**, VGAE+(MC) outperforms or is equal to the strong baseline DEAL on 4 out of 6 datasets. On the IMDb and CiteSeer datasets, the VGAE+(MC) score is only 2% less than DEAL.

AP

For the AP metric, similar to Single Neighbor Queries, VGAE+ is very competitive with DEAL and GiGaMAE. In **inductive joint link queries**, VGAE+(Det) outperforms on Photo dataset by 0.02 AP points. On ACM, and Computers datasets, VGAE+(MC) has the best results. For Cora, IMDb, and CiteSeer, VGAE+ achieves the second best results compared to all baselines. For **semi-inductive joint link queries**, VGAE+(Det) achieves the second best results on 3 out of 6 datasets (Cora, CiteSeer, and Computer), and VGAE+(Det) achieves the second best results on 3 out of 6 datasets (ACM, IMDb, and Photo). The reported AP results for both VGAE+(MC) and VGAE+(Det) are competitive with DEAL and GiGaMAE on all 6 datasets.

5.3 Node Classification Queries

Tables 5.6, 5.7, and 5.8 show the AUC, AP, and F1-score macro scores for node classification queries.

		Single Node Queries						**Joint Node Queries**					
		Cora	ACM	IMDb	CiteSeer	Photo	Computers	Cora	ACM	IMDb	CiteSeer	Photo	Computers
Semi-Inductive	VGAE+(Det)	0.977	0.715	0.789	**0.957**	**0.973**	0.948	0.977	0.668	0.634	0.959	0.910	0.824
	VGAE+(MC)	**0.986**	0.701	0.842	0.930	0.946	**0.986**	**0.980**	0.471	0.568	**0.960**	**0.959**	**0.909**
	GAT	0.861	0.876	0.813	0.907	0.967	0.951	0.893	0.526	0.621	0.896	0.889	0.866
	GraphSAGE	0.740	0.477	0.580	0.911	0.863	0.804	0.677	0.416	0.460	0.907	0.840	0.626
	GiGaMAE	0.920	**0.823**	**0.890**	0.842	0.963	0.941	0.932	0.604	0.529	0.852	0.871	0.500
	MVGRL	0.888	0.708	0.788	0.807	0.963	0.980	0.853	**0.715**	**0.767**	0.815	0.953	0.892
	DeepWalk	0.868	0.643	0.684	0.847	0.950	0.862	0.726	0.567	0.553	0.731	0.937	0.819
Inductive	VGAE+(Det)	0.974	0.540	0.821	**0.976**	0.958	0.965	**0.961**	0.641	0.780	**0.961**	**0.912**	**0.890**
	VGAE+(MC)	**0.986**	0.693	**0.856**	0.924	**0.976**	**0.987**	0.954	0.695	**0.796**	0.922	0.890	0.886
	GAT	0.790	0.812	0.724	0.865	0.902	0.896	0.870	0.446	0.668	0.826	0.806	0.784
	GraphSAGE	0.970	**0.973**	0.797	0.900	0.970	0.980	0.877	0.416	0.460	0.907	0.840	0.626
	GiGaMAE	0.906	0.604	0.501	0.850	0.778	0.847	0.932	0.513	0.482	0.850	0.778	0.730
	MVGRL	0.650	0.804	0.684	0.550	0.835	0.801	0.614	**0.679**	0.632	0.534	0.892	0.827

Table 5.6: ROC-AUC results for node classification in semi-inductive and inductive settings.

5.3.1 Single Node Queries

In these queries, the task is to determine the class of the target node. (See Figure **??**.)

		Single Node Queries						Joint Node Queries					
		Cora	ACM	IMDb	CiteSeer	Photo	Computers	Cora	ACM	IMDb	CiteSeer	Photo	Computers
Semi-inductive	VGAE+(Det)	0.807	**0.706**	**0.784**	**0.846**	0.875	0.811	**0.883**	**0.718**	**0.798**	**0.858**	0.783	**0.803**
	VGAE+(MC)	0.712	0.650	0.614	0.803	**0.892**	**0.840**	0.652	0.513	0.685	0.790	0.846	0.790
	GAT	0.628	0.590	0.605	0.686	0.719	0.445	0.679	0.502	0.486	0.661	0.643	0.500
	GraphSAGE	0.445	0.615	0.485	0.780	0.475	0.480	0.531	0.519	0.432	0.790	0.483	0.465
	GiGaMAE	**0.855**	0.615	0.687	0.785	0.884	0.699	0.848	0.549	0.613	0.820	0.766	0.502
	MVGRL	0.669	0.501	0.561	0.518	**0.892**	0.749	0.574	0.501	0.532	0.526	**0.849**	0.707
	DeepWalk	0.405	0.364	0.366	0.566	0.470	0.465	0.384	0.305	0.290	0.357	0.489	0.397
Inductive	VGAE+(Det)	0.862	**0.796**	0.752	**0.892**	**0.893**	0.921	**0.933**	**0.756**	**0.798**	**0.918**	0.823	0.819
	VGAE+(MC)	**0.882**	0.763	**0.754**	0.866	0.886	**0.946**	0.783	0.753	0.745	0.774	**0.892**	**0.912**
	GAT	0.685	0.514	0.668	0.720	0.680	0.415	0.479	0.434	0.518	0.542	0.598	0.445
	GraphSAGE	0.669	0.490	0.460	0.782	0.488	0.580	0.408	0.470	0.462	0.739	0.529	0.569
	GiGaMAE	0.810	0.595	0.604	0.815	0.850	0.680	0.759	0.500	0.648	0.889	0.753	0.500
	MVGRL	0.510	0.520	0.457	0.411	0.796	0.650	0.426	0.462	0.417	0.491	0.727	0.609

Table 5.7: AP results for node classification in semi-inductive and inductive settings.

		Single Node Queries						Joint Node Queries					
		Cora	ACM	IMDb	CiteSeer	Photo	Computers	Cora	ACM	IMDb	CiteSeer	Photo	Computers
Semi-inductive	VGAE+(Det)	0.792	0.498	0.487	**0.831**	0.742	0.687	0.872	0.411	0.451	**0.914**	0.698	0.638
	VGAE+(MC)	**0.894**	0.606	0.548	0.791	0.801	0.758	**0.903**	0.450	0.448	0.816	0.758	0.715
	GAT	0.529	0.570	**0.671**	0.709	0.814	0.471	0.644	0.468	0.426	0.658	0.606	0.488
	GraphSAGE	0.551	0.416	0.439	0.779	0.598	0.399	0.486	0.435	0.475	0.777	0.476	0.362
	GiGaMAE	0.856	0.440	0.457	0.798	0.770	0.569	0.784	0.430	0.451	0.748	0.797	0.490
	MVGRL	0.886	**0.803**	0.653	0.710	**0.960**	**0.816**	0.789	**0.708**	0.604	0.717	**0.950**	**0.739**
	DeepWalk	0.700	0.632	0.650	0.700	0.850	0.750	0.856	0.605	**0.787**	0.632	0.851	0.663
Inductive	VGAE+(Det)	0.760	0.411	0.500	0.950	0.785	0.643	0.863	0.594	**0.542**	**0.905**	0.628	0.534
	VGAE+(MC)	**0.855**	0.372	**0.741**	**0.794**	**0.863**	**0.758**	**0.935**	0.442	0.481	0.791	**0.845**	0.674
	GAT	0.628	0.543	0.470	0.665	0.724	0.498	0.540	0.448	0.461	0.435	0.586	0.434
	GraphSAGE	0.486	0.416	0.439	0.777	0.850	0.362	0.351	0.335	0.437	0.771	0.789	0.393
	GiGaMAE	0.847	0.406	0.441	0.786	0.779	0.340	0.852	0.189	0.421	0.732	0.793	0.320
	MVGRL	0.430	**0.786**	0.460	0.416	0.743	0.720	0.501	**0.735**	0.478	0.365	0.801	**0.706**

Table 5.8: F1-score macro results for node classification in semi-inductive and inductive settings.

ROC-AUC

For **inductive single** node classification, VGAE+ achieves the top score on 5 out of 6 datasets. VGAE+(MC) outperforms on 4 datasets (Cora, IMDb, Photo, and Computers) by almost 0.2 AUC points. GraphSAGE was designed for inductive node classification and is accordingly a strong baseline, especially on the ACM dataset.

For **semi-inductive single** node classification, VGAE+ achieves the top score on 4 out of 6 datasets. VGAE+(MC) outperforms on 2 datasets (Cora and Computers) by almost 0.12 AUC points. VGAE+(Det) outperforms on 4 datasets (CiteSeer and Photo) by almost 0.12 AUC points. The biggest improvement is observed on Cora, with 0.06 AUC points over GiGaMAE. As with link prediction, single node classification in the semi-inductive setting is the most extensively researched node classification setting, so I expect the baselines to perform well.

AP

For the AP metric in **inductive single** node classification, VGAE+ achieves the top score on all datasets. VGAE+(MC) outperforms on 3 datasets (Cora, IMDb, and Computers) by almost 0.26 AP points on Computers. VGAE+(Det) outperforms on 3 datasets (ACM, CiteSeer, and Photo) by almost 0.2 AP points on ACM.

For **semi-inductive single** node classification, VGAE+ achieves the top score on 5 out of 6 datasets. VGAE+(MC) outperforms on 2 datasets (Photo and Computers) by almost 0.1 AP points. VGAE+(Det) outperforms on 3 datasets (ACM, IMDb, and CiteSeer) by almost 0.1 AP points on IMDb.

F1-Score macro

For the F1-Score macro metric in **inductive single** node classification, VGAE+(MC) achieves the top score on 5 out of 6 datasets, by almost 0.2 on IMDb.

For **semi-inductive single** node classification, VGAE+ achieves the top score on 2 out of 6 datasets. VGAE+(Det) on Cora by 0.03 points and VGAE+(MC) on CiteSeer by 0.12 points. On the 4 remaining datasets, GAT outperforms on IMDb by 0.13 points, and MVGRL outperforms on Photo and Computers by at most 0.16 points.

5.3.2 Joint Node Queries

In these queries, the task is to determine the class of the neighbors of the target node. (See Figure **??**.)

ROC-AUC

For **inductive joint** node classification, VGAE+(Det) achieves a higher score than the baseline methods on 4 out of 6 datasets, especially on CiteSeer (at least 0.06 AUC points). On ACM, the strong node classification baseline MVGRL achieves a higher score. Similarly, in the **semi-inductive joint** node classification setting, VGAE+ has a higher score than the baselines on 4 out of 6 datasets. The improvement is strongest on CiteSeer (0.06 over GraphSAGE). MVGRL is exceptionally strong in the semi-inductive setting on ACM and IMDb.

AP

For the AP metric in **inductive joint** node classification, VGAE+ achieves the top score on all datasets. VGAE+(Det) outperforms on 4 datasets (Cora, ACM, IMDb, and CiteSeer) by almost 0.26 AP points on ACM. VGAE+(MC) outperforms on 2 datasets (Photo and Computers) by almost 0.31 AP points on Computers.

For **semi-inductive joint** node classification, VGAE+(Det) achieves the top score on 5 out of 6 datasets (all datasets except for Photo) by almost 0.18 AP points on IMDb. MVGRL outperforms VGAE+ on Photo by only 0.003 AP points.

F1-Score macro

For the F1-Score macro metric in **inductive joint** node classification, VGAE+ achieves the top score on 4 out of 6 datasets. VGAE+(MC) outperforms on 2 datasets (Cora and

Photo) by almost 0.08 points on Cora. VGAE+(Det) outperforms on 2 datasets (IMDb and CiteSeer) by almost 0.17 points on CiteSeer. MVGRL outperforms on ACM and Computers by 0.13 points.

For **semi-inductive joint** node classification, VGAE+ achieves the top score on 2 out of 6 datasets (VGAE+(MC) on Cora by 0.05 points and VGAE+(Det) on CiteSeer by 0.13 points), and is competitive on the remaining datasets. DeepWalk outperforms on IMDb by 0.33 points, and MVGRL outperforms on ACM, Photo, and Computers by at most 0.25 points on ACM.

5.4 Subgraph Queries vs. Node Classification and Link Prediction

Reviewing the results of inductive subgraph prediction in terms of link prediction and node classification, I observe that the high score of VGAE+(MC) on IMDb is due to its excellent score on *joint node classification*. On the ACM dataset, VGAE+(MC) achieves strong subgraph prediction through a high *joint link prediction score*. The strong AUC score on Computers is due to both high joint node classification scores and high joint link prediction scores.

Overall, my experiments provide strong evidence that VGAE+ *achieves an excellent balance between predicting links and node labels across different query types*. For single query types (e.g., link prediction), predictive performance is very competitive with custom baselines.

5.5 Ablation Study

5.5.1 Objective Function

Table 5.9 examines the importance of the components of our new training objective, Equation 3.2, for VGAE+. Table 5.9a shows that not reconstructing the node features leads to worse scores, especially on the IMDb and Computers datasets. This is remarkable since the neighborhood queries do not contain node features as a target. Table 5.9b shows the importance of using features and labels in link prediction task. By setting β and γ zeroes, the objective function will be similar to VGAE objective function, and the results shows are for link prediction task. This table shows the effect of reconstructing the node features and node labels on link prediction task. Also Table 5.9c shows that turning off label reconstruction ($\gamma = 0$) and link reconstruction ($\alpha = 0$) leads to worse scores for node classification task. The strong performance of VGAE+ highlights the value of using both node features (node labels and node features) and graph structure (links) as co-training objectives that improve both node classification and link prediction.

	Single Neighbor Queries						Neighborhood Queries					
	Cora	ACM	IMDb	CiteSeer	Photo	Computers	Cora	ACM	IMDb	CiteSeer	Photo	Computers
VGAE+(Det)	**0.932**	0.751	0.880	**0.963**	0.947	0.945	0.858	0.797	0.824	**0.943**	**0.885**	**0.886**
($\beta = 0$)	0.749	**0.782**	0.843	0.879	0.864	0.856	0.719	0.675	0.660	0.715	0.788	0.785

(a) AUC results for β=0

	Single Link Queries						Joint Link Queries					
	Cora	ACM	IMDb	CiteSeer	Photo	Computers	Cora	ACM	IMDb	CiteSeer	Photo	Computers
VGAE+(Det)	**0.891**	**0.963**	**0.939**	**0.950**	0.936	0.925	**0.754**	**0.954**	**0.867**	**0.925**	**0.857**	**0.821**
($\beta = 0, \gamma = 0$) (VGAE)	0.874	0.895	0.876	0.927	**0.970**	**0.943**	0.742	0.752	0.736	0.898	0.846	0.844

(b) AUC results for β=0 and γ=0; this setting corresponds to the traditional VGAE model.

	Single Node Queries						Joint Node Queries					
	Cora	ACM	IMDb	CiteSeer	Photo	Computers	Cora	ACM	IMDb	CiteSeer	Photo	Computers
VGAE+(Det)	**0.974**	0.540	**0.821**	**0.976**	**0.958**	**0.965**	**0.961**	**0.641**	**0.780**	**0.961**	**0.912**	**0.890**
($\beta = 0, \alpha = 0$)	0.798	**0.690**	0.687	0.801	0.780	0.795	0.907	0.654	0.667	0.775	0.683	0.691

(c) AUC results for β=0 and α=0

Table 5.9: Ablation Study on the training objective 3.2.

5.5.2 Iterative Joint Link Prediction

For the joint link prediction setting, all reported results assume that links are i.i.d. (independent and identically distributed), given the node embeddings, so I predict them all at once. Another approach to joint link prediction is to predict links one by one using the chain rule with a random ordering. If we assume that we have m target links in our query, $\boldsymbol{A}[u_1, v_1], ..., \boldsymbol{A}[u_m, v_m] \in \mathcal{A}^{\boldsymbol{Y}}$, then instead of $P(\mathcal{A}^{\boldsymbol{Y}}|\boldsymbol{E})$, the output will be calculated using chain rule:

$$\begin{aligned} P(\mathcal{A}^{\boldsymbol{Y}}|\boldsymbol{E}) &= P(\boldsymbol{A}[u_1, v_1]|\boldsymbol{E}) \times P(\boldsymbol{A}[u_2, v_2]|\boldsymbol{E}, \boldsymbol{A}[u_1, v_1]) \times ... \times \\ & P(\boldsymbol{A}[u_m, v_m]|\boldsymbol{E}, \boldsymbol{A}[u_1, v_1], ..., \boldsymbol{A}[u_{m-1}, v_{m-1}]) \end{aligned} \tag{5.1}$$

Table 5.10 shows the results for iterative inductive link prediction. As we can see, using the chain rule and predicting links iteratively usually gives better results (for 5 out of 6 datasets with AUC improvement in the range of 0.01 to 0.06). Although there is an improvement with this method, it takes a lot of inference time and memory. While I used 100 test nodes for evaluating our model previously, with my available computational resources, it was only possible to use 10 test nodes with iterative joint link prediction. To give a sense of the difference, for the densest Computers dataset, sampling 10 node embeddings takes about 30 times longer than the proposed approach in the thesis (316.03s/query for VGAE+(iterative) vs. 9.09s/query for VGAE+). Even for ACM, which is a sparse dataset, sampling 10 node embeddings takes about 5 times longer than the proposed approach in the thesis (16.23s/query for VGAE+(iterative) vs. 3.02s/query for VGAE+). The time overhead varies based on the average number of target links for the evaluation dataset. For example,

	Joint Link Queries					
	Cora	ACM	IMDb	CiteSeer	Photo	Computers
VGAE+	**0.761**	0.916	0.919	0.937	0.930	0.940
VGAE+(iterative)	0.723	**0.979**	**0.937**	**0.998**	**0.965**	**0.956**

Table 5.10: ROC-AUC results for iterative joint link prediction

the average number can be as small as 6 for ACM, which is a sparse graph, and can be as large as 30 for Computers, which is a dense graph.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

A subgraph prediction query asks for the probability of a target subgraph, which is a set of nodes and links in the graph, given the information from an evidence subgraph. Supporting inference to answer subgraph queries (SQs) is a new use case for deep Graph Generative Models (GGMs). Such a query answering system facilitates applying graph prediction in a production environment where multiple users pose a range of queries to be answered. In this research, I showed how VGAE+ model can be used to answer SQs in zero-shot manner, without retraining the model. Bayesian optimization was effective in balancing the relative importance of modeling links, node features, and node labels in a dataset-dependent manner. I conducted an empirical evaluation on six benchmark datasets and a range of test queries. The application of joint prediction from a single VGAE yielded higher accuracy than baseline methods that predict graph components independently.

6.2 Limitation and Future Works

Limitations. A limitation of this evaluation is that I considered only homogeneous graphs with a single link type. Deterministic and MC inference can be straightforwardly extended to knowledge graphs using a relational VGAE model.

Future Work. While the VGAE is a well-established GGM for a single training graph, other GGMs, especially auto-regressive and diffusion models, are known to have greater modeling power to capture complex correlation patterns in graphs [23]. Leveraging the greater expressive power of these GGMs to improve subgraph predictions over VGAE+ is a fruitful direction for future research, especially if they can be trained on single graph inputs.

A valuable direction for future research is to enhance the model's ability to differentiate between links explicitly specified as absent in the query and unspecified links. In the current study, the model treats absent links as non-existent. This is a simplification that facilitates leveraging graph neural networks, but potentially leads to lower accuracy for SQs involving

absent links. A promising approach would involve representing unspecified links as missing values in the adjacency matrix, thereby distinguishing them from absent links.

Another promising direction for future work is to design the model to use only a specific subgraph around the target subgraph instead of the entire input graph as evidence. While this approach might miss some information, it would be computationally efficient. Additionally, since most GNNs usually use at most 5-hop neighbors for nodes to generate embeddings, selecting an appropriate subgraph could still yield good embeddings.

Finally, another valuable direction related to subgraph prediction is to apply inference from a model to find the *most likely subgraph* given the evidence.

Bibliography

[1] Emily Alsentzer, Samuel Finlayson, Michelle Li, and Marinka Zitnik. Subgraph neural networks. *NeurIPS*, 2020.

[2] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE transactions on knowledge and data engineering*, 2018.

[3] Pedro Domingos and Daniel Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan and Claypool Publishers, 2009.

[4] Faezeh Faez, Yassaman Ommi, Mahdieh Soleymani Baghshah, and Hamid R Rabiee. Deep graph generators: A survey. *IEEE Access*, 2021.

[5] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. *ACM*, 2018.

[6] Michael Galkin, Zhaocheng Zhu, Hongyu Ren, and Jian Tang. Inductive logical query answering in knowledge graphs. *NeurIPS*, 2022.

[7] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *ACM*, 2016.

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *NeurIPS*, 2017.

[9] William L Hamilton. *Graph representation learning*. Morgan & Claypool Publishers, 2020.

[10] Yu Hao, Xin Cao, Yixiang Fang, Xike Xie, and Sibo Wang. Inductive link prediction for nodes having only attribute information. *IJCAI*, 2020.

[11] Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. *ICML*, 2020.

[12] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. *ACM*, 2020.

[13] Bert Huang, Angelika Kimmig, Lise Getoor, and Jennifer Golbeck. Probabilistic soft logic for trust analysis in social networks. *International Workshop on StarAI*, 2012.

[14] Przemyslaw Kazienko and Tomasz Kajdanowicz. Label-dependent node classification in the network. *Neurocomputing*, 2012.

[15] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. *CoRR*, 2013.

[16] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *NeurIPS*, 2016.

[17] Zhida Li, Ana Laura Gonzalez Rios, and Ljiljana Trajković. Machine learning for detecting anomalies and intrusions in communication networks. *IEEE Journal on Selected Areas in Communications*, 2021.

[18] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Charlie Nash, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. *NeurIPS*, 2019.

[19] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. *ACM*, 2003.

[20] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. *ACM*, 2015.

[21] Nikhil Mehta, Lawrence Carin Duke, and Piyush Rai. Stochastic blockmodels meet graph neural networks. *ICML*, 2019.

[22] Aditya Krishna Menon and Charles Elkan. Link prediction via matrix factorization. *ECML*, 2011.

[23] Parmis Naddaf, Erfaneh Mahmoudzaheh Ahmadi Nejad, Kiarash Zahirnia, Manfred Jaeger, and Oliver Schulte. Joint link prediction via inference from a model. *CIKM*, 2023.

[24] Fernando Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python, 2014.

[25] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. *ACM*, 2014.

[26] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, 2008.

[27] Ryan A Rossi, Rong Zhou, and Nesreen K Ahmed. Deep inductive graph representation learning. *IEEE*, 2018.

[28] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *The AI Magazine*, 2008.

[29] Yucheng Shi, Yushun Dong, Qiaoyu Tan, Jundong Li, and Ninghao Liu. Gigamae: Generalizable graph masked autoencoder via collaborative latent space reconstruction. *ACM*, 2023.

[30] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. *ICANN*, 2018.

[31] Jianheng Tang, Fengrui Hua, Ziqi Gao, Peilin Zhao, and Jia Li. Gadbench: Revisiting and benchmarking supervised graph anomaly detection. *NeurIPS*, 2024.

[32] Komal Teru, Etienne Denis, and Will Hamilton. Inductive relation prediction by subgraph reasoning. *ICML*, 2020.

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

[34] João Vitorino, Isabel Praça, and Eva Maia. Sok: Realistic adversarial attacks and defenses for intelligent network intrusion detection. *Computers & Security*, 2023.

[35] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. Graphgan: Graph representation learning with generative adversarial nets. *AAAI*, 2018.

[36] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

[37] Zongqian Wu, Mengmeng Zhan, Haiqi Zhang, Qimin Luo, and Kun Tang. Mtgcn: A multi-task approach for node classification and link prediction in graph data. *Information Processing & Management*, 2022.

[38] Shunxin Xiao, Shiping Wang, Yuanfei Dai, and Wenzhong Guo. Graph neural networks in node classification: survey and evaluation. *Machine Vision and Applications*, 2022.

[39] Tian Xie, Bin Wang, and C-C Jay Kuo. Graphhop: An enhanced label propagation method for node classification. *IEEE*, 2022.

[40] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *ICLR*, 2019.

[41] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *NeurIPS*, 2019.

[42] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. GraphSAINT: Graph sampling based inductive learning method. *ICLR*, 2020.

[43] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *NeurIPS*, 2018.

[44] Muhan Zhang and Yixin Chen. Inductive matrix completion based on graph neural networks. *ICLR*, 2020.

[45] Zijia Zhang, Yaoming Cai, and Wenyin Gong. Semi-supervised learning with graph convolutional extreme learning machines. *Expert Systems with Applications*, 2023.

[46] Xiaojin Zhu Zhuxj, Zoubin Ghahramani, and John Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. *ICML*, 2003.

Appendix A

Additional Information and Examples

A.1 Code

You can access the code for VGAE+ in the following GitHub repository.

A.2 Examples For All Queries

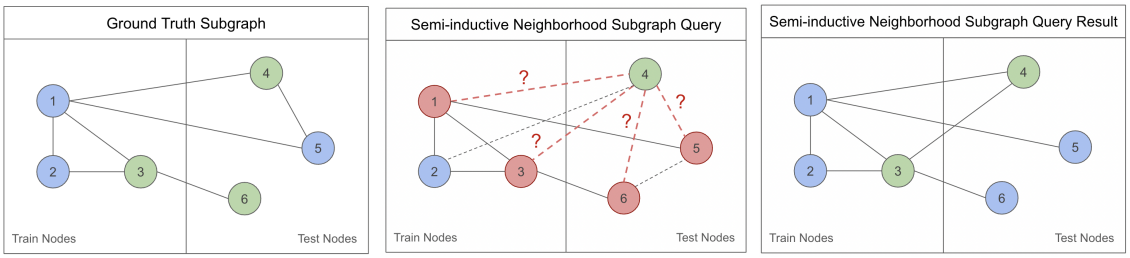
In this section, I give some examples of all the query types that have been used in this research.

A.2.1 Inductive Query Examples

In the inductive setting, the target node u should belong to V_{te}. For all the links in $\mathcal{A}^{\boldsymbol{Y}}$, at least one node for each link belongs to V_{te}. Also, at least one node in $\mathcal{L}^{\boldsymbol{Y}}$ belongs to V_{te}.

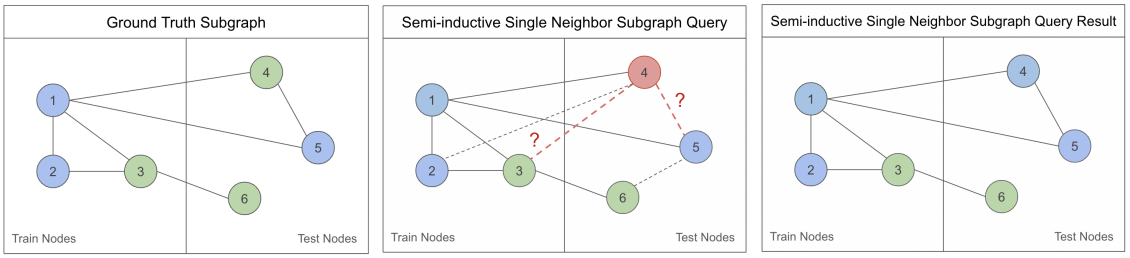
Subgraph Queries

All the reported results for VGAE+ are in this setting (Chapter 5). VGAE+ answers link prediction tasks and node classification tasks at the same time. The objective function (Equation 3.2) trains the model to answer link prediction and node classification at the same time. Section 5.1 shows these joint results. To our knowledge, there is no other model that answers subgraph queries. Other baselines that I have used to evaluate my model in Section 5.1 (GAT, GraphSAGE, and GiGaMAE) generate embeddings for one task, either link prediction or node classification, and are trained separately for each task with different objective functions. Then, the average of link prediction and node classification ROC-AUC and AP results are reported.



(a) Input graph (b) Neighborhood query. (c) The output of the model.

An example of Inductive Neighborhood Query

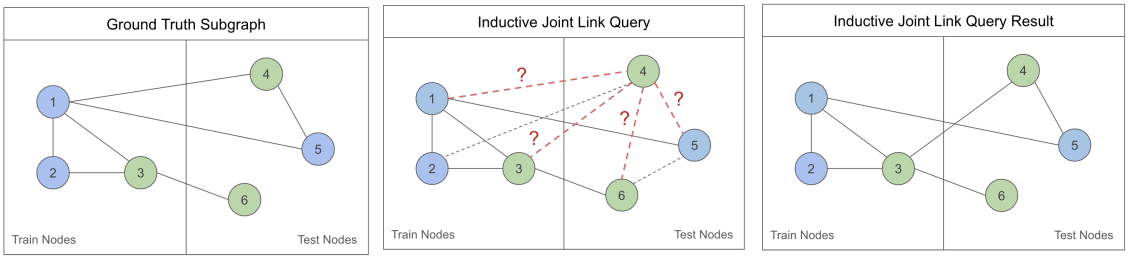


Input graph Single Neighbor Query The output of the model

An example of Inductive Single Neighbor Query.

Link Queries

Figure **??** shows example of inductive joint link prediction queries, and Figure **??** show example of inductive single link prediction query that we have used for our link prediction baselines (SEAL, GAT, GraphSAGE, GiGaMAE, and DEAL) in Section 5.2.

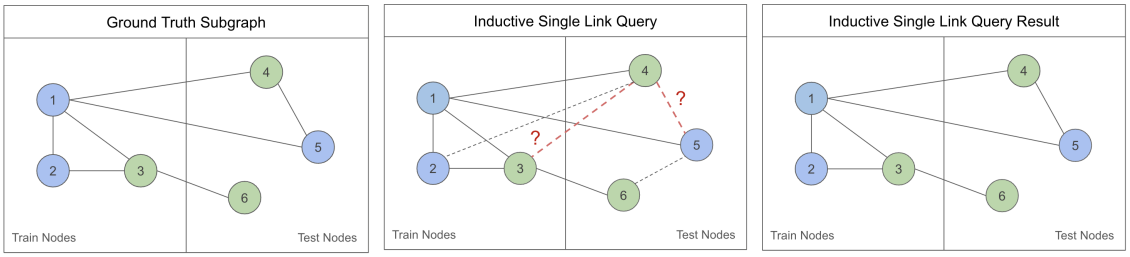


Input graph Joint Link Query. The output of the model.

An example of Inductive Joint Link Query

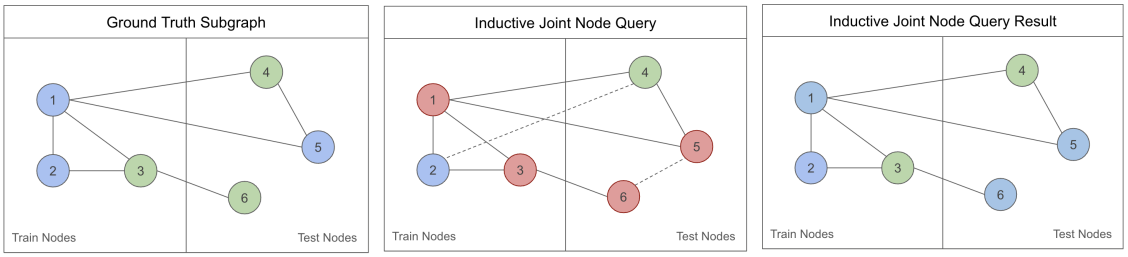
Node Queries

Figure **??** show examples of inductive joint node classification queries and Figure **??** show examples of inductive single node classification queries that we have used for our node classification baselines (GAT, GraphSAGE, GiGaMAE, DeepWalk and MVGRL) in Section 5.3.



(a) Input graph Single Link Query The output of the model

An example of Inductive Single Link Query.

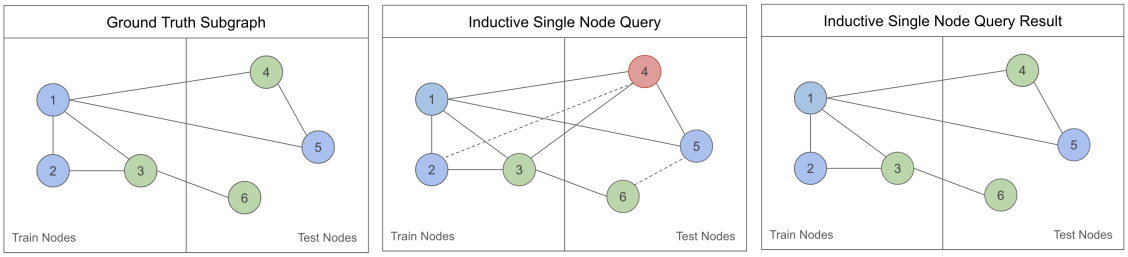


Input graph Joint Node Query The output of the model

An example of Inductive Joint Node Query.

A.2.2 Inductive Query Examples

For inductive queries the only difference is that all the target nodes in $\mathcal{L}^{\boldsymbol{Y}}$ and $\mathcal{A}^{\boldsymbol{Y}}$ should belong to V_{te}. For semi-inductive setting all the target nodes in $\mathcal{L}^{\boldsymbol{Y}}$ and $\mathcal{A}^{\boldsymbol{Y}}$ can belong to V_{te} or V_{tr}.



(a) Input graph (b) Single Node Query (c) The output of the model

An example of Inductive Single Node Query.