Module Guide for Re-ProtGNN

Yuanqi Xue

April 18, 2025

1 Revision History

Date	Version	Notes
Mar 19, 2025	1.0	First Draft

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
OS	Operating System
R	Requirement
SC	Scientific Computing
SRS	Software Requirements Specification
ProgName	Explanation of program name
UC	Unlikely Change
Re-ProtGNN	Re-implementation of the Prototype-based Graph Neural Network model

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section 6 specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The specific hardware on which the software is running.

AC2: The values of hyperparameters used for model architecture, optimization, and explanation.

AC3: The format of the initial input data.

AC4: The pipeline stages of the system, such as adding a stage for visualizing model explanations after training.

AC5: The optimization algorithm used in training, such as changing from Adam to Stochastic Gradient Descent.

AC6: The prediction postprocessing method when doing inference.

AC7: The activation function of the model.

AC8: The method by which prototype vectors are mapped to real subgraphs.

AC9: The visualization techniques for presenting model outputs and explanations.

AC10: The loss function used for model training.

AC11: The batching mechanism for graph data.

AC12: The figure rendering strategy.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC2: The use of `Tensor` from PyTorch for numerical computation, including matrix operations and gradient-based learning.

UC3: The use of PyTorch Geometric libraries for handling graph data, including the `Data` object, which encapsulates node features, edge indices, and optionally labels, to represent individual graphs, and the `DataLoader`, which provides mini-batch support for training and inference workflows.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module

M2: Configuration Module

M3: Input Format Module

M4: Control Module

M5: Training Module

M6: Output Visualization Module

M7: Model Module

M8: Inference Module

M9: Explanation Module

M10: PyTorch Module

M11: PyTorch Geometric Module

M12: GUI Module

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Configuration Module
	Input Format Module
	Control Module
	Training Module
	Output Visualization Module
Software Decision Module	Model Module
	Inference Module
	Explanation Module
	PyTorch Module
	PyTorch Geometric Module
	GUI Module

Table 1: Module Hierarchy

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by Parnas et al. (1984). The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *Re-ProtGNN* means the module will be implemented by the Re-ProtGNN software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Modules (M1)

Secrets: The data structure and algorithm used to implement the virtual hardware.

Services: Serves as a virtual hardware used by the rest of the system. This module provides the interface between the hardware and the software. So, the system can use it to display outputs or to accept inputs.

Implemented By: OS

7.2 Behaviour-Hiding Module

Secrets: The contents of the required behaviours.

Services: Includes programs that provide externally visible behaviour of the system as specified in the software requirements specification (SRS) documents. This module serves as a communication layer between the hardware-hiding module and the software decision module. The programs in this module will need to change if there are changes in the SRS.

Implemented By: –

7.2.1 Configuration Module (M2)

Secrets: The internal definitions of experiment configuration classes, including dataset selection, model hyperparameters, training schedule, explanation strategy parameters, and reward computation settings.

Services: Exports user-defined hyperparameter instances and global configuration objects (`data_args`, `model_args`, `train_args`, etc.) for consistent access across modules.

Implemented By: Re-ProtGNN

Type of Module: Library

7.2.2 Input Format Module (M3)

Secrets: The format of the input graph datasets.

Services: Loads and parses datasets into standardized graph representations using PyTorch Geometric data structures.

Implemented By: Re-ProtGNN

Type of Module: Library

7.2.3 Control Module (M4)

Secrets: : The execution flow of the system.

Services: Manages the training, inference, and visualization processes.

Implemented By: Re-ProtGNN

Type of Module: Main program

7.2.4 Training Module (M5)

Secrets: The optimization techniques used in training.

Services: Trains the ProtGNN model using supervised and prototype-based losses, and records the loss value and accuracy for each epoch in log files.

Implemented By: Re-ProtGNN

Type of Module: Library

7.2.5 Output Visualization Module (M6)

Secrets: How model prediction outputs and explanations are visualized.

Services: Generates visual representations of explanations and outputs prediction accuracy.

Implemented By: Re-ProtGNN

Type of Module: Library

7.3 Software Decision Module

Secrets: The design decision based on mathematical theorems, physical facts, or programming considerations. The secrets of this module are *not* described in the SRS.

Services: Includes data structure and algorithms used in the system that do not provide direct interaction with the user.

Implemented By: –

7.3.1 Model Module (M7)

Secrets: The architecture of the Re-ProtGNN model.

Services: Defines the parameters and structural design of the Re-ProtGNN model.

Implemented By: Re-ProtGNN

Type of Module: Abstract Data Type

7.3.2 Inference Module (M8)

Secrets: The details of how inference is conducted using the trained model.

Services: Runs inference in evaluation mode and outputs predictions.

Implemented By: Re-ProtGNN

Type of Module: Library

7.3.3 Explanation Module (M9)

Secrets: How prototype vectors are mapped onto real subgraphs.

Services: Updates prototypes to be near real subgraph embeddings, improving interpretability.

Implemented By: Re-ProtGNN

Type of Module: Library

7.3.4 PyTorch Module (M10)

Secrets: Internal tensor operations, optimizers, and loss functions used during training and inference.

Services: Provides tensor-based computation (e.g., matrix operations), gradient propagation, and loss/optimizer routines that are foundational for training deep learning models.

Implemented By: PyTorch Library

Type of Module: External Library

7.3.5 PyTorch Geometric Module (M11)

Secrets: Graph representations, batch processing, and transformation utilities specific to graph-structured data.

Services: Provides data structures (`Data`, `Dataset`, `DataLoader`) and utility functions (e.g., `to_networkx`) for handling graph input/output and mini-batching.

Implemented By: PyTorch Geometric Library

Type of Module: External Library

7.3.6 GUI Module (M12)

Secrets: The rendering and saving process of images used to display predictions and subgraph explanations.

Services: Supports figure creation and rendering via Matplotlib.

Implemented By: Matplotlib Library

Type of Module: External Library

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	**Modules**
R1	M1, M3, M4, M10, M11
R2	M2, M5, M7, M10, M12
R3	M6, M8, M9, M10

Table 2: Trace Between Requirements and Modules

AC	**Modules**
AC1	M1
AC2	M2
AC3	M3
AC4	M4
AC5	M5
AC6	M8
AC7	M7
AC8	M9
AC9	M6
AC10	M10
AC11	M11
AC12	M12

Table 3: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

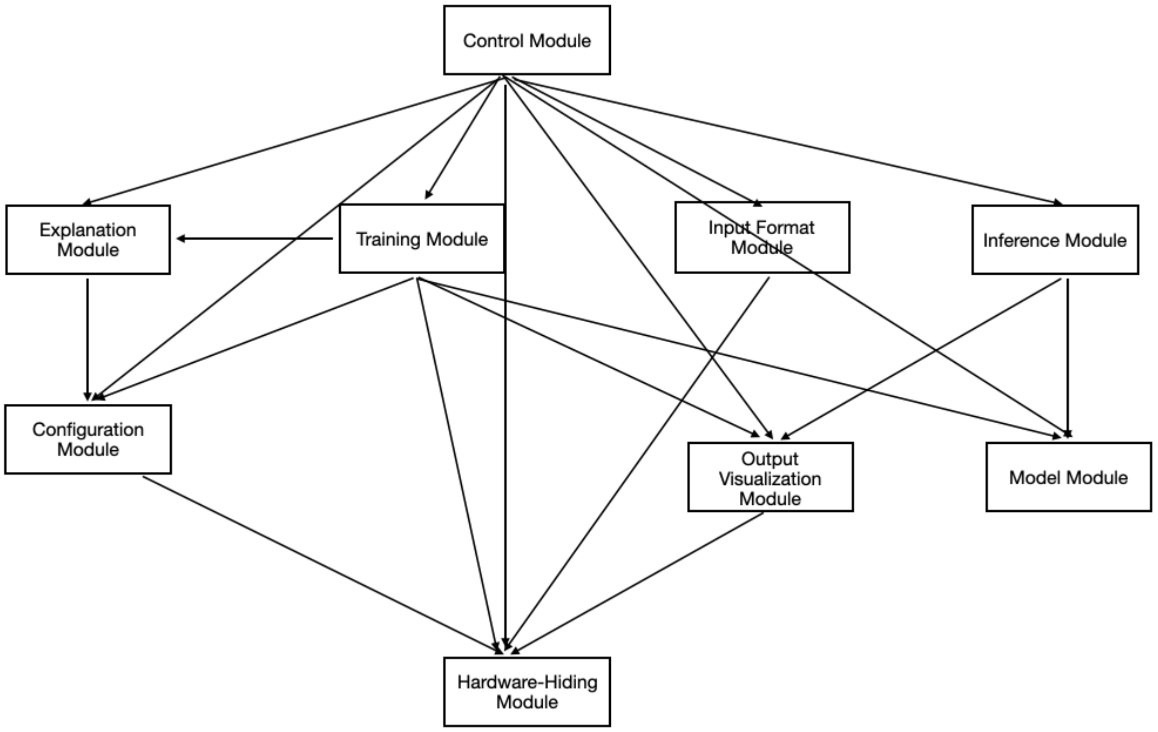

Figure 1: Use hierarchy among modules

10 User Interfaces

The user interface for the Re-ProtGNN system will be a terminal-based interface. Users will interact with the system by running commands from the terminal. Input graph datasets must be placed in a specified input folder, and the system will automatically process them through the training and inference pipeline. Explanations will be generated as image files and saved to a designated output folder, along with prediction results and logs stored in a structured log file.

11 Design of Communication Protocols

Not applicable.

12 Timeline

Please refer to Github.

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.