
# VAGUE

## Variational Autoencoder for Graph Understanding and Expansion

## Master Thesis

Merlin Streilein

Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

PD Dr. Kaspar Riesen

June 2025



# Abstract

This thesis presents VAGUE (Variational Autoencoder for Graph Understanding and Expansion), a framework for synthetic graph data generation using Variational Autoencoders (VAEs). The goal is to alleviate the scarcity of large-scale, high-quality graph datasets, particularly in fields such as bioinformatics and chemistry.

VAGUE introduces a pipeline consisting of three main parts. A VAE for graph understanding and embedding. A latent space interpolation approach to generate new graph samples, which are then combined with existing data to augment training sets. And a downstream evaluation suite to probe the effectiveness of these extended datasets.

The evaluation of the extended datasets demonstrates substantial improvements in performance, even when only a small amount of synthetic data is added. This indicates that interpolating in the latent space of graph VAEs can produce realistic and useful samples that enhance model performance. The framework is designed to be general and model-agnostic while incorporating node attributes.

The VAGUE pipeline provides a practical and extensible approach to graph dataset augmentation and contributes to addressing the growing demand for rich, diverse graph data in machine learning applications.

# Contents

# 1 Introduction

Machine learning and pattern recognition are fundamental tools for analyzing and understanding complex data. Within this area, deep learning, a subfield of machine learning, has led to major breakthroughs in domains such as natural language processing, computer vision, and bioinformatics. Similarly, pattern recognition, including its graph-based subfield, aims to partially emulate human perception and cognition while addressing the challenge of representing data in a structured and meaningful way. Combining these approaches provides a powerful foundation for advancing the frontiers of modern computer science.

Traditional machine learning typically relies on data representations in Euclidean spaces, which are well-suited for many applications. However, in domains such as social networks, molecular biology, or transportation systems, data is more naturally represented as graphs. This has motivated a growing interest in graph-based machine learning, which directly operates on graph structures, capturing both the attributes of individual components and the relationships between them.

Graphs offer a excellent way to represent structured, relational data. Formally, a graph consists of a finite set of nodes and edges, with optional node and edge attributes. Nodes represent entities, while edges encode interactions or relationships between pairs of nodes. Graph-based representations introduce an inductive bias that enables models to generalize better in scenarios where the data's structure is essential. They also allow for variable-sized, non-Euclidean input and support learning from both node features and graph topology.

Despite their expressive power, graph representations come with challenges. Operations such as graph matching [1] are computationally expensive, with the general graph edit distance [2] problem known to be NP-complete. Many graph kernels [3] also suffer from high time complexity. Additionally, graph datasets, particularly in specialized domains such as bioinformatics or chemistry, are often small. This data scarcity can hinder the performance and generalization of deep learning models trained on graphs.

Data augmentation is a well-established strategy for increasing dataset size and improving model generalization, especially in image and text domains. In image processing, for example, augmentation may involve using cropped or mirrored versions of images during training. However, augmenting graph data is more difficult due to the need to preserve semantic and structural properties. Nevertheless, graph augmentation has been shown to improve the robustness and performance of graph-based models [4]. The increasing interest in generative models such as Variational Autoencoders (VAEs) [5] and Generative Adversarial Networks (GANs) [6] presents new opportunities to enrich graph datasets for downstream tasks such as classification, link prediction, or regression.

Using VAEs for data generation is a well-explored approach [7], particularly because they provide a principled framework for learning latent distributions and generating new samples. A VAE typically consists of an encoder and decoder network that work together to map input data into a latent representation and reconstruct it. When the dimensionality of the latent space is smaller than that of the original input, the model is forced to retain only the most informative features. The regularized latent space of VAEs allows for smooth interpolation, meaningful sampling, and the generation of new data from randomly sampled latent vectors.

Previous work on data augmentation includes techniques in image [8] and text domains [9] as well as more recent methods involving Large Language Models (LLMs) such as Grok [10], DeepSeek [11], and GPT [12] for synthetic data generation [13]. In the graph domain, various augmentation strategies have been explored [14], including specialized graph VAE architectures [15] and general-purpose VAEs used for graph generation [16, 17].

In this thesis, we build upon the idea of using VAEs to augment graph datasets and evaluate the impact on downstream tasks. Our contributions are threefold:

1. We implement a VAE with an auxiliary node label prediction head to facilitate latent space learning on graphs.
2. We utilize a spherical linear interpolation sampling strategy for generating synthetic graphs from the learned latent space.
3. We evaluate the effectiveness of the synthetic data by training a downstream classifier on an extended dataset consisting of varying proportions of original and synthetic graphs.

These contributions are mirrored in the three parts of the Variational Autoencoder for Graph Understanding and Expansion (VAGUE) pipeline, an overview over which is visualized in Figure 1.1:

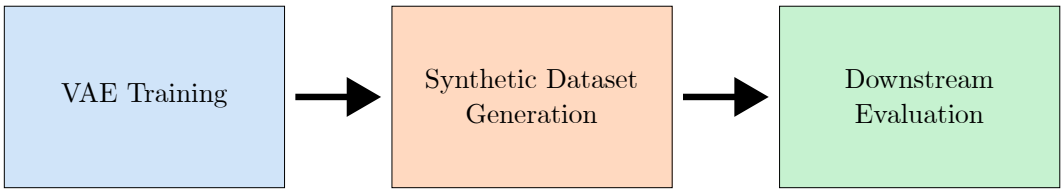


Figure 1.1: Brief Overview over the VAGUE Pipeline

This thesis is structured as follows:

Chapter 2 provides the necessary background on deep learning, graph-based pattern recognition, and autoencoder architectures, with a particular focus on VAEs and their extensions to graph data. It also surveys related work on data augmentation and generative modeling in the graph domain.

Chapter 3 introduces our proposed method, VAGUE, which leverages a VAE framework for graph data augmentation. This chapter details the model architecture, including the encoder, decoder, and classification head, and describes the sampling approach used to generate synthetic graphs. It also outlines the evaluation setup using downstream tasks.

Chapter 4 presents our experimental setup and results. We evaluate the effectiveness of our augmentation strategy both qualitatively and quantitatively, and include ablation studies to analyze individual components of the model.

Finally, Chapter 5 concludes the thesis by summarizing key findings and discussing potential directions for future work. Supplementary and extended results can be found in Appendix A and additional implementation details are documented in Appendix B.

# 2

# Background and Prior Work

This chapter provides the theoretical foundation and situates this thesis within the context of existing research. We begin by introducing the background of deep learning in Section 2.1. Section 2.2 explores the domain of graph-based pattern recognition, including common representations and embeddings for graphs, methods for graph classification, and the main challenges encountered in graph-based learning. We then discuss autoencoders and variational autoencoders in Section 2.3, followed by their graph-specific counterpart, the variational graph autoencoder, in Section 2.4. Finally, Section 2.5 reviews related work on graph generation and graph-based data augmentation, highlighting how this thesis builds upon prior approaches.

## 2.1 Deep Learning

Deep learning is a branch of machine learning that utilizes artificial neural networks composed of multiple layers to identify patterns and learn representations directly from data. These representations are then used to make predictions on previously unseen data. Inspired by the structure and behaviour of the human brain, deep learning models are capable of automatically extracting features from raw input, thereby reducing the need for manual feature engineering.

Advancements in deep learning have led to remarkable progress in various domains, including computer vision, natural language processing, and biomedical research. A key factor behind the success of deep learning models is the availability of large-scale datasets, which are essential for comprehensive training and effective generalization [18]. The size and diversity of the training data strongly influence a model's performance, particularly its ability to make accurate predictions in real-world scenarios.

Several prominent examples highlight the importance of data in deep learning:

**Large Language Models**: Models such as Grok [10], DeepSeek [11], and GPT [12], have shown exceptional proficiency in understanding and generating human language. These models are trained on extensive corpora of text data, including books, articles, and online sources such as Wikipedia. The richness and variety of this data enables the models to produce context-aware responses and perform complex language-related tasks, such as the cloze task [19].

**Computer Vision Models**: Image recognition systems used in applications like medical diagnostics [20] and autonomous driving [21] rely heavily on large, often labeled, datasets. Convolutional Neural Networks (CNNs) [22] achieve high accuracy in image classification and object detection when trained on datasets such as ImageNet [23], which contains millions of annotated images. Without access to such comprehensive datasets, these models would be significantly less effective during inference.

**Speech Recognition Systems**: Modern speech-to-text technologies, including those used in virtual assistants, depend on deep learning techniques trained on vast datasets of spoken language [24]. Notable examples of such speech-to-text models are OpenAI's Whisper [25] or Mozilla's DeepSpeech [26]. These datasets must cover a wide range of accents, dialects, and background noise conditions to ensure the robustness and adaptability of the models in diverse environments.

As deep learning continues to evolve, the demand for high-quality, large-scale datasets remains a central challenge in improving model performance [27]. At the same time, ethical considerations, such as data collection practices, bias mitigation, and the protection of user privacy, are becoming increasingly important areas of focus in both academic research and practical applications.

## 2.2 Graph-Based Pattern Recognition

Graph-based pattern recognition is a subfield of machine learning that focuses on leveraging the expressive power and structural flexibility of graphs. By representing entities as nodes and their interactions or relationships as edges, graphs can effectively model complex dependencies across a wide range of domains. This representation enables sophisticated analysis and learning tasks that go beyond traditional vector-based approaches.

Typical tasks in graph-based pattern recognition include node classification [28], link prediction [29], and graph classification [30]. This field has evolved significantly over time. Early research emphasized methods such as graph matching [31] and graph clustering [32]. Later, the introduction of graph kernels [3] allowed for more scalable comparisons between graphs by embedding them into a high-dimensional feature space. More recently, advances in deep learning have led to the development of Graph Neural Networks (GNNs) [33] and graph embedding techniques [34], which have shown strong performance in various pattern recognition tasks.

### 2.2.1 Graph Representations and Embeddings

Graph embedding techniques aim to transform graph-structured data into continuous vector spaces. This transformation allows for the use of conventional machine learning algorithms, which typically require fixed-length input vectors and operate more naturally in Euclidean spaces. By learning embeddings that

preserve the structural and relational information of the original graphs, these techniques enable effective downstream learning.

There are two main approaches to graph embedding: shallow and deep learning-based methods.

**Shallow methods**, such as DeepWalk [35] and Node2Vec [36], use random walks to generate sequences of nodes. These sequences are then processed by techniques like Word2Vec to produce low-dimensional node embeddings. While these methods are computationally efficient and perform well on smaller graphs, they could face limitations when applied to large or highly structured datasets.

**Deep learning methods**, such as Graph Convolutional Networks (GCNs) [37] and more general Graph Neural Networks (GNNs) [38], offer a more powerful approach. These models learn node or graph-level embeddings by recursively aggregating information from neighboring nodes. The hierarchical nature of this message passing enables the models to capture both local and eventually global structural patterns within the graph.

A central challenge in graph embedding lies in preserving the essential structural and semantic properties of the original graph. Ensuring that the learned vector representations reflect the key relational patterns is crucial for the success of downstream tasks such as classification, clustering, and link prediction.

### 2.2.2 Graph Classification

Graph classification is a fundamental task in machine learning on structured data, where the goal is to assign labels to entire graphs based on their structural and attribute information. Applications range from bioinformatics and chemistry to social network analysis and program verification. Several approaches have been developed to tackle this problem, ranging from graph embedding techniques, often in combination with general classifiers, to graph neural networks (e.g., GCN, GIN).

Among these approaches graph kernels have emerged as a powerful and interpretable class of models. Graph kernels define similarity measures between graphs, effectively mapping them into an implicit feature space without the need to explicitly compute feature vectors. This approach allows the use of linear classifiers, such as Support Vector Machines (SVMs), on graph data without relying on deep neural network architectures.

In the following subsections, we present a widely used graph kernel and a supervised learning classifier; their combination serves as one strong example among many options of graph classification without deep learning.

#### Weisfeiler-Lehman Kernel

The Weisfeiler-Lehman (WL) kernel [39] is among the most widely used and powerful graph kernels. It is based on the one-dimensional Weisfeiler-Lehman test for graph isomorphism, which iteratively updates node labels by aggregating the labels of neighboring nodes. This procedure captures increasing structural information about the graph with each iteration.

Formally, given a graph $G = (V, E)$ with initial node labels $X_0(v)$, the node labels at iteration $h$ are updated as:

$$X_h(v) = \text{hash}\left(X_{h-1}(v), \{X_{h-1}(u) \mid u \in \mathcal{N}(v)\}\right)$$

where $\mathcal{N}(v)$ denotes the set of neighbors of node $v$, and the hash function ensures a unique encoding of the combined label information. After $H$ iterations, the graph is represented by a feature vector that counts the occurrences of all node labels (i.e., subtree patterns) observed throughout the iterations:

$$\phi(G) = [\#\text{ occurrences of label } \sigma \text{ in } G \text{ over all } h \in [0, H]]_{\sigma \in \Sigma}$$

where $\Sigma$ is the set of all unique labels encountered. These feature vectors can then be used with kernel-based classifiers such as SVMs.

#### Support Vector Machines

Support Vector Machines (SVMs) [40] are supervised learning models commonly used for binary classification. The objective is to find a hyperplane that best separates two classes by maximizing the margin between them. Given a training dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ with binary labels $y_i \in \{-1, +1\}$, the SVM solves the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w}, b, \boldsymbol{\xi}} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^\top \phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i \end{aligned}$$

Here, $\mathbf{w}$ is the weight vector, $b$ is the bias term, $\xi_i$ are slack variables to allow for misclassifications (soft margin), and $C$ is a regularization parameter controlling the trade-off between margin size and classification error. The mapping $\phi(\cdot)$ projects input data into a higher-dimensional feature space, typically defined through a kernel function $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$.

In the dual formulation, the decision function is expressed as:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right)$$

Only a subset of the training points, known as support vectors, have non-zero Lagrange multipliers $\alpha_i$ and influence the position of the decision boundary.

### 2.2.3 Challenges in Graph-Based Learning

Despite recent advances, graph-based learning presents several challenges that impact performance, generalization, and scalability.

A primary concern is computational complexity. Many graph operations, such as message passing in GNNs or spectral transformations, scale poorly with increasing graph size. Operations involving adjacency matrix manipulation or eigenvalue decomposition often become infeasible for large-scale graphs, necessitating the use of approximation techniques or more efficient model architectures.

Another critical issue is data sparsity and imbalance. Real-world graphs frequently exhibit a power-law distribution [41], with a small number of nodes having many connections and a large number having very few. Additionally, missing connections or underrepresented classes can result in biased learning outcomes, where models fail to generalize to less common structures or patterns.

These limitations highlight the need for generative approaches that can address data sparsity and imbalance without compromising scalability. One such approach involves the use of VAEs, which learn latent representations of graph structures and can generate synthetic data to enhance model training. The following section explores the role of VAEs in graph generation and augmentation in more detail.

## 2.3 Autoencoder and Variational Autoencoder

While graph embeddings offer effective low-dimensional representations of graph data, they are generally designed to encode existing graphs. For data augmentation purposes, however, it is essential to generate new, plausible graphs that preserve the structural characteristics of the original dataset.

Autoencoders [42] are a class of neural networks that learn efficient latent representations by encoding input data into a lower-dimensional space and then reconstructing it. An autoencoder consists of an encoder, which compresses the input into a latent vector, and a decoder, which attempts to reconstruct the original input from this latent representation.

However, traditional autoencoders employ deterministic mappings and thus lack variability in their learned representations. This limits their capacity to generate truly novel samples, as they are primarily focused on reconstruction rather than generation.

Variational Autoencoders (VAEs) [5] address this limitation by providing a probabilistic framework for both representation learning and data generation. Unlike standard autoencoders, VAEs learn a probabilistic mapping between the observed data and a continuous latent space. This introduces variability into the representations, enabling the generation of new samples that resemble the training data but are not identical to it. This makes VAEs particularly suitable for generative tasks where the objective is not only to encode original data but also to synthesize new examples.

The training objective of a VAE is to maximize the Evidence Lower Bound (ELBO), which serves as a lower bound on the log-likelihood of the observed data [43]:

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}[q_\phi(z|x) \parallel p(z)]$$

Here, $x$ denotes the observed data, $z$ is the latent variable, $p_\theta(x|z)$ represents the decoder's likelihood function, $q_\phi(z|x)$ is the encoder's approximate posterior, and $p(z)$ is the prior over the latent variables, typically assumed to be a standard normal distribution $\mathcal{N}(0, I)$. The term $D_{KL}$ denotes the Kullback-Leibler divergence, which penalizes deviation of the learned posterior from the prior.

To encourage disentangled and more interpretable latent representations, a generalization of the VAE known as the $\beta$-VAE [44] introduces a scaling factor $\beta$ to the KL divergence term in the ELBO:

$$\mathcal{L}_\beta(\theta, \phi; x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] - \beta D_{KL}[q_\phi(z|x) \parallel p(z)]$$

This modification enables the model to trade off reconstruction accuracy against latent space regularization, allowing more control over the generative properties of the model.

## 2.4 Variational Graph Autoencoder

The Variational Graph Autoencoder (VGAE) [45] extends the VAE framework to graph-structured data. In this setting, the encoder maps both the graph structure and node features into a latent space, and the decoder reconstructs the graph, typically in the form of an adjacency matrix and optionally node labels.

The VGAE objective mirrors that of the standard VAE but is adapted to graphs. Let $G = (V, E)$ denote a graph, where $V$ is the set of nodes and $E$ the set of edges. The goal is to learn a variational distribution $q_\phi(z|G)$ over latent graph representations. The ELBO for the VGAE is given by:

$$\mathcal{L}_{VGAE}(\theta, \phi; G) = \mathbb{E}_{q_\phi(z|G)}[\log p_\theta(G|z)] - D_{KL}[q_\phi(z|G) \parallel p(z)]$$

Here, $p_\theta(G|z)$ denotes the graph decoder's likelihood, $q_\phi(z|G)$ is the approximate posterior inferred by the graph encoder, and $p(z)$ is typically chosen as a standard normal prior $\mathcal{N}(0, I)$.

The encoder learns a probabilistic mapping from the graph $G$ to a latent representation $z$:

$$q_\phi(z|G) = \mathcal{N}(z; \mu_\phi(G), \sigma^2_\phi(G))$$

The decoder then reconstructs the graph from the latent code:

$$p_\theta(G|z) = \prod_{(u,v) \in V \times V} p_\theta(e_{uv}|z)$$

Variational Graph Autoencoder models enable not only the reconstruction of observed graphs but also the generation of new graphs sampled from the learned latent space, making them a valuable tool for data augmentation and exploration of structural variations within graph datasets.

## 2.5 Related Work

Data augmentation is a widely used technique to improve model performance by artificially increasing the diversity of training data through transformations or the generation of synthetic samples. In computer vision, standard augmentation techniques include image rotation, scaling, and flipping, which help models generalize by introducing controlled variations to the input data [8].

In contrast, data augmentation for graph-structured data poses unique challenges due to the non-Euclidean nature and discrete topology of graphs. Conventional augmentation methods in this domain include node feature masking, edge perturbation (e.g., adding or removing edges), and subgraph sampling [4]. These techniques aim to inject variability while maintaining the underlying structural semantics, thereby enhancing the robustness and generalization capabilities of graph neural networks (GNNs). For instance, adversarial edge dropping [46] has been shown to prevent overfitting by encouraging the model to rely on more general patterns rather than memorizing specific connectivity details.

Variational Autoencoders have been successfully applied across various domains for data augmentation tasks. In computer vision, VAEs are commonly used to generate synthetic images [8], expanding training datasets in domains such as medical imaging, where acquiring labeled data is costly and time-consuming. In natural language processing, VAEs have been utilized to generate diverse sentence structure [9],

contributing to improvements in machine translation, text classification, and sentiment analysis. These examples highlight VAEs' strength in modeling complex data distributions and generating realistic, high-fidelity samples.

The application of VAEs to graph data has also gained increasing attention. Notable early work includes the contributions by Ma et al. [16] and Simonovsky et al. [17], who employed VAE-based architectures for the generation of small graphs. These models were evaluated using standard metrics derived from the ELBO, establishing a foundational framework for generative modeling in the graph domain.

Beyond VAEs, other generative models such as GANs have also been explored for data augmentation. Techniques like the Synthetic Minority Over-sampling Technique (SMOTE) [47] address class imbalance by generating synthetic samples for underrepresented classes, improving classification performance in imbalanced datasets. While GANs have shown success in continuous domains such as images, their adaptation to graphs remains more complex due to the discrete nature of graph structures.

Recent work has also examined the impact of different interpolation strategies within the latent space of generative models. In particular, spherical linear interpolation (SLERP) [48] has been explored as a method to better preserve the geometric properties of the latent manifold compared to standard linear interpolation for graph data. Highlighting how interpolation choice can significantly influence the quality and diversity of generated samples [49], especially in models like VAEs that rely on a structured latent space.

In the context of graph-based learning, a line of work focuses specifically on data augmentation to improve downstream task performance. Strategies include topological modifications, node or edge feature perturbations, and the application of generative models to synthesize entire graphs. Recent developments, such as the AutoGDA framework [50], propose automated augmentation strategies tailored to node classification tasks. These approaches aim to alleviate challenges such as data scarcity and class imbalance by learning optimal augmentation policies directly from the data.

# 3

# VAGUE Method

This chapter outlines the methodology developed and used in this thesis. We start with a high-level overview of the full pipeline in Section 3.1. Section 3.2 presents the core component of the pipeline, our VAE, including details on the encoder, decoder, node label classifier, and the loss function used during training. Section 3.3 describes the data augmentation procedure based on interpolations in latent space. The final component, the downstream evaluation, is introduced in Section 3.4, where we compare the performance of a deep learning-based classifier with a graph-based classifier to assess the quality of the generated synthetic data.

## 3.1 Overview

The Variational Autoencoder for Graph Understanding and Expansion (VAGUE) pipeline comprises two key components: a VAE for learning structured graph representations and a data augmentation strategy for generating synthetic graphs. These synthetic samples are subsequently used to enhance the performance of a downstream graph classification task.

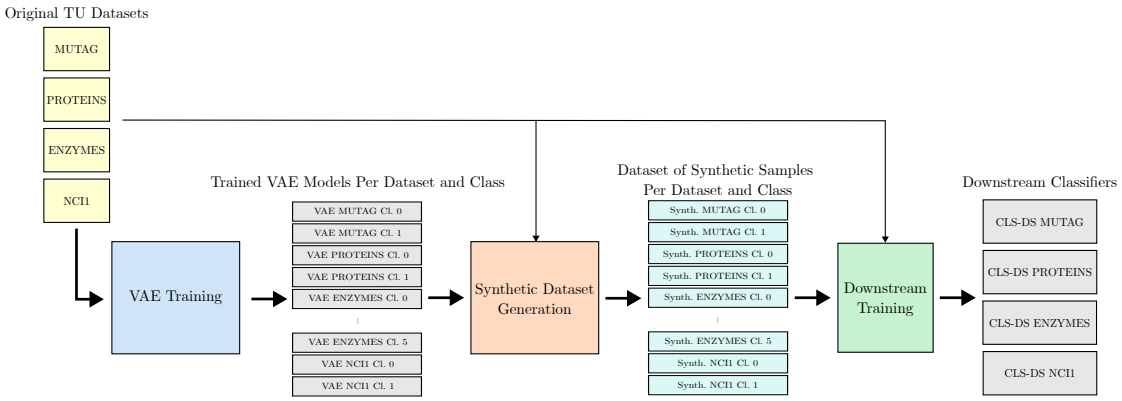


Figure 3.1: Overview of the VAGUE pipeline.

Figure 3.1 contains a overview over VAGUE. The overarching goal is to take an original dataset (yellow) and generate additional synthetic datasets (light blue) which then improve the performance of a given downstream task (light grey). The named components of *VAE Training*, *Synthetic Dataset Generation* and *Downstream Training* are laid out in detail in the following sections.

## 3.2 Variational Autoencoder

Figure 3.2 provides a more detailed visualization of the VAE training process, and the following subsections elaborate on each component of the model.

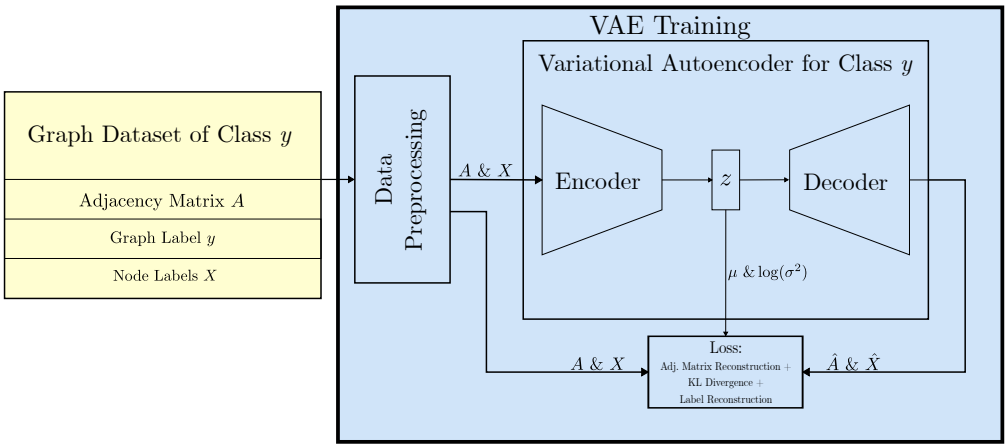


Figure 3.2: Overview of the Variational Autoencoder architecture used in VAGUE.

For each dataset and class, an individual VAE model is trained, enabling class-conditional generation of synthetic graphs without relying on an auxiliary classifier. The VAE architecture follows a standard encoder–decoder approach and is extended with a classifier to predict node labels.

Let $A \in \{0,1\}^{n \times n}$ denote the adjacency matrix of a graph with $n$ nodes, flattened to an input vector $a \in \mathbb{R}^{n^2}$, and let $X \in \{0, \ldots, C\}^n$ denote the associated node label vector, where $C$ refers to the number of node classes.

In the following, we provide a detailed explanation of each component of the VAE used in this thesis. A comprehensive overview and visualization of the complete VAE architecture is available in Appendix Section B.1.

### 3.2.1 Encoder

The encoder maps the input vector $a$ into a latent representation $z$. It consists of two fully connected layers with batch normalization and ReLU activations:

$$h_1 = \text{ReLU}(\text{BN}_2(W_2 \cdot \text{ReLU}(\text{BN}_1(W_1 a + b_1)) + b_2))$$

The encoder outputs the parameters of a multivariate Gaussian distribution in the latent space:

$$\mu = W_\mu h_1 + b_\mu$$
$$\log \sigma^2 = W_{\log \sigma^2} h_1 + b_{\log \sigma^2}$$

To enable backpropagation through the sampling step, the reparameterization trick is applied:

$$z = \mu + \sigma \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

The resulting vector $z$ represents the learned latent encoding of the input $A \to a$, encapsulating the key information the model derives from the input graph $G$.

### 3.2.2 Decoder

The decoder $p_\theta(a \mid z)$ reconstructs the input adjacency matrix using three fully connected layers with dropout, batch normalization, and ReLU activations:

$$\hat{a} = \sigma(W_5 \cdot \text{Drop}(\text{ReLU}(\text{BN}(W_4 \cdot \text{Drop}(\text{ReLU}(\text{BN}(W_3 z + b_3))) + b_4))) + b_5)$$

The output $\hat{a} \in [0,1]^{n^2}$ is reshaped into $\hat{A} \in [0,1]^{n \times n}$, representing the reconstructed adjacency matrix.

### 3.2.3 Node Label Classifier

A classifier is incorporated to predict the node labels based on the latent representation $z$:

$$\hat{X} = \text{reshape}(W_7 \cdot \text{Drop}(\text{ReLU}(W_6 z + b_6)) + b_7)$$

The output $\hat{X} \in \mathbb{R}^{n \times C}$ contains the logits for each node.

### 3.2.4 Loss Function

The total training loss for the VAE model combines three components: (1) a reconstruction loss for the adjacency matrix entries using binary cross-entropy (BCE), (2) a Kullback–Leibler (KL) divergence between the approximate posterior and a unit Gaussian prior, and (3) a classification loss for node label prediction using cross-entropy:

$$\mathcal{L}_{\text{recon}} = \text{BCE}(\hat{a}, a) = -\sum_{i=1}^{n^2} \left(a_i \log \hat{a}_i + (1 - a_i) \log(1 - \hat{a}_i)\right)$$

$$\mathcal{L}_{\text{KL}} = -\frac{1}{2} \sum_{j=1}^{d_z} \left(1 + \log \sigma_j^2 - \mu_j^2 - \sigma_j^2\right)$$

$$\mathcal{L}_{\text{CE}} = \text{CrossEntropy}(\hat{X}, X) = \sum_{i=1}^{n} \left(-\hat{x}_{i,X_i} + \log\left(\sum_{j=1}^{C} e^{\hat{x}_{i,j}}\right)\right)$$

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{recon}} + \beta \cdot \mathcal{L}_{\text{KL}} + \alpha \cdot \mathcal{L}_{\text{CE}}$$

The weighting parameter $\beta$ is gradually annealed from 0.1 to 1.0 over the course of training, allowing the model to initially prioritize structural reconstruction before emphasizing latent regularization. The classification weight $\alpha$ is set equal to $\beta$, encouraging a similar progression of focus from structure to semantics.

## 3.3 Data Augmentation

The goal of this step is to create a synthetic dataset that closely reflects the original data distribution. Figure 3.3 highlights the process to obtain a new synthetic sample by interpolation between two existing samples in latent space.

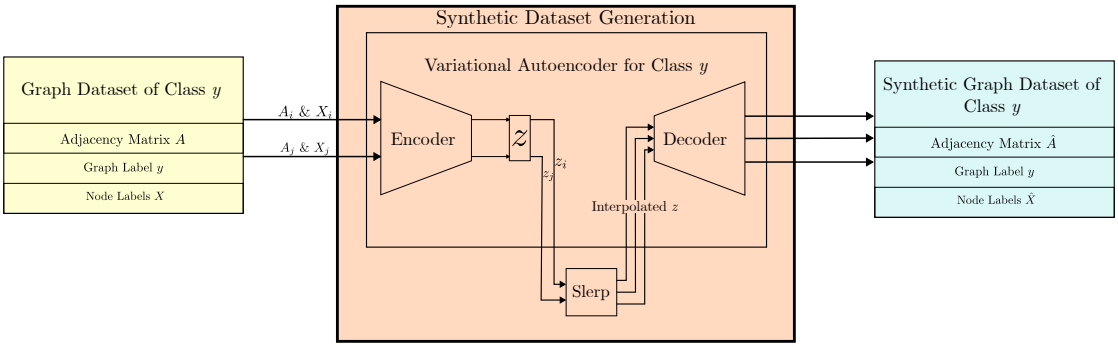


Figure 3.3: Overview of the data augmentation step used in VAGUE

We encode all samples from a given class $y$ into their respective latent representations and interpolate between pairs of different latent vectors $z_i, z_j$ to obtain representations for synthetic samples:

$$\text{synthetic sample } z_s = \text{slerp}(z_i, z_j, t), \text{ with } i \neq j \text{ and } \forall t \in \{0.25, 0.5, 0.75\}$$

A simplified visualization of this approach can be found in Fig. 3.4.

For interpolation, we apply spherical linear interpolation (slerp), commonly used in computer graphics [48]:

$$\text{slerp}(p_0, p_1; t) = \frac{\sin[(1-t)\Omega]}{\sin \Omega} p_0 + \frac{\sin[t\Omega]}{\sin \Omega} p_1,$$

where $p_0$ and $p_1$ are the latent vectors of two original samples, and $\Omega$ is the angle between them with $\cos \Omega = p_0 \cdot p_1$. As $\Omega \to 0$, this expression converges to the standard linear interpolation.

Using a geodesic interpolation method instead of a linear one allows for more natural sampling of latent representations. This improves the quality of the generated data, as samples remain closer to the true data distribution learned by the decoder.

Pseudo code for the generation of synthetic samples in a batched manner can be found in Alg. 2 in Section B.2 of the appendix.

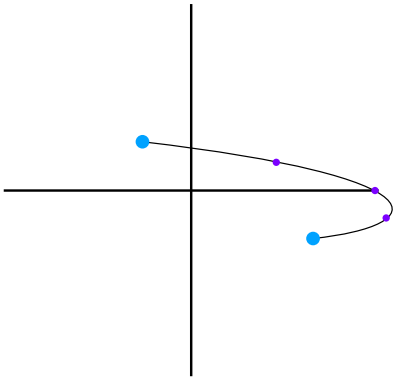

Figure 3.4: Slerp used to obtain new synthetic samples in purple

## 3.4 Downstream Evaluation Model

To evaluate the impact of our synthetic data, we train both a traditional graph-based classifier and a deep learning-based evaluation model. This dual setup enables us to compare classical and modern learning paradigms, ensuring that observed improvements are not biased toward a particular model family. The graph-based method relies on kernel-based similarity and structural properties, whereas the deep learning model assesses how well the synthetic data integrates into end-to-end representation learning. By analyzing performance gains in both settings, we validate the usefulness and generality of our augmentation approach.

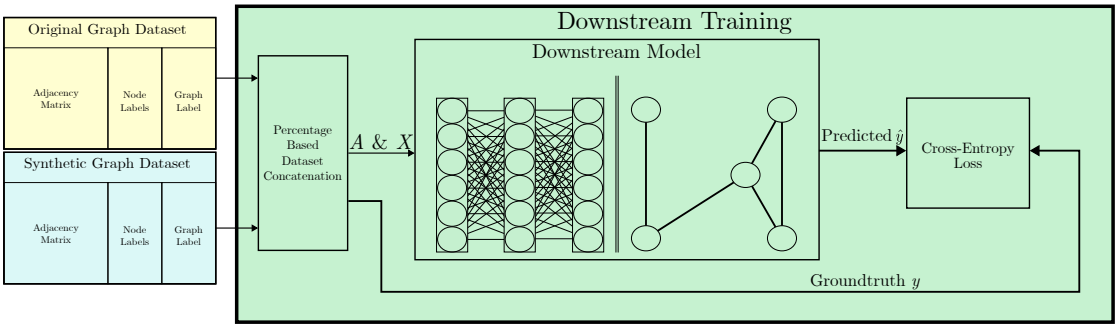


Figure 3.5: Overview of the downstream evaluation step in VAGUE

Figure 3.5 higlights the combination of the original and synthetic dataset into an extended dataset as well as the general classification training setup used to evaluate the quality of the data.

To construct the extended dataset, we add a specified number of synthetic graphs, balanced across all classes:

$$\#\text{ synthetic samples} = \frac{\%\text{ of added data} \cdot \#\text{ original samples}}{\#\text{ graph classes}}$$

Figure 3.6 shows this procedure of concatenating both original and synthetic datasets to gain extended datasets with different levels of added synthetic data.

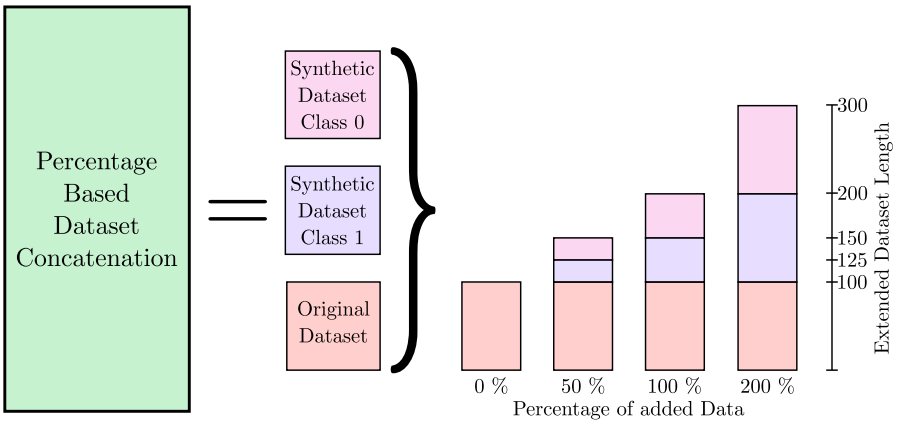


Figure 3.6: Combining original and synthetic samples to form the extended dataset

For example, in the case of the MUTAG dataset, adding 100% synthetic data results in 188 additional samples. These are evenly split between the two classes: 94 graphs from the *Synth. MUTAG Cl. 0* dataset and 94 from the *Synth. MUTAG Cl. 1* dataset. The resulting extended dataset would therefore contain $188 + 188 = 376$ samples.

### 3.4.1 Deep Learning-Based Classifier

The deep learning model is a lightweight feedforward classifier. The input consists of the flattened adjacency matrix $a \in \{0,1\}^{n \times n}$ concatenated with the node label vector $X \in \{1, \ldots, C\}^n$, resulting in an input $x$ of dimension $n \times n + n$.

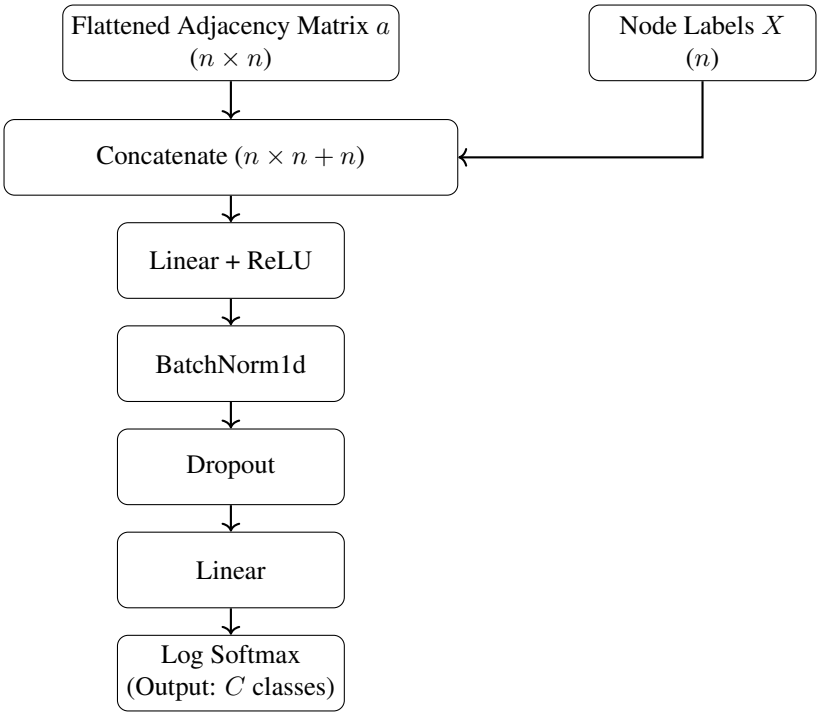


Figure 3.7: The `CLHeader` classification head used to predict graph-level labels.

The classifier, displayed in Fig. 3.7, is composed of two fully connected layers, with batch normalization and dropout applied between them to improve generalization and training stability. The resulting output of this classification head is formally given as:

$$\hat{y} = \log \mathrm{Softmax}\left(W_2 \cdot \mathrm{Dropout}\left(\mathrm{BN}\left(\mathrm{ReLU}\left(W_1 \cdot x + b_1\right)\right)\right) + b_2\right)$$

### 3.4.2 Graph-Based Classifier

For the traditional graph-based method, we use the Weisfeiler-Lehman subtree kernel in combination with a Support Vector Machine for graph classification. This method enables the use of kernel-based learning algorithms on structured graph data by mapping graphs into a high-dimensional feature space defined by subtree patterns.

After mapping graphs to feature vectors $\phi(G)$, we compute a kernel matrix $K(G_i, G_j) = \langle \phi(G_i), \phi(G_j) \rangle$ and train a SVM on it. The classifier finds a decision boundary that maximizes the margin between classes in this space, enabling robust graph-level classification.

# 4

# Experiments

This chapter presents the experimental evaluation of the proposed VAGUE framework and the resulting synthetic datasets. Section 4.1 outlines the experimental setup, including details on dataset preprocessing and preparation as well as the training configurations and model parameters. The results are reported in Section 4.2, starting with a qualitative assessment of the generated graphs and latent space organization, followed by a quantitative evaluation of downstream classification performance with and without data augmentation. Finishing with ablation studies examining the influence of different augmentation strategies, node label information, and variations of the $\beta$-VAE training regime on overall performance.

## 4.1 Setup

This section provides an overview of the experimental setup used to evaluate the extended datasets. Subsection 4.1.1 details the choice of datasets, their relevant statistics, and the preprocessing steps applied to prepare them for training and evaluation. Subsection 4.1.2 describes the model architecture, the selected hyperparameters, and other training configurations used throughout the experiments.

### 4.1.1 Dataset Setup

The data used in this work is derived from the TU Dataset collection [51]. We select four datasets listed in Tab. 4.1, each split into a fixed 85/15 training and testing partition. These datasets differ in several key metrics, including total size, average number of edges, and, most importantly, the number of classes. This diversity enables a meaningful comparison and evaluation of the synthetic data.

| Dataset | Total Size | Avg. # Nodes | Avg. # Edges | # Classes |
|---|---|---|---|---|
| MUTAG | 188 | 17.93 | 19.79 | 2 |
| PROTEINS | 1113 | 39.06 | 72.82 | 2 |
| NCI1 | 4110 | 29.87 | 32.30 | 2 |
| ENZYMES | 600 | 32.63 | 62.14 | 6 |

**Table 4.1:** Statistics of the benchmark graph datasets used in our experiments.

MUTAG, PROTEINS and NCI1 are all binary graph classification problems:

MUTAG [52] is a dataset of 188 mutagenic aromatic and heteroaromatic nitro compounds. Each graph represents a molecule, with vertices denoting atoms and edges representing chemical bonds. The task is to classify compounds based on their mutagenicity on the Salmonella enterica serovar Typhimurium bacterium.

The PROTEINS [53] dataset contains graphs derived from protein structures. Nodes represent secondary structure elements, and edges indicate neighborhood relations. Each graph is labeled as either an enzyme or a non-enzyme in the context of protein function prediction.

NCI1 [54] is a subset of the National Cancer Institute's screening data, comprising over 4,000 chemical compounds screened for activity against non-small cell lung cancer. Each compound is represented as a graph, with atoms as nodes and bonds as edges. The task is to classify compounds as active or inactive based on their ability to inhibit cancer cell growth.

In contrast to these binary problems we also work with a multi-class graph classification problem like ENZYMES, which allows us to evaluate the quality of the synthetic data on variety of slightly different graphs and a more challenging task.

The ENZYMES [55] dataset consists of 600 protein graphs categorized into six enzyme classes, following the Enzyme Commission classification. Each node corresponds to a secondary structure element, and edges encode spatial closeness. In this dataset the task is to classify enzymes into one of the six top-level EC classes, each representing a distinct type of catalyzed chemical reaction.

We preprocess each dataset by cropping or padding the adjacency matrices $A \in [0,1]^{n \times n}$ to a uniform size of $n = 100$, such that $A \in [0,1]^{100 \times 100}$ throughout this work. Similarly, node label vectors are resized or padded from $[n, C]$ to $[100, C]$, where $C$ denotes the number of node classes present in the dataset.

### 4.1.2 Model Setup

The VAE model is configured with `max_num_nodes` set to 100, this is corresponding to an accepted input adjacency matrix of shape $100 \times 100$ and a node label vector of length 100. This value was selected based on an analysis of the original datasets, ensuring that over $90\%$ of graphs are retained in their entirety to minimize unnecessary truncation.

We further trained the VAE with a latent dimension of `latent_dim` $= 64$ and a hidden layer size of `hidden_dim` $= 1024$. Training is conducted for a maximum of 1000 epochs, with early stopping enabled

and evaluation occurring every $100$ epochs. The batch size `train_batch_size` varies by dataset, ranging from $16$ to $256$.

For the data augmentation step, we interpolate between the latent representations of pairwise distinct original samples. Three interpolated samples are generated for each pair at $t$ values of $t \in \{0.25, 0.5, 0.75\}$.

The downstream deep learning-based classifier, referred to as CLS-DS, uses the same input representation and a hidden dimension of $\texttt{hidden\_dim} = 64$. In addition, we employ a graph kernel-based classifier using the Weisfeiler-Lehman kernel with $\texttt{n\_iter} = 3$ iterations and a `VertexHistogram` kernel.

Pseudo code for training of both the VAE as well as the downstream classifier CLS-DS can be found in Appendix B.2.

## 4.2 Results

In this section, we present and analyze the outcomes of our experiments to evaluate the performance of the VAGUE framework. Subsection 4.2.1 explores qualitative results, showcasing visual and structural properties of generated graphs. Subsection 4.2.2 follows with a quantitative evaluation, measuring the downstream classification performance of models trained on datasets augmented using our method. Finally, Subsection 4.2.3 provides ablation studies that examine the impact of individual components of our approach, such as different augmentation strategies, the use of node labels, and the effect of varying the $\beta$ parameter in the VAE objective.

### 4.2.1 Qualitative Results

This subsection presents qualitative results evaluating the performance of different VAEs within the VAGUE framework and visualizes some of their limitations. Specifically, we analyze two VAE models trained on the NCI1 dataset, one for class 0 and one for class 1.

Graph reconstructions are generated by encoding an original graph $G_i$ to obtain its latent representation $z_i$ using the VAE encoder, followed by decoding $z_i$ with the decoder to produce the reconstructed graph $\hat{G}_i$. Additionally, we visualize the learned latent space by applying Principal Component Analysis (PCA) [56] to reduce the dimensionality from $64$ to the two leading components.

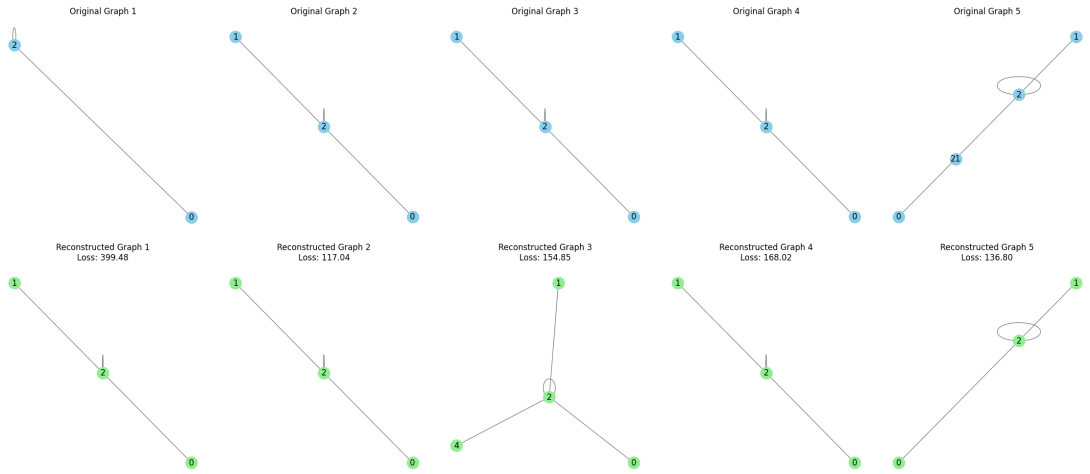


Figure 4.1: Qualitative test set results of the VAE model trained on NCI1 Class 0.

Figure 4.1 displays samples from the NCI1 class 0 test set alongside their reconstructed counterparts. While Graphs 2 and 4 are reconstructed accurately, others reveal minor discrepancies. For instance, Graphs 1 and 3 include an extra node that was not present in the original graph. Conversely, in Graph 5, node 21 is missing in the reconstructed graph $\hat{G}$, despite being present in the original input.

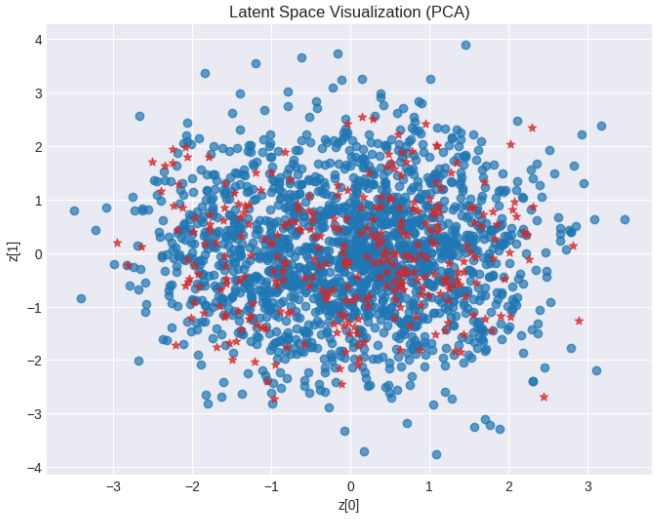


Figure 4.2: Visualization of the latent space learned by the VAE model for NCI1 Class 0. Blue: training data; red: test data.

The latent space visualization in Fig. 4.2 illustrates the distribution of training and test samples. The KL divergence objective encourages the latent distribution to approximate a standard Gaussian $\mathcal{N}(0, I)$

by minimizing $\mu \to 0$ and $\sigma^2 \to 1$. The zero-mean property is achieved, while the observed variance greater than one may result from PCA transformation and the inherent variability of the learned latent representations.

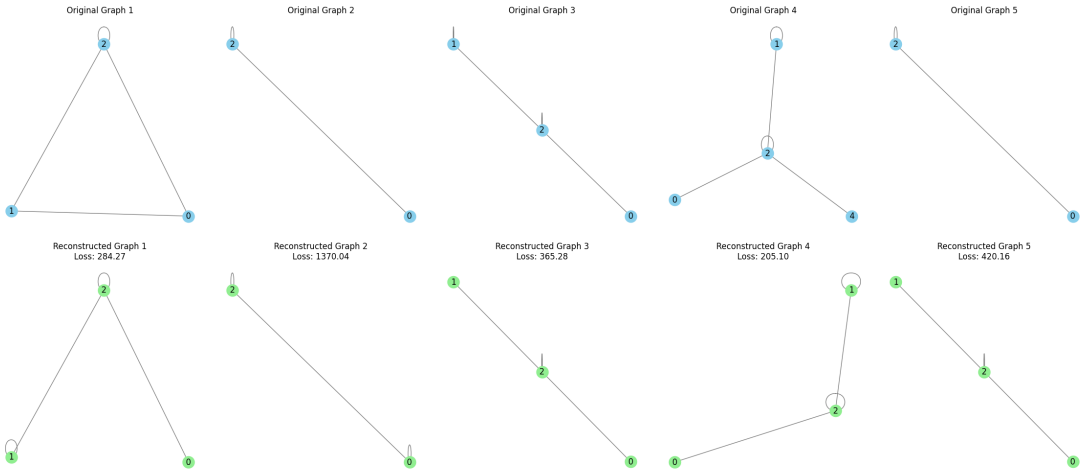


Figure 4.3: Qualitative test set results of the VAE model trained on NCI1 Class 1.

Similarly, Fig. 4.3 presents reconstructed samples from the NCI1 class 1 test set. In this case, all reconstructed graphs $\hat{G}$ exhibit some level of deviation from the originals. These include incorrect edge connections (e.g., Graph 1), mismatched self-loops (e.g., Graphs 2 and 3), and added or omitted nodes (e.g., Graphs 4 and 5).

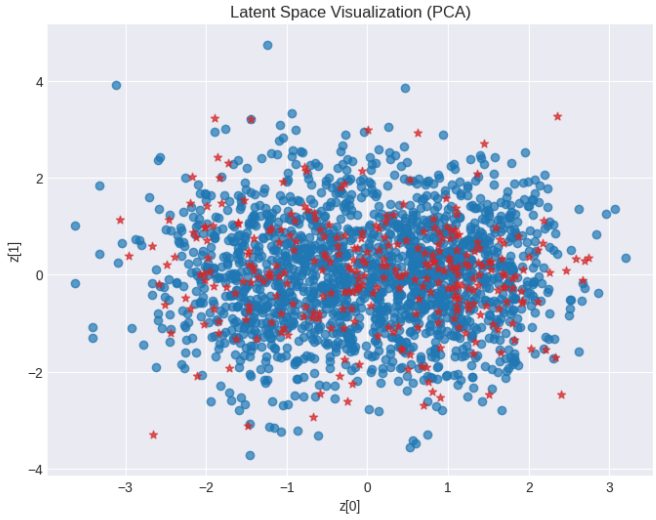


Figure 4.4: Visualization of the latent space learned by the VAE model for NCI1 Class 1. Blue: training data; red: test data.

The latent space corresponding to class 1, shown in Fig. 4.4, displays a qualitatively similar behavior. However, the first principal component appears to capture more variance than the second, leading to a distribution that is more elongated and less spherical.

#### Visualization of one Interpolation Step

To qualitatively assess the quality of the generated synthetic samples, we visualize an interpolation between two graphs from the NCI1 Class 0 training dataset. Specifically, we encode both graphs, $G_i$ and $G_j$, into the latent space using the VAE encoder.

Following the same procedure used for generating synthetic samples, we interpolate between their latent representations $z_i$ and $z_j$ at interpolation factors $t \in \{0.25, 0.5, 0.75\}$. The results are shown in Fig. 4.5, where the original graphs $G_i$ and $G_j$, along with their reconstructions $\hat{G}_i$ and $\hat{G}_j$, are presented. Between these, the reconstructed graphs corresponding to the interpolated latent vectors at each $t$ value are displayed, illustrating the smooth transition between the two original graphs.

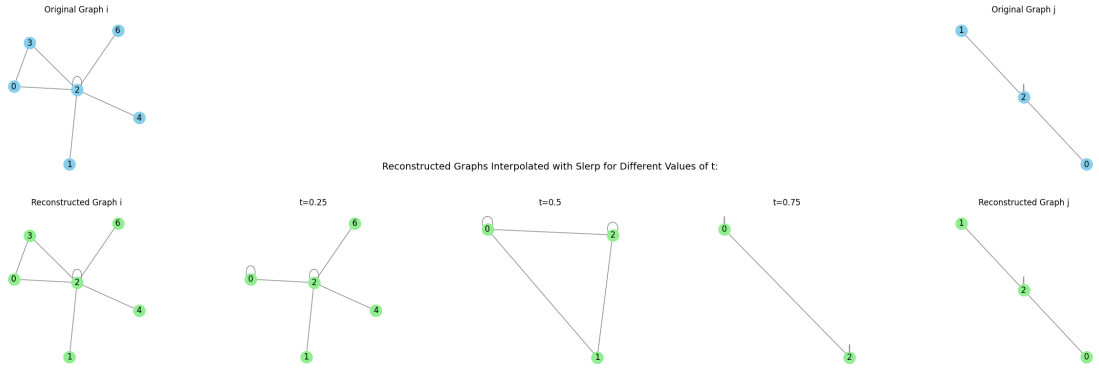


Figure 4.5: Visualization of Interpolated Synthetic Graphs between two Original Graphs

In summary, these results serve as a qualitative proof of concept for the VAGUE pipeline and its potential in data augmentation for graph-based learning tasks. Although there remains room for improvement in the visual quality and structural consistency of the generated graphs, the current outputs are already of sufficient quality to yield measurable benefits. This is shown by the improved performance of downstream models trained on the augmented data, as will be demonstrated in the quantitative evaluation presented in the following subsection.

### 4.2.2 Quantitative Results

In this subsection, we present a detailed quantitative evaluation of the proposed method. First we assess the performance of deep learning-based downstream models on the augmented datasets. Next, we evaluate the same datasets on a graph-based classifier and finally we present a visualization of the extended testset to further explain the positive contribution of the extended dataset.

### Deep Learning-Based Evaluation

Table 4.2 presents the test accuracies of downstream classifier models trained on varying amounts of added synthetic data, as introduced in Section 3.3. The amount of synthetic data ranges from $0\%$ (using only the original dataset) to $1500\%$ (meaning the synthetic data was 15 times the size of the original dataset).

For each percentage, a separate classifier was trained using identical configurations, differing only in the quantity of training data. The test set always comprised only the original test data to ensure consistency and to avoid overrepresenting synthetic samples in the evaluation.

**Table 4.2:** Test Accuracies for Deep Learning (DS-CLS) Evaluation

| | | Percentage of Added Synthetic Data | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **0%** | **100%** | **250%** | **500%** | **750%** | **1000%** | **1250%** | **1500%** |
| **Dataset** | **MUTAG** | 82.1 | 92.9 | 89.3 | 92.9 | 89.3 | 92.9 | 85.7 | 89.3 |
| | **PROTEINS** | 74.1 | 80.7 | 88.0 | 87.4 | 88.6 | 92.2 | 89.8 | 91.0 |
| | **ENZYMES** | 46.7 | 73.3 | 78.9 | 74.4 | 78.9 | 73.3 | 82.2 | 81.1 |
| | **NCI1** | 65.6 | 81.5 | 84.7 | 87.3 | 89.6 | 89.1 | 89.8 | 90.8 |

Figure 4.6 summarizes the results shown in Tab. 4.2, displaying performance across the four datasets with increasing synthetic data:

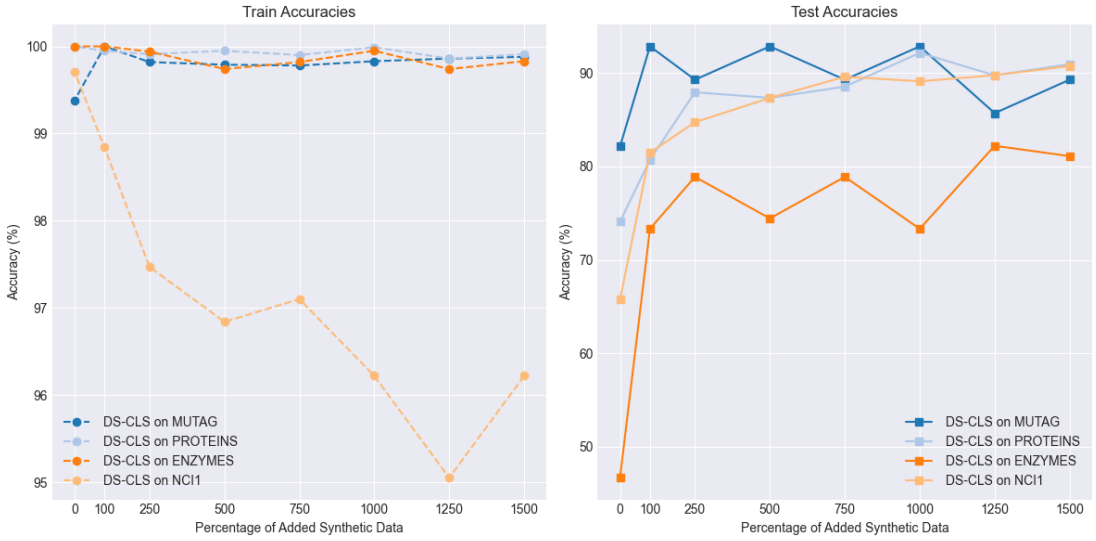


Figure 4.6: Evaluation of DL Downstream Classifier

Incorporating synthetic data into the training set consistently leads to improved test accuracy across all evaluated scenarios. While the inclusion of such data may introduce some outliers or less representative graphs, this variability promotes greater model robustness, which is advantageous for real-world applications as the model is able to generalize more accurately to unseen data.

The degree of improvement varies by dataset. This is partly due to differences in dataset size. For example, MUTAG contains only 188 samples (with 160 used for training), whereas NCI1 contains 4110 samples (3494 used for training). The negative effect of the added data to the NCI1 dataset is interesting as it does not carry over to the test set. This interaction may stem from the fact that the NCI1 dataset contains more variability and MUTAG is more tightly controlled in structure. Additionally, the average graph size differs: MUTAG graphs have roughly 10 fewer nodes than those in NCI1. Combined with better VAE performance on MUTAG, this may explain the different trends.

Notably, doubling the training data ($100\%$ added synthetic data) consistently yields higher test accuracy compared to the baseline ($0\%$ added synthetic data). This improvement stems from increased data diversity and reduced variance, allowing the downstream model to better estimate parameters and generalize more effectively. Proving the success of our VAGUE approach in the field of generative graph-data augmentation.

### Graph-Based Evaluation

To assess whether the improvements in downstream performance are consistent or merely coincidental, we re-evaluate the same downstream task using a graph-based classification approach. Table 4.3 presents the performance of a graph kernel classifier based on the Weisfeiler-Lehman kernel. These results further support the contribution of synthetic data in enhancing model performance, consistent with the findings from the deep learning-based evaluation.

**Table 4.3:** Test Accuracies for Graph-Based (Weisfeiler-Lehman) Evaluation

| | | **Percentage of Added Synthetic Data** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **0%** | **100%** | **250%** | **500%** | **750%** | **1000%** | **1250%** | **1500%** |
| **Dataset** | **MUTAG** | 82.1 | 89.3 | 92.9 | 92.9 | 92.9 | 92.9 | 92.9 | 96.4 |
| | **PROTEINS** | 77.1 | 83.1 | 87.4 | 88.0 | 89.2 | 90.4 | 90.4 | 91.6 |
| | **ENZYMES** | 52.2 | 54.4 | 56.7 | 57.8 | 58.9 | 56.7 | 55.6 | 57.8 |
| | **NCI1** | 84.3 | 84.9 | 86.9 | 89.8 | * | * | * | * |

Due to the large size of the NCI1 dataset and hardware constraints, evaluations with $\geq 750\%$ synthetic data could not be completed and are marked with asterisks (*).

Figure 4.7 visualizes the results of Tab. 4.3 and provides a useful cross examination to the deep learning-based results discussed above:

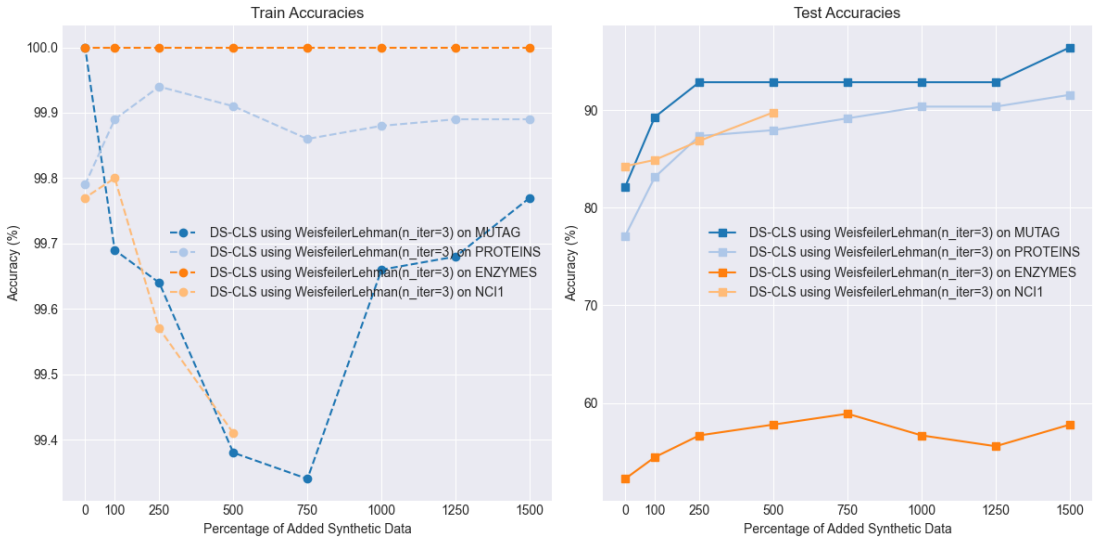


Figure 4.7: Evaluation of Downstream Classifier using Weisfeiler-Lehman Kernel

Compared to deep learning models, the WL classifier appears more sensitive to the quality of training data and less scalable with increasing dataset size. While deep learning models may better generalize from high-variance data, WL-based methods rely more directly on exact structural similarities.

In the binary classification tasks, MUTAG, PROTEINS and NCI1, both the deep learning classifier and WL kernel combined with SVM achieve strong performance because the decision boundary between two classes is relatively simple and fixed feature representations are often sufficient. However, in multi-class settings with more classes in the ENZYMES dataset, the deep learning model outperforms the WL kernel + SVM approach as it can learn task-specific, hierarchical features optimized end-to-end. This adaptability enables it to capture more complex and subtle patterns necessary to distinguish between multiple classes, whereas WL kernel features remain fixed and may lack the expressiveness required for fine-grained multi-class discrimination.

### Visualization of the Extended Dataset

To better understand the improvements in Fig. 4.6 and 4.7, we visualize the extended test set. Each graph $G$ or $\hat{G}$ is represented by flattening its adjacency matrix $A \in [0,1]^{n \times n}$ and concatenating it with the corresponding node label vector, resulting in a vector representation $x \in \mathbb{Z}_{\geq 0}^{n^2+n}$. We compute pairwise Euclidean distances between these vectors and display them in Fig. 4.8.

In Fig. 4.8, samples 0–165 correspond to the original PROTEINS test set. Indices 166–248 correspond to synthetic samples of class 1, and the remainder belong to class 0.

The distances indicate that the synthetic graphs are similar in structure to the original graphs, corroborating the earlier findings that the VAE successfully generated valid samples. This is consistent with the performance gains observed in both deep learning and graph kernel-based classifiers.

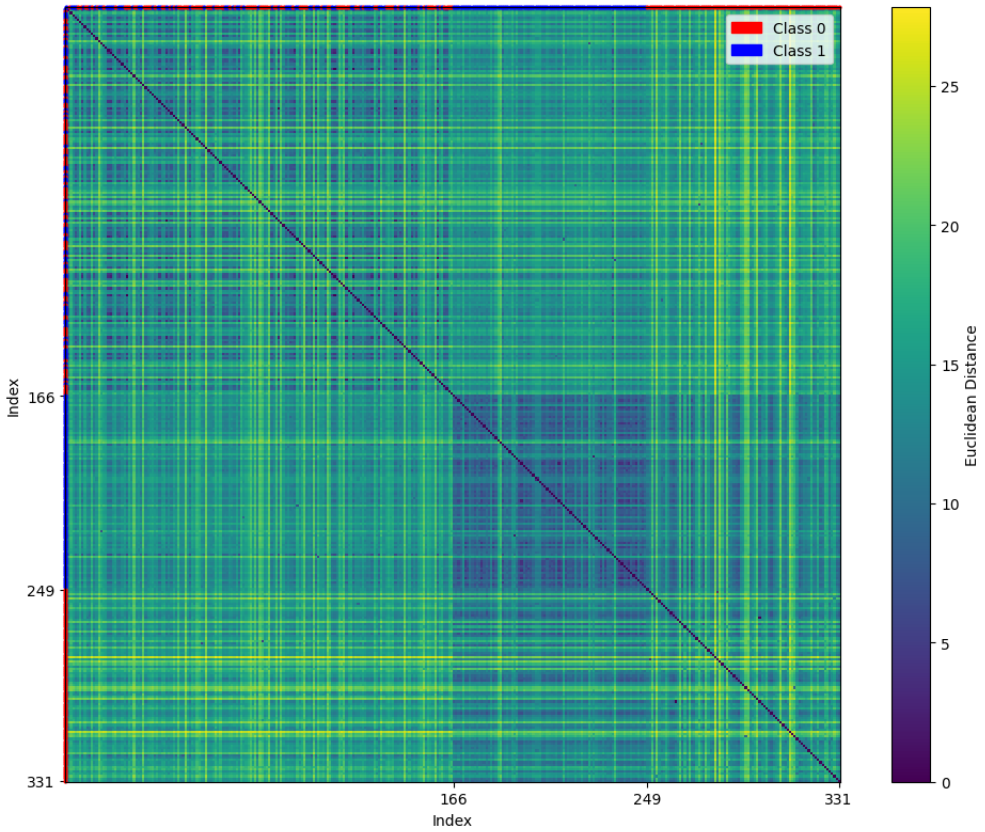


Figure 4.8: Euclidean Distance Between Graph Representations in Extended PROTEINS Test Set

The visual separation between synthetic class 1 and class 0 samples may be attributed to training separate VAEs for each class, resulting in slightly different generative behavior due to different encoder–decoder weights based on the different train sets.

Based on these findings, we conclude that incorporating synthetic data into existing datasets has a significant and consistently positive impact on the performance of downstream models. The augmented data complements the original samples, leading to improved model robustness across all dataset sizes, a variety of graph structures, and different numbers of classes.

To better understand the source of these performance gains, the following subsection examines specific components of the VAGUE pipeline in greater detail. We conduct ablation studies to isolate the contributions of individual elements, providing further insight into the mechanisms that drive the observed improvements.

### 4.2.3 Ablation Studies

This subsection presents a series of ablation studies designed to assess the individual contributions of key components in the proposed approach. Specifically, we investigate the impact of different graph

augmentation strategies, evaluate the role of node labels in classification performance, and examine the influence of varying the $\beta$ parameter in the $\beta$-Variational Autoencoder framework.

### Augmentation Strategies

To assess the impact of different augmentation strategies, we generated new datasets based on the four techniques introduced in Section 3.3:

*random* Sampling latent vectors randomly from the prior: $z_r \sim \mathcal{N}(0, I)$.

*linear* Linear interpolation between an encoded graph and a random latent vector:
$z_l = t \cdot z_i + (1 - t) \cdot z_r$, where $z_i = \text{Encoder}(G_i)$ and $t \in \{0.25, 0.5, 0.75, 1\}$.

*slerp* Spherical linear interpolation between an encoded graph and a random latent vector:
$z_{s1} = \text{slerp}(z_i, z_r, t)$, with $z_i = \text{Encoder}(G_i)$ and $t \in \{0.25, 0.5, 0.75, 1\}$.

*slerp2* Spherical linear interpolation between two distinct encoded graphs (default strategy in this thesis):
$z_{s2} = \text{slerp}(z_i, z_j, t)$, where $z_i, z_j = \text{Encoder}(G_i), \text{Encoder}(G_j)$, with $i \neq j$ and
$t \in \{0.25, 0.5, 0.75\}$.

A simplified overview over these latent space-based interpolation techniques can be found in Fig. 4.9:

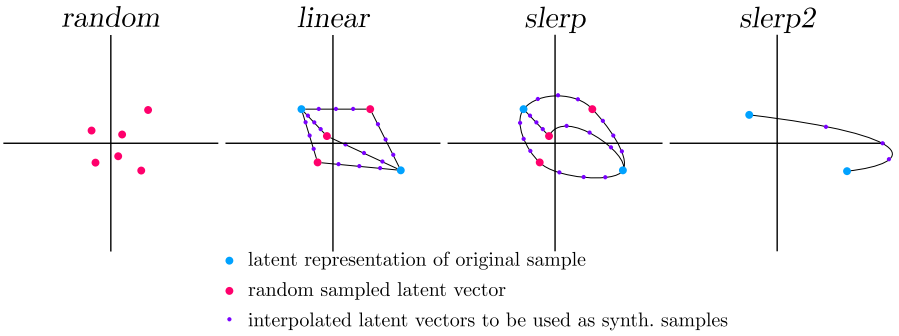


Figure 4.9: Simplified Visualization of Augmentation Strategies

The performance of these augmentation strategies on the PROTEINS dataset is illustrated in Fig. 4.10.

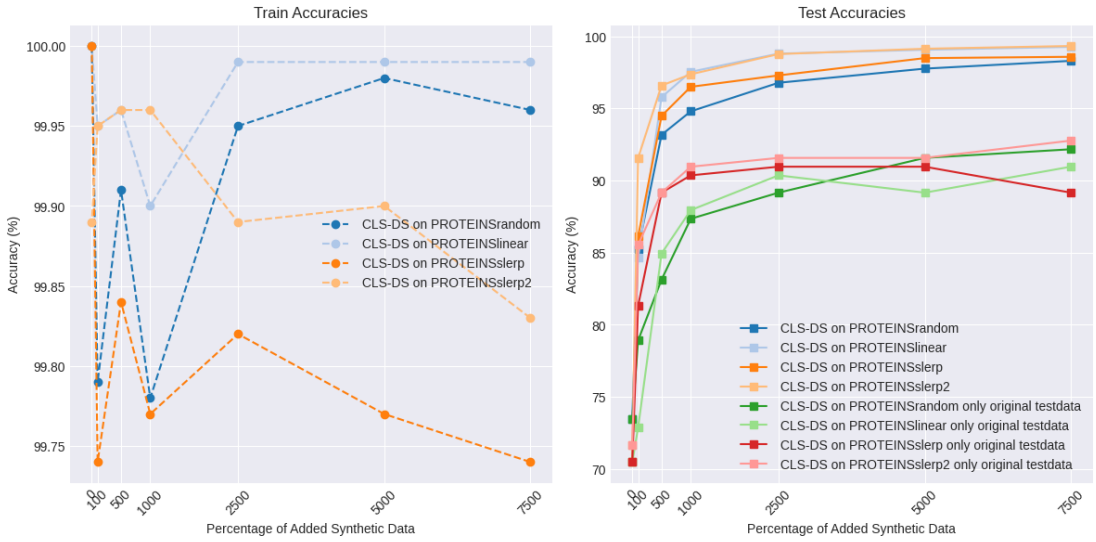


Figure 4.10: Comparison of Augmentation Strategies on PROTEINS Dataset

Figure 4.10 displays the test accuracy for both the complete extended test set and the subset containing only the original test data. As expected, the accuracy on the extended test set is higher, as shown by the performance of *CLS-DS on PROTEINSslerp2* compared to *CLS-DS on PROTEINSslerp2 only original test data*. This discrepancy arises mainly because the synthetic data outweighs the original test samples significantly.

Among the augmentation methods, the *slerp2* strategy consistently outperforms *linear* and *slerp*, and marginally surpasses the *random* baseline. This effect becomes even more pronounced when focusing solely on the performance on the original test data, which is the primary focus of our evaluation. This is because the original test samples may otherwise be overshadowed by the additional synthetic test data. The performance gap is especially notable when using a relatively small amount of additional data, such as $1000\%$ of the original dataset. A likely reason for this advantage is the *slerp2* method's strong reliance on actual data samples during interpolation, which better preserves structural properties and outlier characteristics of the original dataset compared to sampling from the prior.

### Node Labels

To investigate the role of node labels, we trained one model using the PROTEINS dataset with node labels and another without. The results are presented in Fig. 4.11.

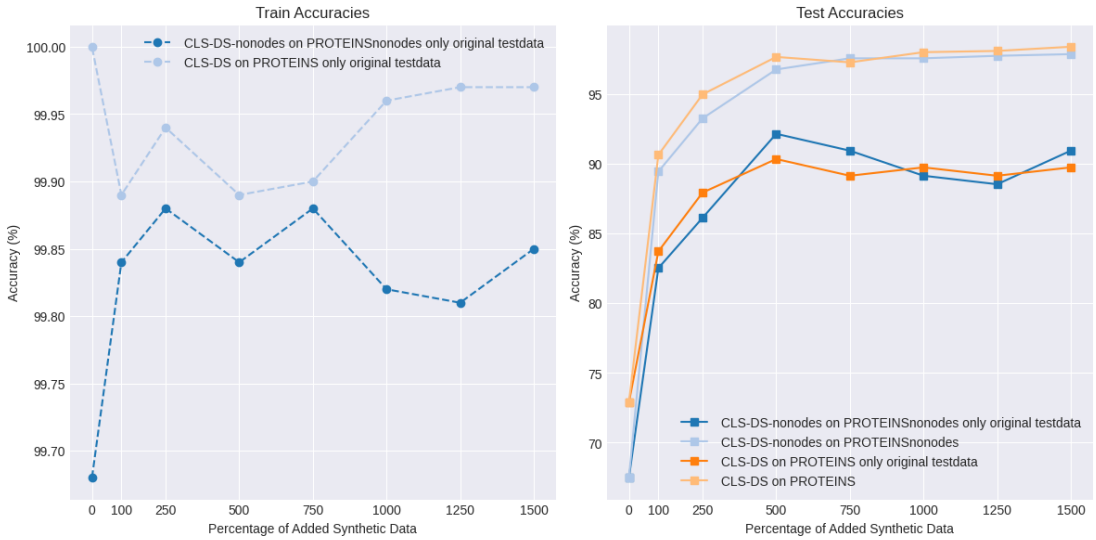


Figure 4.11: Ablation Study on the Inclusion of Node Labels

As shown in Fig. 4.11, the inclusion of node labels significantly improves classification accuracy on the original test set. However, the improvement diminishes when evaluating on the extended dataset. This suggests that the predicted node labels in the synthetic graphs are not sufficiently informative to yield consistent performance gains, and may even hinder the model's accuracy when evaluating only the original data.

### $\beta$-Variational Autoencoder

In accordance with the $\beta$-VAE framework, we conducted an ablation study using three configurations on the PROTEINS dataset:

*CLS-DS* The default model, in which $\beta$ is annealed linearly from $0.1$ to $1$ during training:

$$\beta(e) = 0.1 + \frac{(1 - 0.1)}{E - 1} \cdot (e - 1)$$

where $e$ is the current epoch and $E$ is the total number of epochs.

*CLS-DS2Beta* A variation of the default model, where $\beta$ increases from $0.1$ to $2$ over training.

*CLS-DSNoBeta* A baseline model with a fixed $\beta = 1$ throughout training.

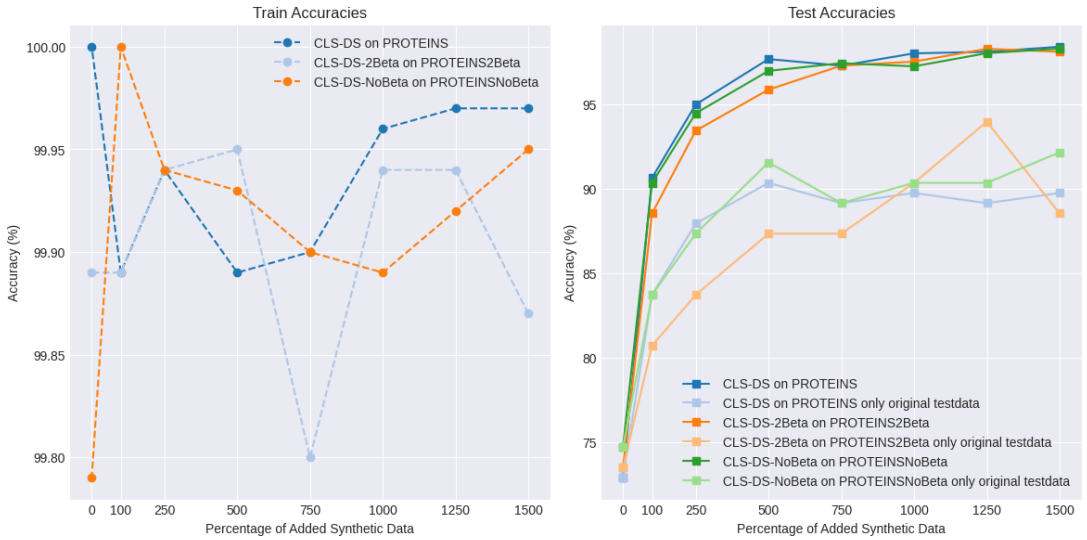


Figure 4.12: Ablation Study on Different $\beta$ Values

Figure 4.12 shows that the $\beta$ parameter had only a minor impact on model performance, possibly due to the simplicity of the VAGUE architecture. Although the strategy of initially focusing on reconstruction and gradually emphasizing the KL divergence (to enforce a Gaussian latent distribution) has theoretical merit, the results did not show significant benefits.

It is important to note that the node label loss coefficient $\alpha$ in this implementation was directly dependent on $\beta$, meaning that changes to $\beta$ also influenced the training of node label prediction. Consequently, variations in $\beta$ indirectly affected final classification performance as well.

# 5

# Conclusions and Future Work

In this work, we explored the use of VAEs for the generation of synthetic graph data. We proposed and evaluated a framework that integrates novel latent space interpolation techniques and assessed the impact of the generated data on downstream classification tasks. This work addresses the growing data demands of deep learning models in combination with the challenges associated with collecting large-scale, high-quality graph datasets. Challenges that are especially prevalent in domains such as bioinformatics, social networks, and chemistry.

The proposed approach consists of three major components. First, a VAE is trained on a limited but high-quality graph dataset. Second, the latent space of the trained VAE is sampled using a novel interpolation technique to generate new synthetic graphs. Lastly, these synthetic graphs are combined with the original data and the resulting extended dataset is evaluated on a downstream classifier. This evaluation was conducted on different sized datasets, highlighting the potential for generalization, as well as graphs of different structures.

The results are promising: even a relatively small amount of generated data leads to a substantial performance boost in classification accuracy. This demonstrates that interpolating in the latent space of graph VAEs can yield useful and realistic samples, which complement and enrich the original training data. Our method offers a flexible, model-agnostic approach to dataset augmentation, applicable across a variety of graph types.

However, the current work also has limitations. The architecture employed is relatively basic and was designed with generalizability in mind rather than task-specific optimization. While this makes the approach broadly applicable, it may not capture the full performance potential for specific graph domains. Moreover, the current VAE implementation is limited in its ability to handle graphs of variable sizes and does not leverage node attributes during encoding.

Future research could address several promising directions.

A key area of improvement lies in the use or development of more advanced encoder and decoder architectures akin to the VGAE, particularly those leveraging GNNs. Using a graph-based encoder could enable the model to capture structural dependencies more effectively, improving the quality and diversity of generated graphs. This would also allow the model to scale to larger graphs and potentially handle graphs of varying sizes more naturally. Additionally, incorporating node features or labels during encoding could enrich the latent representation and lead to more semantically meaningful generations. This would allow the synthetic data to better mirror real-world distributions, especially in datasets where node attributes play a critical role.

It would also be of great interest to test the proposed augmentation method on downstream tasks involving deep learning models tailored for graphs, such as GNN-based classifiers. Evaluating the performance improvements in such settings would provide deeper insights into the utility of the synthetic data for modern graph learning pipelines.

Finally, applying this framework to real-world use cases would serve as a valuable validation of its practical benefits. Such deployment could also reveal new challenges not encountered in proof-of-concept experiments.

As a whole, this work presents a step towards dynamic graph dataset augmentation using VAEs. By offering a flexible and extensible framework, it further opens the door to research in synthetic graph data generation and its impact on downstream tasks. We believe this work contributes to bridging the gap between data scarcity and the growing demand for high-quality graph data in the machine learning community.

# A

# Full Evaluation Results

This appendix presents the complete evaluation results that support the analyses and conclusions discussed in the main body of the thesis. It begins with the VAE-training evaluation results (Section A.1), which report the final training and test losses, providing a concise summary of the model's reconstruction performance at the end of training. This is followed by the downstream evaluation results (Section A.2), where the effectiveness of the generated graph data is assessed through the performance of classifiers trained on both the original and augmented datasets. Lastly, ablation results are presented in more detail in Section A.3 to analyze the contribution of key components within the VAGUE pipeline.

## A.1 VAE-training Evaluation Results

Table A.1 gives us insight into the training and test accuracies of all VAE models which were trained following the description in Sections 3.2 and 4.1.2.

Table A.1: Train and Test VAE Losses for Different Models

| Model | Train Loss | Test Loss |
|---|---|---|
| EncModelCl0_ENZYMES | 28.08 | 45.46 |
| EncModelCl1_ENZYMES | 32.56 | 29.89 |
| EncModelCl2_ENZYMES | 31.08 | 40.40 |
| EncModelCl3_ENZYMES | 36.28 | 44.54 |
| EncModelCl4_ENZYMES | 37.67 | 24.26 |
| EncModelCl5_ENZYMES | 25.27 | 32.96 |
| EncModelCl0_MUTAG | 3.24 | 1.75 |
| EncModelCl1_MUTAG | 9.54 | 5.06 |
| EncModelCl0_NCI1 | 36.18 | 157.01 |
| EncModelCl1_NCI1 | 40.44 | 213.34 |
| EncModelCl0_PROTEINS | 51.13 | 362.35 |
| EncModelCl1_PROTEINS | 38.01 | 96.47 |

As is visible in the Tab. A.1 the model still has some room for improvement as there are outliers in performance but the results are satisfactory for the VAGUE usecase. The difference in performance between two models on the same dataset but different classes stems from the imbalance of the original dataset in respect to the amounts of samples per class as the class distribution is not always even.

## A.2 Downstream Evaluation Results

The following Subsection contains all main results of the VAGUE pipeline discussed in Sections 4.2.1 and 4.2.2 in Tab. A.2.

Table A.2 documents all results for the main contribution of the VAGUE pipeline. The qualitative Evaluation of these results can be found in Section 4.2.1 and the associated quantitative results can be found in Section 4.2.2.

Table A.2: Train and Test Accuracies for Deep Learning (DL) and Graph-based WL Evaluation on DS-CLS

| Dataset | Added Data (%) | DL Train | DL Test | WL Train | WL Test |
|---|---|---|---|---|---|
| **MUTAG** | 0 | 99.38 | 82.14 | 100.0 | 82.14 |
| | 100 | 100.0 | 92.86 | 99.69 | 89.29 |
| | 250 | 99.82 | 89.29 | 99.64 | 92.86 |
| | 500 | 99.79 | 92.86 | 99.38 | 92.86 |
| | 750 | 99.78 | 89.29 | 99.34 | 92.86 |
| | 1000 | 99.83 | 92.86 | 99.66 | 92.86 |
| | 1250 | 99.86 | 85.71 | 99.68 | 92.86 |
| | 1500 | 99.88 | 89.29 | 99.77 | 96.43 |
| **PROTEINS** | 0 | 100.0 | 74.10 | 99.79 | 77.11 |
| | 100 | 99.95 | 80.72 | 99.89 | 83.13 |
| | 250 | 99.91 | 87.95 | 99.94 | 87.35 |
| | 500 | 99.95 | 87.35 | 99.91 | 87.95 |
| | 750 | 99.90 | 88.55 | 99.86 | 89.16 |
| | 1000 | 99.99 | 92.17 | 99.88 | 90.36 |
| | 1250 | 99.86 | 89.76 | 99.89 | 90.36 |
| | 1500 | 99.91 | 90.96 | 99.89 | 91.57 |
| **ENZYMES** | 0 | 100.0 | 46.67 | 100.0 | 52.22 |
| | 100 | 100.0 | 73.33 | 100.0 | 54.44 |
| | 250 | 99.94 | 78.89 | 100.0 | 56.67 |
| | 500 | 99.74 | 74.44 | 100.0 | 57.78 |
| | 750 | 99.82 | 78.89 | 100.0 | 58.89 |
| | 1000 | 99.95 | 73.33 | 100.0 | 56.67 |
| | 1250 | 99.74 | 82.22 | 100.0 | 55.56 |
| | 1500 | 99.83 | 81.11 | 100.0 | 57.78 |
| **NCI1** | 0 | 99.71 | 65.75 | 99.77 | 84.25 |
| | 100 | 98.84 | 81.49 | 99.80 | 84.90 |
| | 250 | 97.47 | 84.74 | 99.57 | 86.85 |
| | 500 | 96.84 | 87.34 | 99.41 | 89.77 |
| | 750 | 97.10 | 89.61 | * | * |
| | 1000 | 96.22 | 89.12 | * | * |
| | 1250 | 95.05 | 89.77 | * | * |
| | 1500 | 96.22 | 90.75 | * | * |

## A.3 Ablation Results

This subsection contains the results of the ablation studies discussed in Section 4.2.3 and documented in Tab. A.3, A.4 and A.5 respectively.

**Table A.3:** Train and Test Accuracies for Deep Learning-Based Augmentation Strategies Ablation on PROTEINS Dataset

| | | Percentage of Added Synthetic Data | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **CLS-DS on:** | | **0%** | **100%** | **500%** | **1000%** | **2500%** | **5000%** | **7500%** |
| **PROTEINSrandom** | Train Acc. | 100.00 | 99.79 | 99.91 | 99.78 | 99.95 | 99.98 | 99.96 |
| | Test Acc. | 73.49 | 85.24 | 93.17 | 94.80 | 96.78 | 97.76 | 98.30 |
| **PROTEINSlinear** | Train Acc. | 100.00 | 99.95 | 99.96 | 99.90 | 99.99 | 99.99 | 99.99 |
| | Test Acc. | 70.48 | 84.64 | 95.78 | 97.54 | 98.80 | 99.07 | 99.27 |
| **PROTEINSslerp** | Train Acc. | 100.00 | 99.74 | 99.84 | 99.77 | 99.82 | 99.77 | 99.74 |
| | Test Acc. | 70.48 | 86.14 | 94.48 | 96.50 | 97.29 | 98.49 | 98.58 |
| **PROTEINSslerp2** | Train Acc. | 99.89 | 99.95 | 99.96 | 99.96 | 99.89 | 99.90 | 99.83 |
| | Test Acc. | 71.69 | 91.57 | 96.59 | 97.37 | 98.77 | 99.14 | 99.33 |
| **PROTEINSrandom** | Train Acc. | 100.00 | 99.79 | 99.91 | 99.78 | 99.95 | 99.98 | 99.96 |
| **(original test data)** | Test Acc. | 73.49 | 78.92 | 83.13 | 87.35 | 89.16 | 91.57 | 92.17 |
| **PROTEINSlinear** | Train Acc. | 100.00 | 99.95 | 99.96 | 99.90 | 99.99 | 99.99 | 99.99 |
| **(original test data)** | Test Acc. | 70.48 | 72.89 | 84.94 | 87.95 | 90.36 | 89.16 | 90.96 |
| **PROTEINSslerp** | Train Acc. | 100.00 | 99.74 | 99.84 | 99.77 | 99.82 | 99.77 | 99.74 |
| **(original test data)** | Test Acc. | 70.48 | 81.33 | 89.16 | 90.36 | 90.96 | 90.96 | 89.16 |
| **PROTEINSslerp2** | Train Acc. | 99.89 | 99.95 | 99.96 | 99.96 | 99.89 | 99.90 | 99.83 |
| **(original test data)** | Test Acc. | 71.69 | 85.54 | 89.16 | 90.96 | 91.57 | 91.57 | 92.77 |

Table A.3 contains the results of the Ablation Study on the different interpolation strategies for latent space data augmentation. As previous in this work we introduce different percentages of added synthetic data to the original dataset and evaluate how the downstream classifier performed on different types of augmented data. This means the strategy for the latent space interpolation was the only changing factor in this evaluation as described in Section 4.2.3.

**Table A.4:** Train and Test Accuracies for Deep Learning-Based Node Labels Ablation on PROTEINS Dataset

| | | Percentage of Added Synthetic Data | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **0%** | **100%** | **250%** | **500%** | **750%** | **1000%** | **12500%** | **1500%** |
| **CLS-DS-nonodes** | Train Acc. | 99.68 | 99.84 | 99.88 | 99.84 | 99.88 | 99.82 | 99.81 | 99.85 |
| | Test Acc. | 67.47 | 89.46 | 93.28 | 96.79 | 97.59 | 97.59 | 97.77 | 97.89 |
| **CLS-DS-nonodes** | Train Acc. | 99.68 | 99.84 | 99.88 | 99.84 | 99.88 | 99.82 | 99.81 | 99.85 |
| **(original test data)** | Test Acc. | 67.47 | 82.53 | 86.14 | 92.17 | 90.96 | 89.16 | 88.55 | 90.96 |
| **CLS-DS** | Train Acc. | 100.0 | 99.89 | 99.94 | 99.89 | 99.90 | 99.96 | 99.97 | 99.97 |
| | Test Acc. | 72.89 | 90.66 | 95.00 | 97.69 | 97.30 | 98.03 | 98.12 | 98.42 |
| **CLS-DS** | Train Acc. | 100.0 | 99.89 | 99.94 | 99.89 | 99.90 | 99.96 | 99.97 | 99.97 |
| **(original test data)** | Test Acc. | 72.89 | 83.73 | 87.95 | 90.36 | 89.16 | 89.76 | 89.16 | 89.76 |

Table A.4 documents the results of the node label ablation where the *nonodes* dataset was built from a VAE model which did not use node labels when decoding latent space vectors as well as a downstream classifier which ignored any node labels (if present) entirely in the prediction process. The *(original test data)* results mark the evaluation on only the original data test set while the other results take the entire test partition of the extended dataset as the test set.

**Table A.5:** Train and Test Accuracies for Deep Learning–Based $\beta$-VAE Ablation on PROTEINS Variants

| | | **Percentage of Added Synthetic Data** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **0%** | **100%** | **250%** | **500%** | **750%** | **1000%** | **1250%** | **1500%** |
| **CLS-DS on PROTEINS** | Train Acc. | 100.00 | 99.89 | 99.94 | 99.89 | 99.90 | 99.96 | 99.97 | 99.97 |
| | Test Acc. | 72.89 | 90.66 | 95.00 | 97.69 | 97.30 | 98.03 | 98.12 | 98.42 |
| **CLS-DS on PROTEINS (original test data)** | Train Acc. | 100.00 | 99.89 | 99.94 | 99.89 | 99.90 | 99.96 | 99.97 | 99.97 |
| | Test Acc. | 72.89 | 83.73 | 87.95 | 90.36 | 89.16 | 89.76 | 89.16 | 89.76 |
| **CLS-DS-2Beta on PROTEINS2Beta** | Train Acc. | 99.89 | 99.89 | 99.94 | 99.95 | 99.80 | 99.94 | 99.94 | 99.87 |
| | Test Acc. | 73.49 | 88.55 | 93.45 | 95.88 | 97.30 | 97.54 | 98.30 | 98.12 |
| **CLS-DS on PROTEINS2Beta (original test data)** | Train Acc. | 99.89 | 99.89 | 99.94 | 99.95 | 99.80 | 99.94 | 99.94 | 99.87 |
| | Test Acc. | 73.49 | 80.72 | 83.73 | 87.35 | 87.35 | 90.36 | 93.98 | 88.55 |
| **CLS-DS-NoBeta on PROTEINSNoBeta** | Train Acc. | 99.79 | 100.00 | 99.94 | 99.93 | 99.90 | 99.89 | 99.92 | 99.95 |
| | Test Acc. | 74.70 | 90.36 | 94.48 | 96.99 | 97.45 | 97.26 | 98.04 | 98.31 |
| **CLS-DS-NoBeta on PROTEINSNoBeta (original test data)** | Train Acc. | 99.79 | 100.00 | 99.94 | 99.93 | 99.90 | 99.89 | 99.92 | 99.95 |
| | Test Acc. | 74.70 | 83.73 | 87.35 | 91.57 | 89.16 | 90.36 | 90.36 | 92.17 |

Table A.5 documents the results of the $\beta$-VAE ablation study. The synthetic dataset created originated from different VAE decoders. Each VAE was trained with a different value for $\beta$ in the loss term. Parallel to the approach above the *(original test data)* results mark the evaluation on only the original data test set while the other results take the entire test partition of the extended dataset as the test set. This Ablation can be found in Section 4.2.3.

# B

# Additional Implementation Details

This chapter provides extended visualizations of the Variational Autoencoder (Section B.1) and pseudo code outlining the implementation of key components (Section B.2). These supplementary elements aim to offer deeper insights and enhance the reproducibility and transparency of the presented work.

## B.1 Extended Visualization of VAE

The following section contains a extended visualization of the VAE model architecture as introduced in Section 3.2.

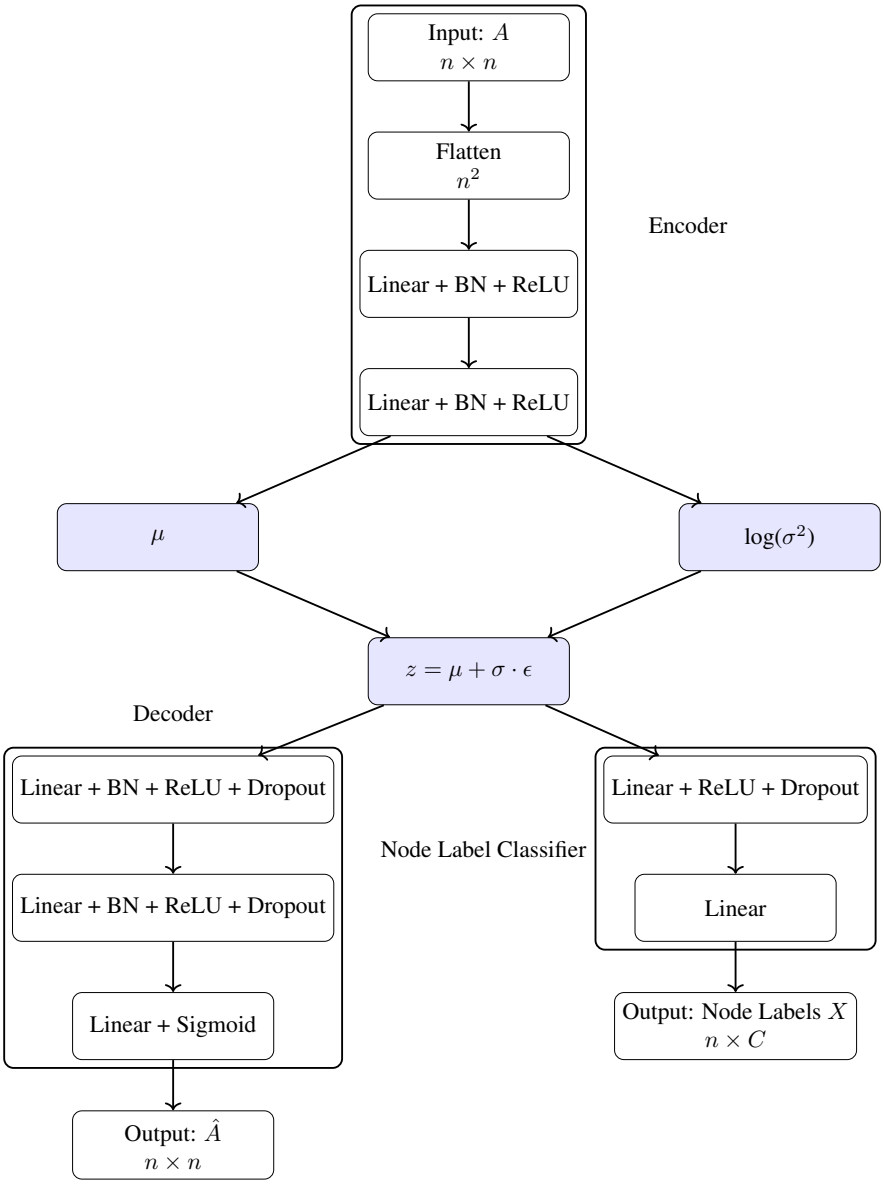


Figure B.1: Visualization of Message Passing in our Version of the VAE

Figure B.1 contains a visualization of the model architecture used for the VAE in VAGUE. The model is based on a simple encoder/decoder architecture with an auxiliary node label classifier to predict the node labels.

## B.2 Pseudo Code of Major Parts

The following section contains the code of the three main parts of VAGUE as introduced in Section 3.1.

Starting with Alg. 1 highlighting the training of the first step in the VAGUE pipeline (Section 3.2). Followed by Alg. 2 documenting the data augmentation in latent space using slerp as detailed in Section 3.3. Finishing with Alg. 3 capturing the standard training approach of the deep learning-based downstream models used for the dataset evaluation as introduced in Section 3.4.

**Algorithm 1** Training loop for one epoch in VAE training

```
1:  function TRAIN_ONE_EPOCH(model, dataloader, optimizer, loss_fn, epoch, total_epoch)
2:      model.eval()
3:      total_loss ← 0
4:      β ← min(1.0, 0.1 + 0.9 · (epoch−1)/total_epochs)
5:      α ← β
6:      for each (data_batch, node_labels_batch) in dataloader do
7:          Move data to device
8:          optimizer.zero_grad()
9:          (recon_adj, label_pred, μ, logvar) ← model(data_batch)
10:         loss ← loss_fn(recon_adj, data_batch, μ, logvar, label_pred, node_labels_batch, α, β)
11:         loss.backward()
12:         Clip gradients
13:         optimizer.step()
14:         total_loss ← total_loss + loss
15:     return total_loss / dataset_size
```

Algorithm 1 contains minor amendments to the standard training loop of any deep learning model. Lines 4 and 5 highlight the implementation of the process to anneal both $\beta$ as well as $\alpha$ parameters of the loss function. Additionially line 12 notes the use of gradient clipping and lines 3, 14 and 15 allow for average loss reporting over the entire dataset.

Algorithm 2 encapsulates the process of sampling the latent space between two samples in a batched manner using slerp as described in Section 3.3. For this we encode original datasamples to get their latent representation which is processed in lines 9 and 10. Lines 11, 12 and 13 allow for pair wise distinct interpolations by setting up a upper triangular matrix with a 0 diagnoal. In line 14 through 18 we then interpolate between two latent vectors and get a new vector in latent space. This new latent vector is then decoded to a graph and added to the synthetic dataset in lines 19 through 23.

The training of the downstream described in Alg. 3 is straight forward and does not deviate from the standard model training process. The model used is described in Section 3.4 and the results can be found in Chapter 4.

**Algorithm 2** Create New Samples via SLERP

**function** CREATENEWSAMPLESSLERP2(vae, encoder_data, data, device, batchsize, threshhold ← 0.5)
    data_new ← CustomDataset()
    vae.eval()
    dataloader ← DataLoader(data, batch_size = int(batchsize), shuffle = True, drop_last = True)
    **with** no gradient:
    **for** each (batch, node_labels, graph_labels) in dataloader **do**
        Move data to device
        $x \leftarrow$ reshape(batch, $[-1, n \times n]$)
        ($\mu$, logvar) ← vae.encode($x$)
        $z \leftarrow$ vae.reparameterize($\mu$, logvar)
        ($i\_idx$, $j\_idx$) ← torch.triu_indices($n$, $n$, offset $= 1$)
        $z\_i\_all \leftarrow z[i\_idx]$
        $z\_j\_all \leftarrow z[j\_idx]$
        **for** each $t$ in linspace(0, 1, 4, endpoint = False) **do**
            **if** $t == 0$ **then**
                **continue** *//Avoids adding original datapoints to synthetic dataset*
            **else**
                $z\_interp \leftarrow$ slerp($t$, $z\_i\_all$, $z\_j\_all$)
            recon_adj ← vae.decode($z\_interp$)
            node_labels ← vae.classifier($z\_interp$).view($-1$, $n$, num_nlabels).argmax(dim = $-1$)
            recon_adj ← (recon_adj.squeeze() $\geq$ threshhold).float()
            graph_labels ← tensor of ones with shape (recon_adj.size(0)) $\times$ encoder_data["class_nr"]
            data_new.add_entry(recon_adj.cpu(), node_labels.cpu(), graph_labels)
    **return** data_new

**Algorithm 3** Training loop for one epoch in classifier training

**function** TRAINONEEPOCH(cls_model, optimizer, dataloader, loss_fn)
    cls_model.train()
    total_loss ← 0
    **for** each (adj_matrices, node_labels, graph_label) in dataloader **do**
        Move data to device
        adj_matrices ← reshape(adj_matrices, [adj_matrices.shape[0], -1])
        optimizer.zero_grad()
        pred ← cls_model(adj_matrices, node_labels)
        loss ← loss_fn(pred, graph_label)
        loss.backward()
        optimizer.step()
        total_loss ← total_loss + loss.item()
    **return** total_loss / len(dataloader.dataset)

# Bibliography


[1] D. CONTE, P. FOGGIA, C. SANSONE, and M. VENTO. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18 (03):265–298, 2004. doi: 10.1142/S0218001404003228. URL https://doi.org/10.1142/S0218001404003228.

[2] H Bunke and G Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245–253, 1983. ISSN 0167-8655. doi: https://doi.org/10.1016/0167-8655(83)90033-8. URL https://www.sciencedirect.com/science/article/pii/0167865583900338.

[3] Nils M. Kriege, Fredrik D. Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5(6), 2020. doi: 10.1007/s41109-019-0195-3. URL https://doi.org/10.1007/s41109-019-0195-3.

[4] Kaize Ding, Zhe Xu, Hanghang Tong, and Huan Liu. Data augmentation for deep graph learning: A survey, 2022. URL https://arxiv.org/abs/2202.08235.

[5] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019. ISSN 1935-8245. doi: 10.1561/2200000056. URL http://dx.doi.org/10.1561/2200000056.

[6] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. URL https://arxiv.org/abs/1406.2661.

[7] Clément Chadebec and Stéphanie Allassonnière. Data augmentation with variational autoencoders and manifold sampling, 2021. URL https://arxiv.org/abs/2103.13751.

[8] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019. ISSN 2196-1115. doi: 10.1186/s40537-019-0197-0. URL https://doi.org/10.1186/s40537-019-0197-0.

[9] Markus Bayer, Marc-André Kaufhold, and Christian Reuter. A survey on data augmentation for text classification. *ACM Comput. Surv.*, 55(7), December 2022. ISSN 0360-0300. doi: 10.1145/3544558. URL https://doi.org/10.1145/3544558.

[10] xAI. Grok: An ai assistant for understanding the universe. https://x.ai/grok, 2023. Accessed: 2025-05-22.

[11] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao

Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

[12] Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018. URL https://api.semanticscholar.org/CorpusID:49313245.

[13] Ranjan Sapkota, Shaina Raza, Maged Shoman, Achyut Paudel, and Manoj Karkee. Multimodal large language models for image, text, and speech data augmentation: A survey, 2025. URL https://arxiv.org/abs/2501.18648.

[14] Kaize Ding, Zhe Xu, Hanghang Tong, and Huan Liu. Data augmentation for deep graph learning: A survey. *SIGKDD Explor. Newsl.*, 24(2):61–77, December 2022. ISSN 1931-0145. doi: 10.1145/3575637.3575646. URL https://doi.org/10.1145/3575637.3575646.

[15] Shuaishuai Zu, Chuyu Wang, Yafei Liu, Jun Shen, and Li Li. Contrastive learning augmented graph auto-encoder. In Biao Luo, Long Cheng, Zheng-Guang Wu, Hongyi Li, and Chaojie Li, editors, *Neural Information Processing*, pages 280–291, Singapore, 2024. Springer Nature Singapore. ISBN 978-981-99-8145-8.

[16] Tengfei Ma, Jie Chen, and Cao Xiao. Constrained generation of semantically valid graphs via regularizing variational autoencoders, 2018. URL https://arxiv.org/abs/1809.02630.

[17] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders, 2018. URL https://arxiv.org/abs/1802.03480.

[18] Tala Talaei Khoei, Hadjar Ould Slimane, and Naima Kaabouch. Deep learning: systematic review, models, challenges, and research directions. *Neural Computing and Applications*, 35:23103–23124, 2023. doi: 10.1007/s00521-023-08957-4. URL https://link.springer.com/article/10.1007/s00521-023-08957-4.

[19] Wilson L. Taylor. "cloze procedure": A new tool for measuring readability. *Journalism Quarterly*, 30 (4):415–433, 1953. doi: 10.1177/107769905303000401. URL https://doi.org/10.1177/107769905303000401.

[20] Mohd Javaid, Abid Haleem, Ravi Pratap Singh, and Mumtaz Ahmed. Computer vision to enhance healthcare domain: An overview of features, implementation, and opportunities. *Intelligent Pharmacy*, 2(6):792–803, 2024. ISSN 2949-866X. doi: https://doi.org/10.1016/j.ipha.2024.05.007. URL https://www.sciencedirect.com/science/article/pii/S2949866X24000662.

[21] Hironobu Fujiyoshi, Tsubasa Hirakawa, and Takayoshi Yamashita. Deep learning-based image recognition for autonomous driving. *IATSS Research*, 43(4):244–252, 2019. ISSN 0386-1112. doi: https://doi.org/10.1016/j.iatssr.2019.11.008. URL https://www.sciencedirect.com/science/article/pii/S0386111219301566.

[22] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015. URL https://arxiv.org/abs/1511.08458.

[23] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[24] Nivedita Sethiya and Chandresh Kumar Maurya. End-to-end speech-to-text translation: A survey. *Computer Speech and Language*, 90:101751, 2025. ISSN 0885-2308. doi: https://doi.org/10.1016/j.csl.2024.101751. URL https://www.sciencedirect.com/science/article/pii/S0885230824001347.

[25] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision, 2022. URL https://arxiv.org/abs/2212.04356.

[26] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin, 2015. URL https://arxiv.org/abs/1512.02595.

[27] Kepan Nan, Rui Xie, Penghao Zhou, Tiehan Fan, Zhenheng Yang, Zhijie Chen, Xiang Li, Jian Yang, and Ying Tai. Openvid-1m: A large-scale high-quality dataset for text-to-video generation, 2025. URL https://arxiv.org/abs/2407.02371.

[28] Shumin Xiao, Shichao Wang, Yuchao Dai, et al. Graph neural networks in node classification: survey and evaluation. *Machine Vision and Applications*, 33(4), 2022. doi: 10.1007/s00138-021-01251-0. URL https://doi.org/10.1007/s00138-021-01251-0.

[29] Xingyu Liu, Juan Chen, and Quan Wen. A survey on graph classification and link prediction based on gnn, 2023. URL https://arxiv.org/abs/2307.00865.

[30] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification, 2022.

[31] Junchi Yan, Xu-Cheng Yin, Weiyao Lin, Cheng Deng, Hongyuan Zha, and Xiaokang Yang. A short survey of recent advances in graph matching. In *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*, ICMR '16, page 167–174, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450343596. doi: 10.1145/2911996.2912035. URL https://doi.org/10.1145/2911996.2912035.

[32] Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007. ISSN 1574-0137. doi: https://doi.org/10.1016/j.cosrev.2007.05.001. URL https://www.sciencedirect.com/science/article/pii/S1574013707000020.

[33] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021. doi: 10.1109/TNNLS.2020.2978386.

[34] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2018.03.022. URL https://www.sciencedirect.com/science/article/pii/S0950705118301540.

[35] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '14, page 701–710. ACM, August 2014. doi: 10.1145/2623330.2623732. URL http://dx.doi.org/10.1145/2623330.2623732.

[36] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks, 2016. URL https://arxiv.org/abs/1607.00653.

[37] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017. URL https://arxiv.org/abs/1609.02907.

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

[39] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(77):2539–2561, 2011. URL http://jmlr.org/papers/v12/shervashidze11a.html.

[40] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995. ISSN 1573-0565. doi: 10.1007/BF00994018. URL https://doi.org/10.1007/BF00994018.

[41] Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, November 2009. ISSN 1095-7200. doi: 10.1137/070710111. URL http://dx.doi.org/10.1137/070710111.

[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*, page 318–362. MIT Press, Cambridge, MA, USA, 1986. ISBN 026268053X.

[43] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022. URL https://arxiv.org/abs/1312.6114.

[44] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Sy2fzU9gl.

[45] Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016. URL https://arxiv.org/abs/1611.07308.

[46] Zhaoliang Chen, Zhihao Wu, Ylli Sadikaj, Claudia Plant, Hong-Ning Dai, Shiping Wang, Yiu-Ming Cheung, and Wenzhong Guo. Adedgedrop: Adversarial edge dropping for robust graph neural networks, 2024. URL https://arxiv.org/abs/2403.09171.

[47] Shujuan Wang, Yajing Dai, Jie Shen, and Jianhua Xuan. Research on expansion and classification of imbalanced data based on smote algorithm. *Scientific Reports*, 11(1):24039, Dec 2021. doi: 10.1038/s41598-021-03430-5. URL https://doi.org/10.1038/s41598-021-03430-5.

[48] Ken Shoemake. Animating rotation with quaternion curves. *SIGGRAPH Comput. Graph.*, 19(3): 245–254, July 1985. ISSN 0097-8930. doi: 10.1145/325165.325242. URL https://doi.org/10.1145/325165.325242.

[49] Lu Mi, Tianxing He, Core Francisco Park, Hao Wang, Yue Wang, and Nir Shavit. Revisiting latent-space interpolation via a quantitative evaluation framework, 2021. URL https://arxiv.org/abs/2110.06421.

[50] Tong Zhao, Xianfeng Tang, Danqing Zhang, Haoming Jiang, Nikhil Rao, Yiwei Song, Pallav Agrawal, Karthik Subbian, Bing Yin, and Meng Jiang. Autogda: Automated graph data augmentation for node classification. In *Learning on Graphs Conference*, pages 32–1. PMLR, 2022.

[51] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. URL www.graphlearning.io.

[52] Asim Kumar Debnath, Rosa L. Lopez de Compadre, Gargi Debnath, Alan J. Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of Medicinal Chemistry*, 34(2):786–797, 1991. doi: 10.1021/jm00106a046. URL https://doi.org/10.1021/jm00106a046.

[53] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, 2003. doi: 10.1016/s0022-2836(03)00628-4.

[54] Nikil Wale, Ian Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowl. Inf. Syst.*, 14:347–375, 03 2008. doi: 10.1109/ICDM.2006.39.

[55] Ida Schomburg, Antje Jäde, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. Brenda, the enzyme database: Updates and major new developments. *Nucleic acids research*, 32:D431–3, 01 2004. doi: 10.1093/nar/gkh081.

[56] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1):37–52, 1987. ISSN 0169-7439. doi: https://doi.org/10.1016/

0169-7439(87)80084-9. URL https://www.sciencedirect.com/science/article/pii/0169743987800849. Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists.