Software Requirements Specification for Re-ProtGNN: Re-implementation of the ProtGNN model

Yuanqi Xue

April 13, 2025

Contents

Revision History

Date	Version	Notes
Feb 6, 2024	1.0	First Draft

1 Reference Material

This section records information for easy reference.

1.1 Table of Units

Not applicable, as there is no unit in this software.

1.2 Table of Symbols

The table that follows summarizes the symbols used in this document along with their units. The choice of symbols was made to be consistent with the heat transfer literature and with existing documentation for solar water heating systems. The symbols are listed in alphabetical order.

symbol	unit	description
X	Unitless	Node feature matrix
X_p	Unitless	Prototype node feature matrix
A	Unitless	Adjacency matrix
A_p	Unitless	Prototype adjacency matrix
y	Unitless	True class label
$\hat{y}$	Unitless	Predicted class label
W	Unitless	Weight matrix for the fully connected layer
b	Unitless	Bias term
W_g	Unitless	Weight matrix for GNN encoder
P	Unitless	Prototype representations
h	Unitless	Graph embedding
σ	Unitless	Sigmoid activation function
$\mathcal{L}$	Unitless	Loss function
L_{CE}	Unitless	Cross-entropy loss
L_{Clst}	Unitless	Cluster loss
L_{Sep}	Unitless	Separation loss
L_{Div}	Unitless	Diversity loss
η	Unitless	Learning rate
$\lambda_1, \lambda_2, \lambda_3$	Unitless	Regularization parameters
h_v^{k+1}	Unitless	Node representation at layer $k+1$
h_u^k	Unitless	Representation vector of node u at layer k
$\tilde{A}$	Unitless	Normalized adjacency matrix

$\hat{A}$	Unitless	Adjacency matrix with self-connections
$\hat{D}$	Unitless	Diagonal degree matrix where $\hat{D}_{ii} = \sum_j \hat{A}_{ij}$
W^k	Unitless	Trainable weight matrix at layer k
$\mathcal{N}(v)$	Unitless	Neighborhood of node v, i.e., set of adjacent nodes
$s_{\max}$	Unitless	Cosine similarity threshold for prototype diversity loss
C	Unitless	Number of classes in the classification task

1.3 Abbreviations and Acronyms

symbol	description
A	Assumption
DD	Data Definition
GD	General Definition
GS	Goal Statement
IM	Instance Model
LC	Likely Change
PS	Physical System Description
R	Requirement
SRS	Software Requirements Specification
TM	Theoretical Model
ProtGNN	Prototype-based Graph Neural Network
Re-ProtGNN	Re-implementation of the ProtGNN model
GNN	Graph Neural Network
GCN	Graph Convolutional Network
GIN	Graph Isomorphism Network

1.4 Mathematical Notation

Notation	Description	Source
num_rows($\cdot$)	Number of rows in a matrix	-
num_cols($\cdot$)	Number of columns in a matrix	-
dim($\cdot$)	Dimension of a matrix	-
$M[i, j]$	Element at row i, column j in matrix M	-

$f(\cdot)$	A function representing the graph encoder, as defined in TM4.3.3	-
$\cos(\cdot, \cdot)$	Cosine similarity function	(ScienceDirect, 2024)
$\text{ReLU}(\cdot)$	Outputs input if positive; otherwise, returns zero (i.e., $\max(0, x)$)	(Wikipedia contributors, 2024f)
∇	Gradient operator	(Wikipedia contributors, 2024c)

2 Introduction

Graph Neural Networks (GNNs) have shown strong performance in node classification, graph classification, and link prediction tasks. However, their black-box nature makes it difficult to understand their decision-making process, limiting their usage in critical areas such as medical diagnosis. For instance, in the MUTAG classification task, GNNs are used to classify molecular graphs based on their mutagenic properties, but the lack of interpretability in the model's decision-making process can hinder its deployment in fields requiring clear explanations, such as drug development and toxicity prediction.

To address this, the paper *ProtGNN: Towards Self-Explaining Graph Neural Networks* (Zhang et al., 2022) proposes ProtGNN (Prototype-based Graph Neural Network), an interpretable GNN model designed to improve explainability by learning representative prototypes for each class. The goal of this project is to re-implement ProtGNN and validate its results. Further details on the problem and motivation behind this project can be found in the Problem Statement document (Xue, 2025).

The introduction section outlines the Software Requirement Specifications (SRS) for Re-ProtGNN (Re-implementation of the ProtGNN model), detailing its purpose, scope, target audience, and organization.

2.1 Purpose of Document

The purpose of this SRS document is to define and communicate the requirements, constraints, and models for the Re-ProtGNN project. It serves as a formal reference and links to the other documents of this project.

2.2 Scope of Requirements

Re-ProtGNN is a re-implementation of the existing ProtGNN model, aimed at reproducing its results and validating its performance on graph datasets. The model operates in a supervised setting, performing graph classification while incorporating built-in interpretability.

The model will rely solely on the provided graph structure and features to perform classification tasks. External factors, such as domain-specific knowledge (e.g., the biological context of protein interactions, which may require expert understanding and manual tuning of input), are not considered in this implementation.

2.3 Characteristics of Intended Reader

This document is intended for machine learning researchers, AI practitioners, and students who are interested in interpretable GNNs and prototype-based learning. To effectively understand and utilize this document, the reader should have:

- College-level introductory knowledge of calculus, particularly partial derivatives and gradient-based optimization.
- Completion of an undergrad-level course or equivalent experience in supervised learning (e.g., logistic regression) and neural networks (including backpropagation, loss functions, and optimization).
- Familiarity with graph-based learning (e.g., message-passing mechanisms in GNNs) is recommended but not required.
- Basic knowledge of model explainability in AI, including the concept of prototypes in classification models.

This document is designed to be accessible to readers with the above qualifications. However, researchers with limited exposure to graph neural networks can still benefit from the explanations provided, as long as they have a strong foundation in deep learning and optimization methods.

2.4 Organization of Document

This SRS document is based on the template by (Smith and Lai, 2005; Smith et al., 2007). It systematically presents the system's objectives, theoretical foundations, assumptions, and models. For a structured reading approach, readers may start with the goal statements (Section 4.1.3), followed by the theoretical models (Section 4.3.3) and general definitions (Section 4.3.4), which build upon these goals. The document concludes with instance models (Section 4.3.7), providing a concrete understanding of the system's actual implementation.

3 General System Description

This section provides general information about the system. It identifies the interfaces between the system and its environment, describes the user characteristics and lists the system constraints.

3.1 System Context

- User Responsibilities:
  - Download the MUTAG dataset and load it to Re-ProtGNN.

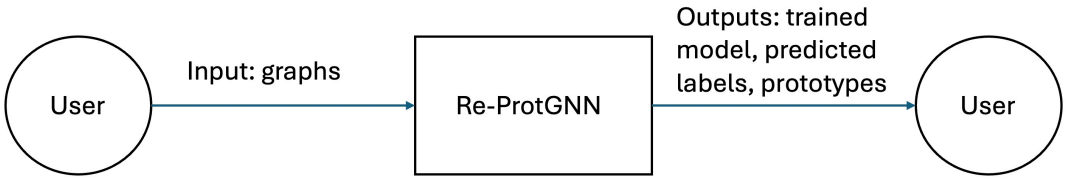

Figure 1: System Context

  - Run Re-ProtGNN on a system with appropriate computational resources.
  - Interpret the model's output, including classifications and prototype-based explanations.

- Re-ProtGNN Responsibilities:
  - Train a GNN model using gradient-based optimization.
  - Provide classification predictions.
  - Learn prototype representations.
  - Offer visualization tools for understanding learned prototypes.

3.2 User Characteristics

The end user of Re-ProtGNN includes anyone working with GNNs or those interested in interpretable deep learning models. While prior experience with interpretable machine learning is beneficial, it is not required. The basic skill set needed to use Re-ProtGNN includes basic computer literacy and a high school-level understanding of graphs.

3.3 System Constraints

There are no system constraints for this software.

4 Specific System Description

This section first presents the problem description, which gives a high-level view of the problem to be solved. This is followed by the solution characteristics specification, which presents the assumptions, theories, definitions and finally the instance models.

4.1 Problem Description

Traditional GNNs excel in graph classification tasks but function as black-box models, making it difficult for users to understand their classification decisions. ProtGNN addresses this lack of interpretability by incorporating prototype-based explanations into graph classification. This project, Re-ProtGNN, is a re-implementation of ProtGNN, aimed at validating its reproducibility and performance on benchmark datasets.

4.1.1 Terminology and Definitions

This subsection provides a list of terms that are used in the subsequent sections and their meaning, with the purpose of reducing ambiguity and making it easier to correctly understand the requirements:

- Graph: A structure consisting of nodes and edges, typically represented by a node feature matrix and an adjacency matrix.
- Graph Classification: The task of assigning a label to an entire graph based on its structure and node attributes.
- Graph Neural Networks: A type of deep learning model designed to process graph-structured data by aggregating and transforming node information.
- Prototype: A representative subgraph that highlights the key structural patterns influencing classification decisions.
- MUTAG Dataset: A benchmark dataset for molecular graph classification, distinguishing between mutagenic and non-mutagenic compounds (Debnath et al., 1991).
- Supervised Learning: A training approach where the model learns from labeled data, mapping input graphs to their corresponding class labels.

4.1.2 Physical System Description

The physical system of Re-ProtGNN, as shown in Figure 2, includes the following elements:

PS1: Graph Inputs: The dataset consists of labeled graphs used for graph classification (e.g., MUTAG).

PS2: GNN Encoder: A GNN-based encoder (e.g., GCN, GIN) extracts node and graph-level embeddings through message passing. Graph embeddings are used for both classification and prototype learning.

PS3: Prototype Layer: A prototype layer learns representative subgraph embeddings that characterize structural patterns of different classes. Prototypes are updated during training to match key graph structures associated with class labels.

PS4: Fully Connected Layer: The model classifies graphs based on their learned representations. The most similar prototype is retrieved to justify the classification result.

PS5: Training Process: The model is trained using supervised learning, optimizing a combined cross-entropy loss and prototype-based losses.

PS6: Inference Process: The trained model classifies new graphs and provides prototype-based explanations. Prototype similarity scores are used to explain classification decisions.

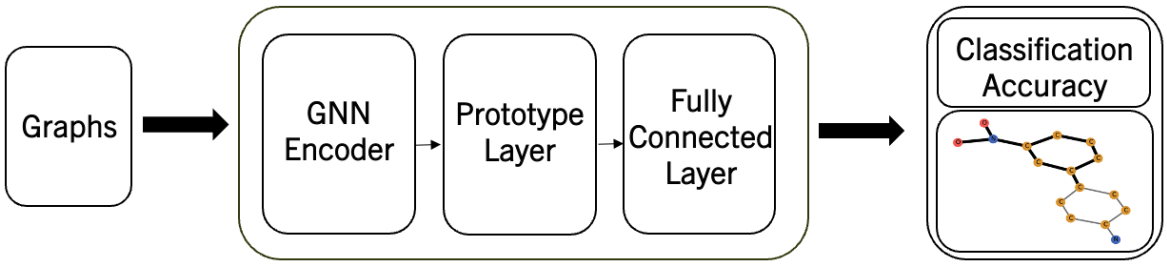

Figure 2: The ProtGNN Framework

4.1.3 Goal Statements

Given the input graphs, the goal statement is:

GS1: Output a trained model that classifies the input graphs and identifies a few prototypes (i.e., key graph structures that contribute to classification) for each class.

4.2 Solution Characteristics Specification

The instance models that govern Re-ProtGNN are presented in Subsection 4.3.7. The information to understand the meaning of the instance models and their derivation is also presented, so that the instance models can be verified.

4.2.1 Types

symbol	description
$\mathbb{Z}$	Integer
$\mathbb{R}$	Real number

4.3 Scope Decisions

Re-ProtGNN focuses on validating the classification accuracy and prototype quality of the ProtGNN model using the MUTAG dataset (Debnath et al., 1991).

4.3.1 Modelling Decisions

Re-ProtGNN follows the same model design and has the same Loss Function (see IM4.3.3) as ProtGNN.

4.3.2 Assumptions

This section simplifies the original problem and helps in developing the theoretical model by filling in the missing information for the physical system. The numbers given in the square brackets refer to the theoretical model [TM], general definition [GD], data definition [DD], instance model [IM], or likely change [LC], in which the respective assumption is used.

A1: The node features of input graphs are relevant to classification, as required by TM4.3.3 to produce meaningful graph embeddings.

A2: The structural patterns of input graphs are relevant to classification, as required by TM4.3.3 to produce meaningful graph embeddings.

A3: For the training phase IM1 to produce an accurate model and the inference phase IM2 to generate reasonable predictions and prototypes, each input graph must belong to one of the two predefined classes (i.e., mutagenic or non-mutagenic) as defined in the MUTAG dataset (Debnath et al., 1991).

4.3.3 Theoretical Models

This section focuses on the general equations and laws that Re-ProtGNN is based on.

RefName: **TM1:GNNFP**

Label: Graph Neural Network (GNN) Forward Pass

Equation: $h_v^{k+1} = ReLU\left(\sum_{u \in \mathcal{N}(v)} W^k h_u^k \tilde{A}_{uv}\right),$

Description: The above equation describes the message-passing paradigm of Graph Convolutional Networks (GCNs), where the representation of each node v is iteratively updated by aggregating representations from its neighboring nodes $\mathcal{N}(v)$, using a trainable weight matrix W^k and an activation function $ReLU(\cdot)$.

- h_u^k: Representation vector of node u at the k-th layer. (Type: vector in $\mathbb{R}^{d_k}$)
- $\tilde{A} = \hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$: Normalized adjacency matrix. (Type: matrix in $\mathbb{R}^{N \times N}$)
- $\hat{A} = A + I$: Adjacency matrix of graph G with self-connections. (Type: matrix in $\mathbb{R}^{N \times N}$)
- $\hat{D}$: Diagonal degree matrix where $\hat{D}_{ii} = \sum_j \hat{A}_{ij}$. (Type: matrix in $\mathbb{R}^{N \times N}$)
- W^k: Trainable weight matrix at the k-th layer. (Type: matrix in $\mathbb{R}^{d_{k+1} \times d_k}$)
- $ReLU(\cdot)$: Rectified Linear Unit activation function. (Input: vector in $\mathbb{R}^{d_{k+1}}$, Output: vector in $\mathbb{R}^{d_{k+1}}$), applied element-wise as $\max(0, x_i)$.
- h_v^{k+1}: Updated representation of node v at the $(k+1)$-th layer. (Type: vector in $\mathbb{R}^{d_{k+1}}$)

Notes: The dimensional consistency of the equation is preserved as follows:

- $h_u^k \in \mathbb{R}^{d_k}$: the feature vector of node u at layer k.
- $W^k \in \mathbb{R}^{d_{k+1} \times d_k}$: when multiplied with h_u^k, gives: $W^k h_u^k \in \mathbb{R}^{d_{k+1}}$
- $\tilde{A}_{uv} \in \mathbb{R}$: a scalar value from the normalized adjacency matrix. Multiplying this scalar does not change the vector's shape: $W^k h_u^k \tilde{A}_{uv} \in \mathbb{R}^{d_{k+1}}$
- Summing over all $u \in \mathcal{N}(v)$ (neighbors of node v) adds together multiple vectors of shape $\mathbb{R}^{d_{k+1}}$. The result is also a vector of the same shape: $\sum_{u \in \mathcal{N}(v)} W^k h_u^k \tilde{A}_{uv} \in \mathbb{R}^{d_{k+1}}$
- Finally, the ReLU function is applied element-wise. It preserves the vector's shape: $ReLU\,(\cdot) : \mathbb{R}^{d_{k+1}} \to \mathbb{R}^{d_{k+1}}$

- Therefore, the updated node representation $h_v^{k+1} \in \mathbb{R}^{d_{k+1}}$, maintaining consistency across all layers.

This ensures the model can propagate consistent-sized feature vectors through each layer in the network.

Source: (Kipf and Welling, 2016)

Ref. By: A1, A2, TM4.3.3, GD2, GD3, GD6, GD7, DD2, IM1

Preconditions for TM1:GNNFP: None

Derivation for TM1:GNNFP: Not Applicable

RefName: **TM2:LF**

Label: Loss Function for Re-ProtGNN

Equation: $\mathcal{L} = \mathcal{L}_{CE} + \lambda_1 \mathcal{L}_{\text{Clst}} + \lambda_2 \mathcal{L}_{\text{Sep}} + \lambda_3 \mathcal{L}_{\text{Div}}$

Description: The overall loss function consists of multiple terms:

- $\mathcal{L}_{CE} = -y \log \hat{y} - (1-y) \log(1-\hat{y})$: Cross-entropy loss for classification. (Type: $\mathbb{R}$)
- $\mathcal{L}_{\text{Clst}} = \frac{1}{n} \sum_{i=1}^{n} \min_{j:p_j \in P_{y_i}} \|f(x_i) - p_j\|_2^2$: Cluster loss to ensure embeddings are close to prototypes of the same class. (Type: $\mathbb{R}$)
- $\mathcal{L}_{\text{Sep}} == -\frac{1}{n} \sum_{i=1}^{n} \min_{j:p_j \notin P_{y_i}} \|f(x_i) - p_j\|_2^2$: Separation loss to push embeddings away from prototypes of other classes. (Type: $\mathbb{R}$)
- $\mathcal{L}_{\text{Div}} = \sum_{k=1}^{C} \sum_{\substack{i \neq j \\ p_i, p_j \in P_k}} \max(0, \cos(p_i, p_j) - s_{\max})$: Diversity loss to prevent prototypes from collapsing. (Type: $\mathbb{R}$)
- y: Ground-truth label of an input graph. (Type: $\mathbb{Z}$)
- $\hat{y}$: Predicted label of an input graph. (Type: $\mathbb{R}$)
- p_j: A prototype embedding representing a characteristic subgraph associated with a specific class. (Type: a vector of $\mathbb{R}$)
- x_i: The node feature representation of the i-th input graph in the dataset. (Type: a matrix of $\mathbb{R}$)
- $f(\cdot)$: A function representing the graph encoder, as defined in TM4.3.3.
- $s_{\max}$: Threshold of the cosine similarity measured by $cos(\cdot, \cdot)$ in the diversity loss. (Type: $\mathbb{R}$)
- $\lambda_1, \lambda_2, \lambda_3$: Hyper-parameters (see DD2) balancing between prediction accuracy (imposed by cross-entropy loss), and prototype finding (imposed by cluster loss, separation loss, and divergence loss). (Type: $\mathbb{R}$)

Notes: The cross-entropy loss is a common loss used in many ML models, while the other losses (i.e., cluster loss, separation loss, and diversity loss) are proposed in the context of Re-ProtGNN.

Source: (Wikipedia contributors, 2024b), (Zhang et al., 2022)

Ref. By: GD4, GD5, GD6, GD7, IM1

Preconditions for TM2:LF: None

Derivation for TM2:LF: Not Applicable

4.3.4 General Definitions

This section collects the laws and equations that will be used in building the instance models.

Number	GD1
Label	**Chain Rule**
SI Units	Unitless
Equation	$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$
Description	The chain rule states that the derivative of a function with respect to an independent variable can be computed as the product of the derivative of the function with respect to an intermediate variable and the derivative of the intermediate variable with respect to the independent variable.
Source	(Wikipedia contributors, 2024a)
Ref. By	GD4, GD5, GD6

Number	GD2
Label	**Sigmoid Function for Fully Connected Layer**
SI Units	Unitless
Equation	$\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}}, \quad \text{where} \quad z = W^\top h + b$
Description	h: Graph embedding obtained after processing input node features X through the GNN encoder (see TM4.3.3). (Type: a vector in $\mathbb{R}^d$) W: Trainable weight vector of the Fully Connected Layer that maps graph embeddings to a scalar logit. (Type: a vector in $\mathbb{R}^d$) b: Bias term added to the logit. (Type: a scalar in $\mathbb{R}$) $z = W^\top h + b$: Logit value before activation. (Type: a scalar in $\mathbb{R}$) $\sigma(z)$: Sigmoid activation function applied to the scalar logit. (Input/Output: scalar in $\mathbb{R}$) $\hat{y}$: Predicted probability for the positive class. (Type: a scalar in $\mathbb{R}$)
Notes	Since Re-ProtGNN performs binary classification (e.g., for the MUTAG dataset), the output $\hat{y} \in \mathbb{R}$ represents the probability of the input graph belonging to the positive class. The sigmoid function squashes the logit into a value in $(0, 1)$, making it suitable for interpretation as a probability.
Source	(Wikipedia contributors, 2024a)
Ref. By	GD4, GD5, GD6

Number	GD3
Label	**Loss Minimization Objective**
SI Units	Unitless
Equation	$\theta^* = \arg\min_\theta \mathcal{L}(\theta)$
Description	The training process aims to find the optimal set of model parameters θ that minimize the loss function $\mathcal{L}(\theta)$, where θ typically includes the GNN encoder weights W_g (see TM4.3.3), fully connected layer weights W, bias b, and prototype matrix P. • $\mathcal{L}(\theta)$: Overall loss function to be minimized. (Type: scalar in $\mathbb{R}$) • θ: Set of all trainable parameters, i.e., $\theta = \{b, W, W_g, P\}$. (Type: set of matrices/vectors; for detailed types, see GD4, GD5, GD6, GD7) • θ^*: Optimal parameter configuration that minimizes $\mathcal{L}$.
Notes	This is a standard unconstrained optimization problem in machine learning. The specific technique used to solve it (e.g., gradient descent) is introduced in subsequent general definitions.
Source	(?)
Ref. By	GD4, GD5, GD6, GD7, IM1

Number	GD4
Label	**Gradient of Loss Function with respect to the bias b of Fully Connected Layer**
SI Units	Unitless
Equation	$\nabla_b \mathcal{L} = (\hat{y} - y)$
Description	With $\hat{y}$ defined by the Sigmoid Function (see GD2), we derive the gradient of Loss Function (see TM4.3.3) with respect to b using the chain rule (see GD1). This gradient allows us to minimize Loss Function during model training (see GD3, IM1). • $\hat{y}$: Predicted class probability. (Type: scalar in $\mathbb{R}$) • y: Ground-truth label. (Type: scalar in $\mathbb{Z}$) • b: Bias term in the fully connected layer. (Type: scalar in $\mathbb{R}$) • $\nabla_b \mathcal{L}$: Gradient of loss with respect to b. (Type: scalar in $\mathbb{R}$)
Source	(Turin, 2020)
Ref. By	IM1

Number	GD5
Label	**Gradient of Loss Function with respect to the weight W of Fully Connected Layer**
SI Units	Unitless
Equation	$\nabla_W \mathcal{L} = (\hat{y} - y)h$
Description	With $\hat{y}$ defined by the Sigmoid Function (see GD2), we derive the gradient of Loss Function (see TM4.3.3) with respect to W using the chain rule (see GD1). This gradient allows us to minimize Loss Function during model training (see GD3, IM1). • $\hat{y}$: Predicted class probability. (Type: scalar in $\mathbb{R}$) • y: Ground-truth label. (Type: scalar in $\mathbb{Z}$) • h: Graph embedding from the GNN encoder. (Type: vector in $\mathbb{R}^d$) • W: Weight vector of the fully connected layer. (Type: vector in $\mathbb{R}^d$) • $\nabla_W \mathcal{L}$: Gradient of loss with respect to W. (Type: vector in $\mathbb{R}^d$)
Source	(Turin, 2020)
Ref. By	IM1

Number	GD6
Label	**Gradient of Loss Function with respect to the weight W_g of GNN Encoder**
SI Units	Unitless
Equation	$\nabla_{W_g}\mathcal{L} = (\hat{y} - y)W \sum_{u \in \mathcal{N}(v)} \tilde{A}_{vu} h_u$
Description	With $\hat{y}$ defined by the Sigmoid Function (see GD2), we derive the gradient of Loss Function (see TM4.3.3) with respect to W_g using the chain rule (see GD1). This gradient allows us to minimize Loss Function during model training (see GD3, IM1). • $\hat{y}$: Predicted class probability. (Type: scalar in $\mathbb{R}$) • y: Ground-truth label. (Type: scalar in $\mathbb{Z}$) • W: Weight vector of the fully connected layer. (Type: vector in $\mathbb{R}^d$) • h_u: Feature vector of neighbor node u. (Type: vector in $\mathbb{R}^d$) • $\tilde{A}_{vu}$: Scalar entry from normalized adjacency matrix. (Type: scalar in $\mathbb{R}$) • $\sum_{u \in \mathcal{N}(v)} \tilde{A}_{vu} h_u$: Aggregated message from neighbors. (Type: vector in $\mathbb{R}^d$) • W_g: Trainable weight matrix of GNN Encoder (see TM4.3.3) that maps input graphs to graph embeddings. (Type: matrix in $\mathbb{R}^{d \times d_{in}}$) • $\nabla_{W_g}\mathcal{L}$: Gradient of loss with respect to W_g. (Type: matrix in $\mathbb{R}^{d \times d_{in}}$)
Source	(Turin, 2020)
Ref. By	IM1

Number	GD7
Label	**Gradient of Loss Function with respect to P**
SI Units	Unitless
Equation	$\nabla_P \mathcal{L} = 2(h_i - p_j) - 2(h_i - p_k) + \sum_{j \neq i} \mathbf{1}[\cos(p_i, p_j) > s_{\max}] \left(\frac{p_j}{\|p_i\| \|p_j\|} - \frac{(p_i \cdot p_j) p_i}{\|p_i\|^3 \|p_j\|} \right)$
Description	We derive the gradient of the Loss Function (see TM4.3.3) with respect to P. This allows us to minimize the Loss Function for training the model (see GD3, IM1). • h_i: Embedding representation of input graph i, computed by the graph encoder $f(\cdot)$ (see TM4.3.3). (Type: vector in $\mathbb{R}^d$) • p_j, p_k: Prototype vectors. (Type: vector in $\mathbb{R}^d$) • P: Set of all prototype vectors. (Type: matrix in $\mathbb{R}^{M \times d}$, where M is the number of prototypes) • $\cos(p_i, p_j)$: Cosine similarity between prototypes. (Type: scalar in $\mathbb{R}$) • $s_{\max}$: Threshold for cosine similarity. (Type: scalar in $\mathbb{R}$) • $\nabla_P \mathcal{L}$: Gradient of loss with respect to P. (Type: matrix in $\mathbb{R}^{M \times d}$)
Source	(Turin, 2020)
Ref. By	IM1

Detailed derivation of gradients for Re-ProtGNN

To optimize Re-ProtGNN, we minimize the objective function:

$$\mathcal{L} = \mathcal{L}_{CE} + \lambda_1 \mathcal{L}_{\text{Clst}} + \lambda_2 \mathcal{L}_{\text{Sep}} + \lambda_3 \mathcal{L}_{\text{Div}}$$

where

- $\mathcal{L}_{CE}$: Cross-entropy loss for classification.
- $\mathcal{L}_{\text{Clst}}$: Cluster loss ensuring embeddings stay close to correct prototypes.
- $\mathcal{L}_{\text{Sep}}$: Separation loss ensuring embeddings are far from incorrect prototypes.
- $\mathcal{L}_{\text{Div}}$: Diversity loss preventing prototype collapse.

Step 1: Gradient of Cross-Entropy Loss on W and b

For a given prediction $\hat{y}$ with label y, the cross-entropy loss is

$$\mathcal{L}_{CE} = -y \log \hat{y} - (1-y) \log(1-\hat{y}).$$

Taking the derivative with respect to $\hat{y}$ gives

$$\frac{d}{d\hat{y}} \mathcal{L}_{CE} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}.$$

Using the property of the sigmoid function

$$\hat{y} = \sigma(Z) = \frac{1}{1+e^{-z}},$$

We differentiate $\hat{y}$ with respect to z

$$\frac{d}{dZ}\sigma(Z) = \sigma(Z)(1-\sigma(Z)) = \hat{y}(1-\hat{y})$$

and apply the chain rule

$$\frac{d}{dZ} \mathcal{L}_{CE} = (\hat{y} - y).$$

Since Z = Wh + b, we apply the chain rule again to get the gradients with respect to W and b:

$$\frac{d\mathcal{L}_{CE}}{dW} = \frac{d}{dZ}\frac{dZ}{W}\mathcal{L}_{CE} = (\hat{y}-y)h,$$
$$\frac{d\mathcal{L}_{CE}}{db} = \frac{d}{dZ}\frac{dZ}{b}\mathcal{L}_{CE} = (\hat{y}-y).$$

Step 2: Gradient of Cross-Entropy Loss on W_g

The GNN encoder generates the embeddings h using message passing:

$$h_v = \sigma\left(\sum_{u \in \mathcal{N}(v)} \tilde{A}_{vu} W_g h_u\right),$$

where $\tilde{A}$ is the normalized adjacency matrix, W_g is the trainable weight matrix of the GNN, and $\sigma(\cdot)$ is a non-linear activation function.

Taking the derivative of the loss function with respect to W_g, we apply the chain rule:

$$\nabla_{W_g} \mathcal{L} = \frac{d\mathcal{L}}{d\hat{y}} \cdot \frac{d\hat{y}}{dh} \cdot \frac{dh}{dW_g}.$$

From Step 1, we already derived:

$$\frac{d\mathcal{L}}{d\hat{y}} = (\hat{y} - y),$$

and

$$\frac{d\hat{y}}{dh} = W.$$

Since the GNN encoder propagates node features through message passing, differentiating h with respect to W_g gives:

$$\nabla_{W_g} h_v = \sum_{u \in \mathcal{N}(v)} \tilde{A}_{vu} h_u.$$

Thus, combining these terms, the final gradient is:

$$\nabla_{W_g} \mathcal{L} = (\hat{y} - y) W \sum_{u \in \mathcal{N}(v)} \tilde{A}_{vu} h_u.$$

This expression shows that computing $\nabla_{W_g} \mathcal{L}$ requires propagating gradients through the entire graph, making the computation iterative rather than explicit.

Step 3: Gradient of Prototype Cluster Loss on P

The cluster loss is defined as

$$\mathcal{L}_{\text{Clst}} = \sum_{i=1}^{n} \min_{j: p_j \in P_{y_i}} ||h_i - p_j||^2$$

Differentiating with respect to p_j,

$$\frac{d}{dp_j} \mathcal{L}_{\text{Clst}} = -2(h_i - p_j)$$

Step 4: Gradient of Prototype Separation Loss on P

The separation loss is

$$\mathcal{L}_{\text{Sep}} = -\sum_{i=1}^{n} \min_{j: p_j \notin P_{y_i}} ||h_i - p_j||^2$$

Differentiating with respect to p_j

$$\frac{d}{dp_j} \mathcal{L}_{\text{Sep}} = 2(h_i - p_j)$$

Step 5: Gradient of the Diversity Loss on P

To prevent prototype collapse, we enforce diversity via cosine similarity

$$\mathcal{L}_{\text{Div}} = \sum_{k=1}^{C} \sum_{i \neq j, p_i, p_j \in P_k} \max(0, \cos(p_i, p_j) - s_{\max})$$

where

$$\cos(p_i, p_j) = \frac{p_i \cdot p_j}{||p_i|| ||p_j||}$$

Using the gradient of cosine similarity

$$\frac{d}{dp_i} \cos(p_i, p_j) = \frac{p_j}{||p_i|| ||p_j||} - \frac{(p_i \cdot p_j) p_i}{||p_i||^3 ||p_j||}$$

The final update step for prototype diversity

$$\frac{d}{dp_i} \mathcal{L}_{\text{Div}} = \sum_{j \neq i} \mathbf{1}[\cos(p_i, p_j) > s_{\max}] \left(\frac{p_j}{||p_i|| ||p_j||} - \frac{(p_i \cdot p_j) p_i}{||p_i||^3 ||p_j||} \right)$$

Gradients Summary

To sum up, Cross-entropy loss $\mathcal{L}_{CE}$ updates the weight W and the bias b of the Fully Connected Layer, as well as updates the weight W_g of GNN Encoder. Cluster Loss, separation Loss, and diversity Loss update the prototype P:

$$\frac{d\mathcal{L}}{db} = \frac{d\mathcal{L}_{CE}}{db},$$

$$\frac{d\mathcal{L}}{dW} = \frac{d\mathcal{L}_{CE}}{dW},$$

$$\frac{d\mathcal{L}}{dW_g} = \frac{d\mathcal{L}_{CE}}{dW_g},$$

$$\frac{d\mathcal{L}}{dP} = \frac{d\mathcal{L}_{Clst}}{dP} + \frac{d\mathcal{L}_{Sep}}{dP} + \frac{d\mathcal{L}_{Div}}{dP}.$$

This leads to the final formula seen in GD4, GD5, GD6, GD7.

4.3.5 Data Definitions

This section collects and defines all the data needed to build the instance models. The dimension of each quantity is also given.

Number	DD1
Label	**Learning Rate**
Symbol	η
SI Units	Unitless
Equation	Not Applicable
Description	The learning rate is a hyperparameter that controls the step size at each iteration while moving toward a minimum of the loss function. It plays an important role in balancing convergence speed and stability.
Sources	(Wikipedia contributors, 2024d)
Ref. By	IM1

Number	DD2
Label	**Regularization Parameters**
Symbol	$\lambda_1, \lambda_2, \lambda_3$
SI Units	Unitless
Equation	Not Applicable
Description	The regularization parameters control the trade-off between the main classification loss and additional constraints (see TM4.3.3) to enhance interpretability and prevent overfitting.
Sources	(Wikipedia contributors, 2024e)
Ref. By	TM4.3.3

4.3.6 Data Types

Not applicable, as the inputs and outputs consist of straightforward types, such as sequences of integers or matrices of real numbers.

4.3.7 Instance Models

This section transforms the problem defined in Section 4.1 into one which is expressed in mathematical terms. It uses concrete symbols defined in Section 4.3.5 to replace the abstract symbols in the models identified in Sections 4.3.3 and 4.3.4.

The goal GS1 is achieved through IM1 and IM2. IM1 trains the model to ensure that IM2 delivers accurate classifications and reliable interpretations.

Number	IM1
Label	**Model Training via Gradient Descent**
Input	Node features X, adjacency matrix A, prototypes P, learning rate η.
Output	Optimized parameters W, b, P
Description	The model is trained by iteratively updating the parameters via gradient descent: 1. Compute embeddings using GNN layers (see TM4.3.3). 2. Compute the loss function $\mathcal{L}$ (see TM4.3.3). 3. Update parameters using $W \leftarrow W - \eta\frac{\partial \mathcal{L}}{\partial W}$, $b \leftarrow b - \eta\frac{\partial \mathcal{L}}{\partial b}$, $P \leftarrow P - \eta\frac{\partial \mathcal{L}}{\partial P}$ (see DD1, GD3, GD4, GD5, GD6, GD7, TM4.3.3).
Sources	(Zhang et al., 2022)
Ref. by	A3, R2, GD4, GD5, GD6, GD7, IM2

Number	IM2
Label	**Model Inference via Forward Propagation**
Input	Node features X, adjacency matrix A, trained parameters W, b, P.
Output	Predicted class labels $\hat{y}$.
Description	During inference, the model forwards input features through the trained model (see IM1) and directly gets the classification results.
Sources	(Run:AI, 2024)
Ref. by	A3, R3

4.3.8 Input Data Constraints

Table 3 shows the data constraints on the input output variables. The column for physical constraints gives the physical limitations on the range of values that can be taken by the variable. The column for software constraints restricts the range of inputs to reasonable values. The software constraints will be helpful in the design stage for picking suitable

algorithms. The constraints are conservative, to give the user of the model the flexibility to experiment with unusual situations. The column of typical values is intended to provide a feel for a common scenario. The uncertainty column provides an estimate of the confidence with which the physical quantities can be measured. This information would be part of the input if one were performing an uncertainty quantification exercise.

The specification parameters in Table 3 are listed in Table 5.

Table 3: Input Variables

Var	Physical Constraints	Software Constraints	Typical Value	Uncertainty
X	$X[i,j] \in \{-\infty, +\infty\}$	$X[i,j] \in \{-\infty, +\infty\}$	depends on the dataset	-
A	$A[i,j] \in \{0,1\}$	$A[i,j] \in \{0,1\}$	depends on the dataset	-
y	$y \in \{-\infty, +\infty\}$	$y \in \{1, ..., C\}$	depends on the dataset	-

Table 5: Specification Parameter Values

Var	Value
C	depends on the dataset

4.3.9 Properties of a Correct Solution

A correct solution must ensure that the predicted class is a valid label and that the prototypes are valid graphs, following the constraints in Table 7. In addition, each prototype should be a subgraph of at least one input graph.

Table 7: Output Variables

Var	Physical Constraints
X_p	num_rows$(X_p) <$ num_rows(X)
A_p	num_rows$(A_p) =$ num_cols(A_p), $dim(A_p) < dim(A)$
$\hat{y}$	$\hat{y} \in \{1, ..., C\}$

5 Requirements

This section provides the functional requirements, the business tasks that the software is expected to complete, and the nonfunctional requirements, the qualities that the software is expected to exhibit.

5.1 Functional Requirements

R1: The system should be able to load the MUTAG dataset (Debnath et al., 1991) .

R2: During the training phase, the system should update model parameters and learn prototype representations by optimizing the Loss Function (see IM1).

R3: During the inference phase, the system should compute classification accuracy. (see IM2).

5.2 Nonfunctional Requirements

NFR1: **Reliability** The system should operate reliably without unexpected failures. Reliability will be evaluated through testings as outlined in the VnV Plan.

NFR2: **Usability** The user interface and interaction flow should be intuitive and require minimal user effort or prior training.

5.3 Rationale

The rationale for the scope decisions (Section 4.3) is that Re-ProtGNN uses the MUTAG dataset (Debnath et al., 1991) because the original ProtGNN model was evaluated on it. Additionally, MUTAG is a well-established benchmark for graph classification, ensuring a fair validation of ProtGNN's performance.

In addition, the model is designed to follow the structure of the original ProtGNN framework to ensure consistency.

6 Likely Changes

LC1: A3 The system may be extended to support training on a broader range of datasets, such as BA-shape, to further validate the reproducibility and robustness of ProtGNN's results.

LC2: The model may use alternative GNN encoders, such as Graph Isomorphism Networks (GINs), to evaluate the impact of different architectures on classification accuracy and interpretability.

7 Unlikely Changes

LC3: It is unlikely that the system will transition to a fully unsupervised setting, as ProtGNN relies on labeled data to learn prototype representations.

LC4: The system is unlikely to remove the prototype-based interpretability mechanism, as this is a fundamental aspect of ProtGNN.

8 Traceability Matrices and Graphs

The purpose of the traceability matrices is to provide easy references on what has to be additionally modified if a certain component is changed. Every time a component is changed, the items in the column of that component that are marked with an "X" may have to be modified as well. Table 9 shows the dependencies of theoretical models, general definitions, data definitions, and instance models with each other. Table 10 shows the dependencies of instance models, requirements, and data constraints on each other. Table 11 shows the dependencies of theoretical models, general definitions, data definitions, instance models, and likely changes on the assumptions.

	TM4.3.3	TM4.3.3	GD1	GD2	GD3	GD4	GD5	GD6	GD7	DD1	DD2	IM1	IM2
TM4.3.3													
TM4.3.3	X												
GD1													
GD2	X												
GD3	X	X				X	X	X	X				
GD4	X	X	X	X	X	X	X	X	X	X	X	X	
GD5	X	X	X	X	X	X	X	X	X	X	X	X	
GD6	X	X	X	X	X	X	X	X	X	X	X	X	
GD7	X	X	X	X	X	X	X	X	X	X	X	X	
DD1													
DD2	X												
IM1	X	X	X	X	X	X	X	X	X	X	X		
IM2	X	X	X	X	X	X	X	X	X	X	X	X	

Table 9: Traceability Matrix Showing the Connections Between Items of Different Sections

	IM1	IM2	R1	R2	R3	NFR1	NFR2
IM1					X		
IM2	X				X		
R1					X	X	
R2	X				X	X	
R3	X	X			X	X	
NFR1						X	
NFR2							

Table 10: Traceability Matrix Showing the Connections Between Requirements and Instance Models

	A1	A2	A3
TM4.3.3	X	X	
TM4.3.3	X	X	
GD1			
GD2	X		
GD4	X	X	X
GD5	X	X	X
GD6	X	X	X
GD7	X	X	X
DD1			
DD2	X	X	
IM1	X	X	X
IM2			X
LC1			X
LC2			
LC3			
LC4			

Table 11: Traceability Matrix Showing the Connections Between Assumptions and Other Items

References

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, February 1991. doi: 10.1021/jm00106a046.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, September 2016. URL https://arxiv.org/abs/1609.02907.

Run:AI. Understanding machine learning inference, 2024. URL https://www.run.ai/guides/machine-learning-inference/understanding-machine-learning-inference. Accessed: February 6, 2025.

ScienceDirect. Cosine similarity, 2024. URL https://www.sciencedirect.com/topics/computer-science/cosine-similarity. Accessed: February 6, 2025.

W. Spencer Smith and Lei Lai. A new requirements template for scientific computing. In J. Ralyté, P. Ågerfalk, and N. Kraiem, editors, *Proceedings of the First International Workshop on Situational Requirements Engineering Processes – Methods, Techniques and Tools to Support Situation-Specific Requirements Engineering Processes, SREP'05*, pages 107–121, Paris, France, 2005. In conjunction with 13th IEEE International Requirements Engineering Conference.

W. Spencer Smith, Lei Lai, and Ridha Khedri. Requirements analysis for engineering computation: A systematic approach for improving software reliability. *Reliable Computing, Special Issue on Reliable Engineering Computation*, 13(1):83–107, February 2007.

Arseny Turin. Logistic regression from scratch. https://towardsdatascience.com/logistic-regression-from-scratch-69db4f587e17, 2020.

Wikipedia contributors. Chain rule — Wikipedia, the free encyclopedia, 2024a. URL https://en.wikipedia.org/wiki/Chain_rule. Accessed: February 6, 2025.

Wikipedia contributors. Cross-entropy — Wikipedia, the free encyclopedia, 2024b. URL https://en.wikipedia.org/wiki/Cross-entropy. Accessed: February 6, 2025.

Wikipedia contributors. Gradient — Wikipedia, the free encyclopedia, 2024c. URL https://en.wikipedia.org/wiki/Gradient. Accessed: February 6, 2025.

Wikipedia contributors. Learning rate — Wikipedia, the free encyclopedia, 2024d. URL https://en.wikipedia.org/wiki/Learning_rate. Accessed: February 6, 2025.

Wikipedia contributors. Regularization (mathematics) — Wikipedia, the free encyclopedia, 2024e. URL https://en.wikipedia.org/wiki/Regularization_(mathematics). Accessed: February 6, 2025.

Wikipedia contributors. Rectifier (neural networks) — Wikipedia, the free encyclopedia, 2024f. URL https://en.wikipedia.org/wiki/Rectifier_(neural_networks). Accessed: February 6, 2025.

Yuanqi Xue. Problem statement and goals for re-protgnn, 2025. URL https://github.com/Yuanqi-X/Re-ProtGNN/blob/main/docs/ProblemStatementAndGoals/ProblemStatement.pdf. Accessed: February 6, 2025.

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