System Verification and Validation Plan for ProgName

Yuanqi Xue

March 27, 2025

Revision History

Date	Version	Notes
Feb 24	1.0	Initial Draft

Contents

List of Tables

1 Symbols, Abbreviations, and Acronyms

symbol	description
T	Test
A	Assumption
R	Requirement
SRS	Software Requirements Specification
ProtGNN	Prototype-based Graph Neural Network
Re-ProtGNN	Re-implementation of the ProtGNN model
GNN	Graph Neural Network
GIN	Graph Isomorphism Network

This document outlines the Verification and Validation (VnV) plan for Re-ProtGNN, a prototype-based interpretable Graph Neural Network. The objective of this plan is to ensure that the system meets the functional and non-functional requirements specified in the Software Requirements Specification (SRS). The document is structured as follows: Section 2 provides general information about Re-ProtGNN, including its objectives and scope. Section 3 details the verification strategies, covering SRS verification, design verification, and implementation verification. Section 4 describes the system tests, including functional and non-functional testing. Finally, Section 5 will cover additional test descriptions as needed.

2 General Information

2.1 Summary

The software under validation is Re-ProtGNN, a model designed to enhance the interpretability of Graph Neural Networks. It aims to classify graph-structured data while generating prototypes that explain its predictions. The system consists of two main components:

- Training Phase: Learns meaningful representations of graphs while optimizing a loss function to improve both classification accuracy and prototype relevance.
- Inference Phase: Uses the trained model to classify unseen graphs and generate a set of representative prototypes that provide human-interpretable explanations.

Re-ProtGNN is implemented in Python, utilizing PyTorch Geometric for graph learning and Pytest for automated testing.

2.2 Objectives

The main goals of this VnV plan are:

- Correctness: Ensure that the system correctly processes graph-structured inputs, trains models effectively, and generates meaningful prototype-based explanations.

- Accuracy: Assess whether the model attains a minimum classification accuracy of 80% on the MUTAG dataset and confirm that the selected prototypes align with expected graph representations.
- Interpretability: Validate that the extracted prototypes contribute to a clearer understanding of the model's decision-making process.

Out-of-Scope Objectives:

- External Library Verification: Core dependencies such as PyTorch and Torch-Geometric are presumed to be reliable and are not explicitly tested in this plan.
- Graph Encoder Validation: Graph encoders, including Graph Isomorphism Networks (GINs), are adopted from prior research, and their correctness is assumed without additional validation.

2.3 Challenge Level and Extras

This is a research project, but extras may be included if time permits.

2.4 Relevant Documentation

The Re-ProtGNN project is supported by several key documents that ensure the system is properly designed, implemented, and validated. These documents include:

- Problem Statement: This document Xue (2025) introduces the motivation of Re-ProtGNN and the core problem it aims to solve.
- Software Requirements Specification (SRS): The SRS Xue (2024) defines the functional and non-functional requirements of Re-ProtGNN, and it also outlines the expected behavior of the system.

3 Plan

This section outlines the Verification and Validation (VnV) plan for Re-ProtGNN. It starts with an introduction to the verification and validation team (Subsection 3.1), detailing the members and their roles. Next, it covers

the SRS verification plan (Subsection 3.2), followed by the design verification plan (Subsection 3.3). The document then presents the VnV verification plan (Subsection 3.4) and the implementation verification plan (Subsection 3.5). Finally, it includes automated testing and verification tools (Subsection 3.6) and concludes with the software validation plan (Subsection 3.7).

3.1 Verification and Validation Team

Name	Document	Role	Description
Yuanqi Xue	All	Author	Create and manage all required documents, develop the VnV plan, conduct VnV testing, and verify the implementation.
Dr. Spencer Smith	All	Instructor/ Reviewer	Review all the documents.
Yinying Huo	All	Domain Expert	Review all the documents.

Table 1: Verification and Validation Team

3.2 SRS Verification Plan

An initial review of the SRS will be conducted by Dr. Spencer Smith and Yinying Huo to ensure its accuracy, completeness, and feasibility. The review will follow a manual inspection process using an SRS Checklist to assess the clarity and alignment of SRS with project objectives.

Reviewers will provide feedback via GitHub issues, and Yuanqi Xue, as the author, will be responsible for addressing revisions.

3.3 Design Verification Plan

The design verification process, covering the Module Guide (MG) and Module Interface Specification (MIS), will be conducted by Dr. Spencer Smith and domain expert Yinying Huo. Their feedback will be shared through GitHub issues, and Yuanqi Xue will be responsible for making the necessary revisions.

To ensure a structured verification process, Dr. Spencer Smith has created an MG checklist and MIS checklist, both of which will be used to evaluate clarity, consistency, and correctness in the design documents. The verification process will ensure that the system architecture, module interactions, and design choices align with the project objectives.

3.4 Verification and Validation Plan Verification Plan

The Verification and Validation (VnV) Plan will be reviewed by Dr. Spencer Smith and domain expert Yinying Huo to ensure that the validation methodology aligns with the project's objectives. Feedback from reviewers will be provided via GitHub issues, where all necessary revisions and updates will be documented.

To maintain consistency and thorough evaluation, a VnV checklist prepared by Dr. Spencer Smith will be used to assess the completeness, correctness, and applicability of the verification and validation process.

3.5 Implementation Verification Plan

The implementation of Re-ProtGNN will be verified through a combination of unit tests, system tests, and a final code walkthrough during the class presentation. This ensures that both functional and non-functional requirements are met, as outlined in Section 4.

3.5.1 Static Verification Techniques

- A code walkthrough will be conducted during the final class presentation, where the author, Yuanqi Xue, will present the system architecture, key implementation details, and model behavior.
- The walkthrough will provide an opportunity for peer review, allowing classmates and the instructor to identify potential issues and suggest improvements.

3.5.2 Dynamic Testing

- The system will undergo unit and system testing, validating both functional and non-functional requirements as specified in Section 4.

- Unit tests will focus on key components such as data preprocessing, model training, and prototype generation.
- System tests will evaluate the end-to-end performance of Re-ProtGNN, ensuring a desired classification accuracy and prototype demonstration.
- The testing tools and methodologies are detailed in Section 3.6, and test cases are outlined in Section 4.

3.6 Automated Testing and Verification Tools

Re-ProtGNN will use a combination of automated testing and verification tools to ensure code correctness:

Unit Testing: Pytest will be used for testing individual components, including data processing, loss functions, and inference logic. These tests will help verify that each module functions as expected before integration into the full system.

Continuous Integration (CI): GitHub Actions will be configured to automate testing after each commit. The workflow will include:

- Running unit tests to verify functionality.
- Performing data integrity checks.
- Validating dependencies to prevent compatibility issues.

Regression Testing: A dedicated test suite will ensure that changes do not negatively impact previously validated functionality. Logs from test runs will be automatically recorded and analyzed to track issues over time, ensuring that updates and modifications do not introduce unexpected errors.

3.7 Software Validation Plan

A separate 20% test split from the MUTAG dataset will be used to assess the model's classification effectiveness and the clarity of its prototype-based explanations, with accuracy serving as the main benchmark.

4 System Tests

This section outlines the system tests designed to evaluate both functional and non-functional requirements.

4.1 Tests for Functional Requirements

This section outlines the tests designed to validate the functional requirements of Re-ProtGNN, ensuring the system behaves as expected under various conditions. The testing areas include input verification, model training correctness, and inference validation, which corresponds to the R1, R2, and R3 of SRS Xue (2024).

4.1.1 Area of Testing1: Input Verification

We need to make sure that all input graphs follow the correct format and structure before training begins. This test checks whether the system properly handles the input dataset files and follows the constraints listed in the SRS Xue (2024).

Test for Valid Inputs

1. T1: Valid Inputs

   Control: Automated Test

   Initial State: Pending Input

   Input: The MUTAG dataset files.

   Output: If the input is valid, the system should load it successfully. Otherwise, it should return an appropriate error message, as in Table 2.

   Test Case Derivation: Since Graph Neural Networks (GNNs) rely on structured input, this test makes sure the model doesn't break when given bad data.

   How test will be performed: An automated script will try loading the MUTAG dataset using the data loader and check whether the system correctly accepts or rejects them.

4.1.2 Area of Testing2: Model Training Verification

We need to make sure the training process work. In other words, the model should be able to optimize its loss function and learn properly.

ID	Graph Type	Valid?	Error Message
TI-1	MUTAG (Adj+Feat)	Yes	NONE
TI-2	MUTAG (Missing Feats)	No	Missing Node Features
TI-3	Corrupted Graph	No	Invalid Adjacency Matrix
TI-4	Non-Graph Input	No	Incorrect Input Type

Table 2: Validation results for loading the MUTAG dataset

Test for Training Loss Optimization

1. T2: Model Training Test

   Control: Automatic

   Initial State: Model is initialized with random weights.

   Input: The loaded MUTAG dataset.

   Output: The system should optimize the loss function and display a decreasing loss curve over multiple epochs.

   Test Case Derivation: Ensures that the model is learning from the input data instead of failing due to improper gradients or data mismatches.

   How test will be performed: Train the model on valid datasets and log the loss over time. A Pytest script will validate loss reduction.

4.1.3 Area of Testing3: Inference Verification

This test ensures that the trained model makes accurate predictions and correctly retrieves prototype-based explanations.

Test for Inference

1. T3: Inference Test

   Control: Automatic

   Initial State: Model is trained and ready for inference.

Input: The testing dataset.

Output: The model should output a predicted label for each graph and a set of prototypes.

Test Case Derivation: Ensures that the trained model can be used in the inference phase for classifying inputs and generating a customized numbers of prototypes.

How test will be performed: A Pytest script will run inference on the test dataset, comparing the predicted labels with the ground truth to compute classification accuracy. The system should also generate a set of representative prototypes for the entire dataset. Pytest will verify that the correct number of prototypes is produced.

4.2 Tests for Nonfunctional Requirements

This section outlines the tests designed to verify the nonfunctional requirements of Re-ProtGNN, including accuracy, usability, and portability.

4.2.1 Area of Testing1: Accuracy

Accuracy

1. T4: Accuracy Test

   Type: Automated

   Initial State: A trained Re-ProtGNN model is loaded.

   Input: The testing dataset.

   Output: The system should achieve a classification accuracy of at least 80%.

   How test will be performed: A Pytest script will run inference on the test dataset and compare the predicted labels with ground truth. The script will calculate the overall classification accuracy and ensure it meets the required threshold ($\geq$ 80%). The accuracy percentage will be logged to evaluate model performance.

4.2.2 Area of Testing2: Usability

Usability

1. T5: Usability Test

   Type: Manual

   Initial State: Trained Re-ProtGNN model is available.

   Input: A set of graphs with learned prototypes.

   Output: The system should visualize prototypes as graph images, making them more interpretable than raw adjacency matrices or node feature representations.

   How test will be performed: A set of sample graphs will be processed through the system, and the generated prototype visualizations will be reviewed for clarity and interpretability. Testers will manually verify whether the images effectively represent key graph structures and provide meaningful insights into model decisions. Feedback will be collected to assess usability improvements.

4.2.3 Area of Testing3: Portability

Portability

1. T6: Portability Test

   Type: Manual

   Initial State: None

   Condition: The users should have PyTorch 1.8.0 and Torch-Geometric 2.0.2 installed.

   Result: The system should successfully run on users' machines, with all functions working as expected.

   How test will be performed: The author, Yuanqi Xue, will install the software on Windows, macOS, and Linux systems, verifying that it runs correctly across all platforms.

	T1	T2	T3	T4	T5	T6
R1	X					
R2		X				
R3			X		X	
NFR1				X		
NFR2					X	
NFR3						X

Table 3: Traceability Matrix of Test Cases and Requirements

4.3 Traceability Between Test Cases and Requirements

5 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

5.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

Yuanqi Xue. Software requirements specification (srs) for re-protgnn, 2024. URL https://github.com/Yuanqi-X/Re-ProtGNN/blob/main/docs/SRS/SRS.pdf. Accessed: 2025-02-24.

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.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

6.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]

Appendix — Reflection

[This section is not required for CAS 741 —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.

4. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?