Module Interface Specification for Re-ProtGNN

Yuanqi Xue

April 19, 2025

1 Revision History

Date	Version	Notes
Mar 19, 2025	1.0	Initial Draft
April 17, 2025	2.0	Final Version

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at https://github.com/Yuanqi-X/Re-ProtGNN/blob/main/docs/SRS/SRS.pdf.

Contents

1 Revision History **i**

2 Symbols, Abbreviations and Acronyms **ii**

3 Introduction **1**

4 Notation **1**

5 Module Decomposition **2**

6 MIS of Configuration Module **3**
6.1 Module . . . 3
6.2 Uses . . . 3
6.3 Syntax . . . 3
6.3.1 Exported Constants . . . 3
6.3.2 Exported Access Programs . . . 3
6.4 Semantics . . . 3
6.4.1 State Variables . . . 3
6.4.2 Environment Variables . . . 3
6.4.3 Access Routine Semantics . . . 4
6.4.4 Local Functions . . . 4

7 MIS of Input Format Module **5**
7.1 Module . . . 5
7.2 Uses . . . 5
7.3 Syntax . . . 5
7.3.1 Exported Constants . . . 5
7.3.2 Exported Access Programs . . . 5
7.4 Semantics . . . 5
7.4.1 State Variables . . . 5
7.4.2 Environment Variables . . . 5
7.4.3 Access Routine Semantics . . . 5
7.4.4 Local Functions . . . 6

8 MIS of Control Module **7**
8.1 Module . . . 7
8.2 Uses . . . 7
8.3 Syntax . . . 7
8.3.1 Exported Constants . . . 7
8.3.2 Exported Access Programs . . . 7
8.4 Semantics . . . 7
8.4.1 State Variables . . . 7

8.4.2 Environment Variables . . . 7
8.4.3 Access Routine Semantics . . . 7
8.4.4 Local Functions . . . 8

9 MIS of Training Module **9**
9.1 Module . . . 9
9.2 Uses . . . 9
9.3 Syntax . . . 9
9.3.1 Exported Constants . . . 9
9.3.2 Exported Access Programs . . . 9
9.4 Semantics . . . 9
9.4.1 State Variables . . . 9
9.4.2 Environment Variables . . . 9
9.4.3 Access Routine Semantics . . . 9
9.4.4 Local Functions . . . 10

10 MIS of Output Visualization Module **11**
10.1 Module . . . 11
10.2 Uses . . . 11
10.3 Syntax . . . 11
10.3.1 Exported Constants . . . 11
10.3.2 Exported Access Programs . . . 11
10.4 Semantics . . . 11
10.4.1 State Variables . . . 11
10.4.2 Environment Variables . . . 11
10.4.3 Access Routine Semantics . . . 12
10.4.4 Local Functions . . . 12

11 MIS of Model Module **13**
11.1 Module . . . 13
11.2 Uses . . . 13
11.3 Syntax . . . 13
11.3.1 Exported Constants . . . 13
11.3.2 Exported Access Programs . . . 13
11.4 Semantics . . . 13
11.4.1 State Variables . . . 13
11.4.2 Environment Variables . . . 13
11.4.3 Access Routine Semantics . . . 13
11.4.4 Local Functions . . . 14

12 MIS of Inference Module **15**
12.1 Module . . . 15
12.2 Uses . . . 15

12.3 Syntax . . . 15
12.3.1 Exported Constants . . . 15
12.3.2 Exported Access Programs . . . 15
12.4 Semantics . . . 15
12.4.1 State Variables . . . 15
12.4.2 Environment Variables . . . 15
12.4.3 Access Routine Semantics . . . 15
12.4.4 Local Functions . . . 16

13 MIS of Explanation Module **17**
13.1 Module . . . 17
13.2 Uses . . . 17
13.3 Syntax . . . 17
13.3.1 Exported Constants . . . 17
13.3.2 Exported Access Programs . . . 17
13.4 Semantics . . . 17
13.4.1 State Variables . . . 17
13.4.2 Environment Variables . . . 17
13.4.3 Access Routine Semantics . . . 17
13.4.4 Local Functions . . . 18

14 MIS of PyTorch Module **19**
14.1 Module . . . 19
14.2 Uses . . . 19
14.3 Syntax . . . 19
14.3.1 Exported Constants . . . 19
14.3.2 Exported Access Programs . . . 19
14.4 Semantics . . . 19
14.4.1 State Variables . . . 19
14.4.2 Environment Variables . . . 19
14.4.3 Assumptions . . . 19
14.4.4 Access Routine Semantics . . . 19
14.4.5 Local Functions . . . 20

15 MIS of PyTorch Geometric Module **21**
15.1 Module . . . 21
15.2 Uses . . . 21
15.3 Syntax . . . 21
15.3.1 Exported Constants . . . 21
15.3.2 Exported Access Programs . . . 21
15.4 Semantics . . . 21
15.4.1 State Variables . . . 21
15.4.2 Environment Variables . . . 21

15.4.3 Assumptions . . . 21
15.4.4 Access Routine Semantics . . . 21
15.4.5 Local Functions . . . 22

16 MIS of GUI Module **23**
16.1 Module . . . 23
16.2 Uses . . . 23
16.3 Syntax . . . 23
16.3.1 Exported Constants . . . 23
16.3.2 Exported Access Programs . . . 23
16.4 Semantics . . . 23
16.4.1 State Variables . . . 23
16.4.2 Environment Variables . . . 23
16.4.3 Assumptions . . . 23
16.4.4 Access Routine Semantics . . . 24
16.4.5 Local Functions . . . 24

3 Introduction

The following document details the Module Interface Specifications for Re-ProtGNN, a re-implementation of an interpretable Graph Neural Network (GNN) Framework.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at https://github.com/Yuanqi-X/Re-ProtGNN/tree/main.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 \,|\, c_2 \Rightarrow r_2 \,|\, \ldots \,|\, c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by Re-ProtGNN.

Data Type	Notation	Description
character	`char`	a single symbol or digit
Integer	$\mathbb{Z}$	A whole number in $(-\infty, \infty)$
Natural Number	$\mathbb{N}$	A positive integer in $[1, \infty)$
Real Number	$\mathbb{R}$	A real value in $(-\infty, \infty)$
Boolean	`bool`	Logical value: `True` or `False`
Vector of dimension d	$\mathbb{R}^d$	A d-dimensional real-valued vector
Matrix of size $n \times m$	$\mathbb{R}^{n \times m}$	A real-valued matrix with n rows and m columns
Index Vector	$\mathbb{N}^n$	A length-n vector of natural number indices
List of type T	`list`$[T]$	A finite sequence of values of type T
String	`str`	A sequence of characters
Dictionary	`dict[K` $\rightarrow$ `V]`	A mapping from keys of type K to values of type V
Tuple	`tuple`$[T_1$`,` T_2`,` `...]`	An ordered, fixed-length collection of elements where each element can have a different type.
Function	`Customized Function`	A user-defined function or callable

Functions in Re-ProtGNN are defined using their argument and return types. For instance, a function $f : \mathbb{N} \to \mathbb{R}$ takes a natural number and returns a real number. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding	
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

6 MIS of Configuration Module

6.1 Module

Configuration

6.2 Uses

None

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

None

6.4 Semantics

6.4.1 State Variables

- `data_args: DataParser` — Stores dataset-level configuration such as name, directory, splitting strategy, and seed.
- `model_args: ModelParser` — Stores GNN architecture settings and prototype-related parameters.
- `train_args: TrainParser` — Stores training hyperparameters including learning rate, batch size, and epoch count.
- `mcts_args: MCTSParser` — Stores Monte Carlo Tree Search and explanation-specific rollout parameters.
- `random_seed: int` — Stores the global seed used for generating random numbers.

6.4.2 Environment Variables

None

6.4.3 Access Routine Semantics

None - The state variables in this module are initialized when the system loads and are accessed directly by other modules using:

```
from utils.Configures import data_args, train_args, model_args, mcts_args
```

As such, no explicit accessor routines are exported.

6.4.4 Local Functions

DataParser(name: str, dir: str, split: list[$\mathbb{R}$], seed: int) $\rightarrow$ DataParser

- output: Returns a configuration object for dataset settings including `name`, `dir`, `split`, and `seed`.

ModelParser(model_name: str, hidden_dim: $\mathbb{N}$, num_prototypes: $\mathbb{N}$) $\rightarrow$ ModelParser

- output: Returns a configuration object containing the GNN model name, hidden dimension, and prototype count.

TrainParser(batch_size: $\mathbb{N}$, lr: $\mathbb{R}$, epochs: $\mathbb{N}$) $\rightarrow$ TrainParser

- output: Returns a configuration object with the training hyperparameters: `batch_size`, `lr`, and `epochs`.

MCTSParser(num_rollouts: $\mathbb{N}$, exploration_const: $\mathbb{R}$) $\rightarrow$ MCTSParser

- output: Returns a configuration object specifying the number of rollouts and exploration constant for MCTS-based explanation.

7 MIS of Input Format Module

7.1 Module

dataUtils

7.2 Uses

PyTorch Geometric Module (15), PyTorch Module (14), Configuration Module (6), Output Visualization Module (10)

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
load_dataset	-	`tuple[Dataset, int, int, dict[str → DataLoader]]`	`FileNotFoundError`, `ValueError`, `NotImplementedError`

7.4 Semantics

7.4.1 State Variables

None

7.4.2 Environment Variables

- `dataset_dir: str` — Filesystem path to the dataset root directory, obtained from `data_args.dataset_dir` defined in the Configuration Module.
- `log_file: str` — Path to the log file used by the `append_record()` routine exported from the Output Visualization Module.

7.4.3 Access Routine Semantics

load_dataset():

- transition:
  - Loads the dataset using `data_args.dataset_name` and `data_args.dataset_dir`, where `data_args` are defined in the Configuration Module.

- Logs the dataset name using `append_record(data_args.dataset_name)`, where `append_record()` is a routine exported from the Output Visualization Module

- output:
  - Returns a tuple: (`dataset`, `input_dim`, `output_dim`, `dataloader`) where:
    - `dataset`: graph dataset object loaded using `_get_dataset()`
    - `input_dim`: number of node features from `dataset.num_node_features`
    - `output_dim`: number of output classes from `dataset.num_classes`
    - `dataloader`: dictionary of DataLoaders split via `_get_dataloader()`

- exception:
  - `FileNotFoundError`: Raised if required dataset files are missing in the specified directory, such as missing raw '.pkl' or '.txt' files for the dataset.
  - `ValueError`: Raised if raw data files exist but are empty or malformed (e.g., missing node labels).
  - `NotImplementedError`: Raised if `data_args.dataset_name` does not match any supported dataset (i.e., not MUTAG, BA_2Motifs, or a MoleculeNet dataset).

7.4.4 Local Functions

_get_dataset(dataset_dir: str, dataset_name: str) → Dataset

- output: Selects an appropriate dataset loader based on `dataset_name` and returns the resulting dataset loaded from `dataset_dir`. See the Pytroch Geometric Module 15 for the type `Dataset`.

_get_dataloader(dataset: Dataset, batch_size: $\mathbb{N}$, data_split_ratio: list[$\mathbb{R}$]) → dict[str → DataLoader]

- output: Splits the input `dataset` into train/eval/test sets according to `data_split_ratio`, and returns DataLoaders batched by `batch_size`. See the PyTorch Geometric Module 15 for the type `DataLoader`.

8 MIS of Control Module

8.1 Module

main

8.2 Uses

Configuration Module (6), Input Format Module (7), Model Module (11), Training Module (9), Inference Module (12), Explanation Module (13), PyTorch Module (14)

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
main	clst: $\mathbb{R}$, sep: $\mathbb{R}$	-	-

8.4 Semantics

8.4.1 State Variables

None

8.4.2 Environment Variables

- `dataset_dir: str` — Filesystem path to the dataset root directory (from `data_args.dataset_dir`).
- `checkpoint_dir: str` — Directory path for saving and loading model checkpoints, constructed using `data_args.dataset_name`.
- `device: str` — Device identifier used by PyTorch for model training and inference (e.g., 'cpu' or 'cuda').

8.4.3 Access Routine Semantics

main(clst, sep):

- transition:
  - Loads the dataset and dataloaders using `load_dataset()`, which references `dataset_dir`.

- Initializes a GNN model and loss function using `setup_model(input_dim, output_dim, model_args)` from Model Module (11).
- Constructs `checkpoint_dir := './src/checkpoint/{data_args.dataset_name}/'`.
- Trains the model using `train(clst, sep, dataset, dataloader, gnnNets, output_dim, criterion, checkpoint_dir)` from Training Module (9).
- Loads the best checkpoint from `checkpoint_dir`, and updates model weights using `update_state_dict()`.
- Evaluates the trained model via `test(dataloader['test'], gnnNets, criterion)` from Inference Module (12).
- Generates explanations using `exp_visualize(dataset, dataloader, gnnNets, output_dim)` from Explanation Module (13).

- output: None
- exception: None

8.4.4 Local Functions

None

9 MIS of Training Module

9.1 Module

Train

9.2 Uses

Configuration Module (6), Model Module (11), Explanation Module (13), Output Visualization Module (10), PyTorch Module (14)

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
train	clst: $\mathbb{R}$, sep: $\mathbb{R}$, dataset: `Dataset`, dataloader: `dict[str` $\rightarrow$ `DataLoader]`, gnnNets: `GnnNets`, output_dim: $\mathbb{N}$, criterion: `Customized Function`, ckpt_dir: `str`	-	None

9.4 Semantics

9.4.1 State Variables

None

9.4.2 Environment Variables

- `checkpoint_dir: str` — Path to the directory for saving model checkpoints.
- `device: str` — Target computation device, used to allocate model weights and prototype vectors.

9.4.3 Access Routine Semantics

train(clst, sep, dataset, dataloader, gnnNets, output_dim, criterion, ckpt_dir):

- transition:

- Initializes the optimizer using parameters from `gnnNets` and `train_args`.
- Logs statistics for `dataset` using `_log_dataset_stats(dataset)`.
- Iteratively trains the model using batches from `dataloader['train']` with cluster/separation losses weighted by `clst` and `sep`.
- Periodically projects prototypes onto embedding space using `_project_prototypes(gnnNets, dataset, ...)`.
- Evaluates performance on the validation set using `_evaluate(dataloader['eval'], gnnNets, criterion)`.
- Saves model checkpoints to `ckpt_dir`.

- output: None
- exception: None

9.4.4 Local Functions

_evaluate(eval_dataloader: DataLoader, model: GnnNets, criterion: Customized Function) → dict[str → float]

- transition: None
- output: Runs model evaluation on `eval_dataloader` and computes loss/accuracy basing on `criterion`. Returns a dictionary with keys `'loss'` and `'acc'`.

_log_dataset_stats(dataset: Dataset) → None

- transition: Computes average number of nodes and edges from `dataset`, and prints the result.
- output: None

_project_prototypes(model: GnnNets, dataset: Dataset, indices: list[ℕ], output_dim: ℕ) → None

- transition: Updates each prototype vector in `model` with a real example from `dataset` using `get_explanation()` from Explanation Module (13).
- output: None

10 MIS of Output Visualization Module

10.1 Module

outputUtils

10.2 Uses

PyTorch Module (14), PyTorch Geometric Module (15), GUI Module (16)

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
ExpPlot	dataset_name: `str`	`ExpPlot`	-
draw	graph: `networkx.Graph`, nodelist: `list[int]`, figname: `str`, kwargs: `dict`	-	`NotImplementedError`
append_record	info: `str`	-	`FileNotFoundError`
save_best	ckpt_dir: `str`, epoch: $\mathbb{N}$, gnnNets: `GnnNets`, model_name: `str`, eval_acc: $\mathbb{R}$, is_best: `bool`	-	-

10.4 Semantics

10.4.1 State Variables

None

10.4.2 Environment Variables

- `log_file: str` — Hardcoded path to the log file: `./results/log/hyper_search`.
- `device: str` — Computation device (e.g., 'cuda' or'cpu') used to store model after saving.

10.4.3 Access Routine Semantics

ExpPlot(dataset_name):

- transition: None
- output: Constructs an object for drawing explanations for `dataset_name`. Returns an `ExpPlot` object.
- Note: Please see in PyTorch Geometric Module (15) for the type `networkx.Graph`.

draw(graph, nodelist, figname, kwargs):

- transition: Calls the drawing routine and uses GUI Module (16) to generate and save a figure to `figname`.
- output: None
- exception: `NotImplementedError` if `dataset_name` is unsupported.

append_record(info):

- transition: Writes `info` as a new line to the file located at `log_file`.
- output: None
- exception: `FileNotFoundError` if the parent directory of `log_file` does not exist.

save_best(ckpt_dir, epoch, gnnNets, model_name, eval_acc, is_best):

- transition:
  - Saves model weights and training metadata to `ckpt_dir`.
  - If `is_best=True`, copies this file to `ckpt_dir`.
  - Moves model between '`cpu`' and `device := model_args.device`.
- output: None
- exception: None

10.4.4 Local Functions

None

11 MIS of Model Module

11.1 Module

GnnNets

11.2 Uses

PyTorch Module (14), Output Visualization Module (10)

11.3 Syntax

11.3.1 Exported Constants

None

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
setup_model	input_dim: $\mathbb{N}$, output_dim: $\mathbb{N}$, model_args: `dict`	tuple[`GnnNets`, `Customized Function`]	-

11.4 Semantics

11.4.1 State Variables

None

11.4.2 Environment Variables

- `device: str` — Target device (e.g., 'cuda' or 'cpu'), used to move the model after initialization.
- `log_file: str` — Path to the log file used in `append_record()`.

11.4.3 Access Routine Semantics

setup_model(input_dim, output_dim, model_args):

- transition: Instantiates a GNN model with specified `input_dim`, `output_dim`, and `model_args`. Moves the model to `device`. Writes the model name to `log_file` using `append_record()`.
- output: A tuple containing:

- `gnnNets`: a model object supporting GNN forward/inference
- `criterion`: a cross-entropy loss function

- exception: None

11.4.4 Local Functions

None

12 MIS of Inference Module

12.1 Module

evaluation.inference

12.2 Uses

Model Module (11), Output Visualization Module (10), PyTorch Module (14)

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
run_inference	test_dataloader: `DataLoader`, model: `GnnNets`, criterion: `Customized Function`	tuple[dict[str $\to \mathbb{R}$], $\mathbb{R}^{n \times c}$, $\mathbb{N}^n$]	None

12.4 Semantics

12.4.1 State Variables

None

12.4.2 Environment Variables

- `log_file: str` — Path to the log file where final test performance is recorded via `append_record()`.

12.4.3 Access Routine Semantics

run_inference(test_dataloader, model, criterion):

- transition:
  - Appends the final test loss and accuracy to the `log_file` using `append_record()`.
- output:
  - Returns a tuple `(test_state, all_probs, all_preds)`:

    - ∗ `test_state`: `dict[str` $\rightarrow \mathbb{R}$`]` containing keys `'loss'` and `'acc'`.
    - ∗ `all_probs`: $\mathbb{R}^{n \times c}$ — class probability matrix for n test samples and c classes, obtained by passing the data in `test_dataloader` into `model`.
    - ∗ `all_preds`: $\mathbb{N}^n$ — vector of predicted class labels.
- exception: None

12.4.4 Local Functions

None

13 MIS of Explanation Module

13.1 Module

Explanation

13.2 Uses

Configuration Module (6), Output Visualization Module (11), PyTorch Geometric Module (15), PyTorch Module (14)

13.3 Syntax

13.3.1 Exported Constants

None

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
get_explanation	data: `Data`, gnnNet: `GnnNets`, prototype: $\mathbb{R}^d$	tuple[list[$\mathbb{N}$], $\mathbb{R}$, $\mathbb{R}^d$]	None
exp_visualize	dataset: `Dataset`, dataloader: `dict[str` $\rightarrow$ `DataLoader]`, gnnNets: `GnnNets`, output_dim: $\mathbb{N}$	-	`FileNotFoundError`

13.4 Semantics

13.4.1 State Variables

None

13.4.2 Environment Variables

- `save_dir: str` — Filesystem path for saving explanation plot images.

13.4.3 Access Routine Semantics

get_explanation(data, gnnNet, prototype):

- transition: None
- output:

- Uses `data`, `gnnNet`, and `prototype` to invoke a Monte Carlo Tree Search (MCTS) via the local function `_mcts()`.
- MCTS explores subgraph coalitions within the graph structure of `data` and evaluates their similarity to the provided `prototype` using `_prot_similarity_scores()`.
- Returns:
  - `coalition`: list of node indices selected as the best explanation subgraph.
  - `P`: similarity score ($\in \mathbb{R}$) between the selected subgraph and the `prototype`.
  - `embedding`: a masked subgraph embedding $\in \mathbb{R}^{1\times d}$ extracted by applying `gnnNet` to the subgraph induced by `coalition`.

- exception: None

exp_visualize(dataset, dataloader, gnnNets, output_dim):

- transition:
  - Computes $K :=$ `output_dim` $\times$ `model_args.num_prototypes_per_class` random prototype vectors of dimension d, fixed for visualization.
  - Samples 16 graphs from `dataloader['train']`.
  - For each selected graph and for each of the first 10 prototype vectors, calls `get_explanation(data, gnnNets, prototype)` to compute explanations.
  - Saves the explanations to image files in the environment variable `save_dir`.
  - Sets `save_dir := './results/plots/<dataset>_<model>_'` using values from `data_args.dataset_name` and `model_args.model_name`, imported from Configuration Module (6).
- output: None
- exception:
  - `FileNotFoundError`: if `save_dir` cannot be created due to missing parent directories, invalid paths, or insufficient filesystem permissions.

13.4.4 Local Functions

_mcts(data: Data, gnnNet: GnnNets, prototype: $\mathbb{R}^d$) $\rightarrow$ tuple[list[$\mathbb{N}$], $\mathbb{R}$, $\mathbb{R}^d$]

- output: Applies a multi-step rollout procedure to search for the optimal node coalition in `data`, maximizing similarity to the reference vector `prototype` $\in \mathbb{R}^d$. Returns the selected node indices, similarity score, and final embedding.

_prot_similarity_scores(coalition: list[$\mathbb{N}$], data: Data, gnnNet: GnnNets, prototype: $\mathbb{R}^d$) $\rightarrow \mathbb{R}$

- output: Computes a scalar similarity score $\in \mathbb{R}$ between the subgraph embedding of nodes in `coalition` (produced by `gnnNet`) and the given reference vector `prototype` $\in \mathbb{R}^d$, based on squared Euclidean distance.

14 MIS of PyTorch Module

14.1 Module

Torch

14.2 Uses

Hardware-Hiding Module

14.3 Syntax

14.3.1 Exported Constants

None

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
Tensor	shape: `list[int]`, dtype: `str`	`Tensor`	-
cross_entropy	logits: `Tensor`, labels: `Tensor`	`Tensor`	-
Adam	parameters: `list[Tensor]`, lr: $\mathbb{R}$	`Optimizer`	-

14.4 Semantics

14.4.1 State Variables

None

14.4.2 Environment Variables

None

14.4.3 Assumptions

None

14.4.4 Access Routine Semantics

Tensor(shape, dtype):

- output: Returns a tensor initialized with zeros of the given `shape` and `dtype`.

cross_entropy(logits, labels):

- output: Computes the cross-entropy loss between `logits` and `labels`.

Adam(parameters, lr):

- output: Returns an Adam optimizer configured with the given `parameters` and learning rate `lr`.

14.4.5 Local Functions

None

15 MIS of PyTorch Geometric Module

15.1 Module

PyG

15.2 Uses

PyTorch Module (14), Hardware-Hiding Module

15.3 Syntax

15.3.1 Exported Constants

None

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
Data	x: `Tensor`, edge_index: `Tensor`	`Data`	-
MoleculeNet	root: `str`, name: `str`	`Dataset`	`FileNotFoundError`
DataLoader	dataset: `Dataset`, batch_size: $\mathbb{N}$	`DataLoader`	-
to_networkx	data: `Data`	`networkx.Graph`	-

15.4 Semantics

15.4.1 State Variables

None

15.4.2 Environment Variables

None

15.4.3 Assumptions

None

15.4.4 Access Routine Semantics

Data(x, edge_index):

- output: Constructs and returns a PyG graph object using `x` as node features and `edge_index` as edge indices.

MoleculeNet(root, name):

- output: Loads the dataset specified by `name` from directory `root` and returns a `Dataset` object.
- exception: `FileNotFoundError` if `root` does not exist.

DataLoader(dataset, batch_size):

- output: Returns a `DataLoader` that batches data from the given `dataset` with batch size `batch_size`.

to_networkx(data):

- output: Converts the input PyG `data` object into a NetworkX graph.

15.4.5 Local Functions

None

16 MIS of GUI Module

16.1 Module

Matplotlib

16.2 Uses

Hardware-Hiding Module

16.3 Syntax

16.3.1 Exported Constants

None

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
axis	axis_choice: `str`	-	-
title	title_sentence: `str`	-	-
save_fig	figname: `str`	-	`FileNotFoundError`
close	choice: `str`	-	-

16.4 Semantics

16.4.1 State Variables

None

16.4.2 Environment Variables

- `figure_path: str` — Path where the current figure will be saved.
- `axis_visible: bool` — Whether axes are displayed in the active figure.
- `figure_title: str` — Title of the current figure.
- `figure_open: bool` — Whether there are any open figures.

16.4.3 Assumptions

None.

16.4.4 Access Routine Semantics

axis(axis_choice):

- transition: If `axis_choice == 'off'`, sets `axis_visible := False` and disables axes using `plt.axis('off')`. Otherwise sets `axis_visible := True`.

title(title_sentence):

- transition: Sets `figure_title := title_sentence` and updates the title of the current figure using `plt.title()`.

save_fig(figname):

- transition: Sets `figure_path := figname` and saves the current figure to the specified path using `plt.savefig(figname)`.
- exception: `FileNotFoundError` if `figname` refers to a non-existent directory.

close(choice):

- transition: Closes all active figure windows using `plt.close(choice)` and sets `figure_open := False`.

16.4.5 Local Functions

None

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.