RUHR-UNIVERSITÄT BOCHUM

RUB

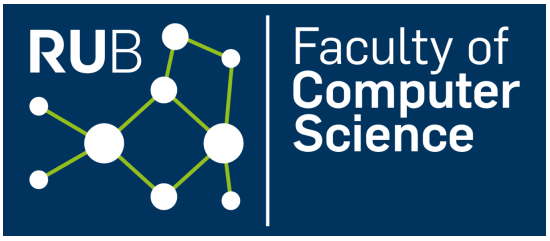
Development of a DL-Based Classifier for Software Repositories

Constanze Ohrem, Student Number: 108017221367

RUHR-UNIVERSITÄT BOCHUM

Bachelor's Thesis – August 7, 2024
Chair of Software Engineering.

Supervisor: Prof. Dr. Thorsten Berger
Advisor: Yorick Sens and Henriette Knopp



Abstract

Mining studies in software engineering are conducted to learn more about the characteristics of software systems. The goal of these studies is to gain more understanding about development practices and associated problems. An essential part of mining studies is to mine data from real-world software systems and to identify the type of software one wants to analyze. A classification of the software systems is needed. This classification can be very challenging. We propose to classify a software system by analyzing the structure of its source code. Our proposed tool contains a pipeline for converting the structure of the source code in the software systems into a graph and piping it into a graph convolutional network for classification. We create a labeled dataset with software systems mined from GitHub and define labels for classification. The tool could be extended to solve other classification problems in the future.

Erklärung

Als öffentlich finanzierte Forscher wollen wir durch die Veröffentlichung dieser Arbeit einen Beitrag zur Forschungsgemeinschaft leisten, damit andere Forscher dieses Wissen nutzen und weiterentwickeln können. Ich bin damit einverstanden, dass meine Dissertation dauerhaft und öffentlich gespeichert wird. Sie kann dann z. B. von Google Scholar indexiert werden, so dass sie von anderen Forschern gefunden und referenziert werden kann.

Consent

As publicly funded researchers, we aim to contribute to the research community by publishing this work, so other researchers can use and further develop this knowledge. I agree that my thesis will be permanently and publicly stored. It can then be indexed by, for instance, Google Scholar so that it can be found and referenced by other researchers.

07.08.2024 — C. Ohrem

DATE — CONSTANZE OHREM

Contents

1 Introduction

In software engineering, mining studies are focused on mining software artifacts in order to gather information on the characteristics of software systems [25]. They are closely related to the field of data mining, which is the process of searching and analyzing large amounts of raw data with the goal of identifying patterns and extracting information [11] [37]. Mining studies in software engineering are conducted to learn more about development practices and problems. For this purpose, data is mined from real-world software systems.

To understand the characteristics of software, we need to determine which context the software is for. There are diverse types of software, ranging from toy projects to applications, and libraries, among many other types. Depending on the type of software, we expect to find different architectures in the source code.

1.1 Motivation

In order to find software systems that fit the research purpose of a mining study one wants to conduct, we need to be able to filter the software systems based on their types. A classification of the software systems is needed.

Classifying software systems manually, which is how it is often done, poses a challenge. It requires a lot of time and effort. We propose a supervised deep learning (DL)-based approach to automate classification. For this purpose, we focus on analyzing the structure of the source code. We store the code structure in an Ecore metamodel (EMM) and classify it with a graph convolutional network (GCN).

1.2 Related Work

Previous studies focus on various aspects of repositories, which contain the mined software systems. An important source for real-world software systems is GitHub because researchers often do not have access to closed-source software systems from the industry. GitHub is currently one of the largest collaborative code hosting sites on the Internet.

1.2.1 Mining and Analysis

As Kalliamvakou et al. [23] already stated, previous qualitative studies have focused on how developers use GitHub's social features to assess success, performance, and possible collaboration opportunities. Furthermore, Kalliamvakou et al. underlines the importance of first establishing that the data mined from GitHub fits the research purpose. This motivates the need for a classification of software systems.

Nahar et al. [28] conducted research on machine learning (ML) products from GitHub. The products are analyzed to learn more about development practices in machine learning. They define ML products and propose a search strategy for such products on GitHub. After retrieving the ML products, Nahar et al. classifies them manually. A classifier for software systems could be applied to this study to help select data for this research.

1.2.2 Classifier for Software Systems

Soll and Vosgerau [33] use an ensemble learning approach to tackle the hard task of classifying software systems. These systems are mined from GitHub. Soll and Vosgerau achieve a precision of 59.90% and a recall of 58.41% for their dataset. In ensemble learning, multiple weak classifiers are combined to create a strong one. These include, for example, a classifier for readme files and a classifier for the metadata of repositories.
The dataset used in Soll and Vosgerau's student competition consists of repositories that were picked at random and classified by hand. They used this dataset both for training and evaluation [33].
Opposed to Soll and Vosgerau's approach, we only rely on the source code for classification.

1.2.3 Dataset

While studies on the classification of software systems exist, they are mostly conducted on small datasets created by the researchers of these studies. The analysis is mostly done with manual inspection. The result is only a small number of software systems being examined.

Idowu et al. [22] focus on ML-related Python projects. They provide a dataset of these projects. The dataset was created partially with manual inspection to identify and define characteristics of ML-related projects implemented in Python. This study is another example for a possible application of a classifier for software systems.

Dang et al. [15] also provides a dataset of classified GitHub repositories. They

gathered 15,262 software systems to evaluate their proposed tool. They achieve good results for their dataset.

A classification method on large datasets for arbitrary software systems is still missing [22].

1.3 Contribution

The contribution of this thesis is a tool based on supervised deep learning for classifying arbitrary software systems.
The tool consists of a pipeline with two converters, that parse and convert the structure of source code first into an Ecore metamodel, then into a matrix structure. The matrix structure is then fed into a graph convolutional network. The GCN can detect features in the source code, based on which the tool classifies the software systems. Additionally, we create a dataset of classified software systems from GitHub to train and evaluate our tool. The tool has a modular design, which makes it easy to extend and to reuse its components for other problems and research questions, which involve analyzing the structure of source code.

1.4 Organization of this Thesis

This thesis provides background information on an Ecore metamodel for storing the structure of source code and the general structure of a graph convolutional network in Chapter 2. It also provides information on evaluation metrics for the GCN in that chapter.
It then explains the key aspects of our proposed tool and its pipeline in Chapter 3, before discussing results both from training our tool with our dataset and from training with a different dataset to demonstrate its reusability for different problem statements in Chapter 4.

2 Background

Our proposed tool parses a software system's code structure into an abstract syntax tree (AST) and converts the tree into an Ecore metamodel, before classifying it with a GCN. In this chapter, we provide background information on the aforementioned data structures and utilities needed for our tool.

2.1 Abstract Syntax Tree

An abstract syntax tree is a tree-like, hierarchical representation of the syntactic structure of program code.
Therefore, the structure of an AST depends on which programming language is used in the code and follows its grammar rules. The abstract syntax tree of Python code is different from the AST of, for example, Java code.
It can help gain a clear understanding of the code's structure and is useful for code analysis. Abstract syntax trees provide a structured and standardized representation of code [5].

When parsing code into an AST, the program is broken down into its constituent parts, such as keywords, identifiers, literals, and operators [5]. The main program is the root node of the tree with the program elements as child nodes. The program elements we focus on are modules, class, and function definitions, function calls, function parameters, and imports of modules, classes, and functions. The relationships between these program elements are represented by the hierarchical structure in the AST [5].

2.1.1 AST Node Types

There are many different types of nodes in an abstract syntax tree. We focus on the ones we use in our tool. The same applies to each node's attributes or contained information.

A ClassDef node in an abstract syntax tree represents a class definition. The node stores the name of the defined class as a raw string, a list of nodes for explicitly specified base classes, and a list of nodes representing the code inside the class [20].

The node type inside the class we focus on are FunctionDefs.

FunctionDefs are nodes that represent function definitions. This node type also stores the function name as a raw string. It contains a node with the arguments of the function [20].
Like the code structure that the node represents, a FunctionDef node can be defined inside of a class, as well as outside of a class.

An Import node represents an import statement and stores the name of the imported program entity, along with a list of alias nodes.
ImportFrom nodes represent 'from x import y' statements. 'x' is the module, which we import from. 'y' is the name of the class, or method, which we import with this statement [20].

Another type of nodes in an abstract syntax tree are Assign nodes. They represent assignments of values. The node holds a list of nodes that represent the instances in the source code that a value is being assigned to.
The same value can be assigned to multiple nodes in the list. The value that is assigned is stored as a single node in the Assign node [20].

Call nodes represent function calls in an abstract syntax tree. The name of the called function, also referred to as the target, is stored in the node. It is often a Name or an Attribute object. The node also holds a list of the arguments that are passed on by position, and a list of arguments passed on by keyword. Both lists can be empty [20].

2.1.2 AST Example

An example of an abstract syntax tree can be seen in Figure 2.1. As shown in Figure 2.1, a single Python file is represented as a module in the AST. It is the root node, with the function call as its child node. In the example the function 'print' is used. The call contains the name of the function that is being called and the parameter, in the example it is the string 'hello world', that is passed on when executing the call.

2.1.3 Generating ASTs

There are two fundamental steps in creating ASTs. The first step is the lexical analysis, also called Tokenization. This step focuses on the different components of the source code. A token can be any entity in the programming language. It can be a literal with a fixed value, a variable, an operator, or a function call.
The second step is the syntax analysis, also called Parsing. The list of tokens is

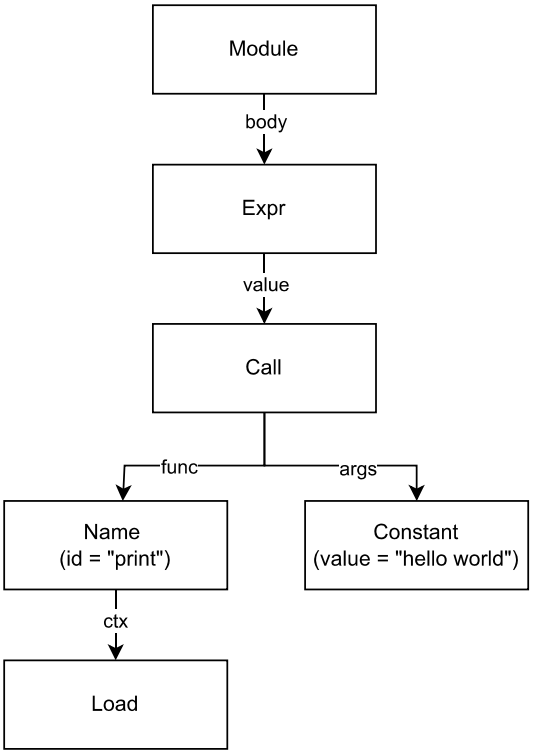


Figure 2.1: AST of a Python file with code "print('hello world')" [32]

turned into a tree structure representing the source code. This structure is the abstract syntax tree. It reflects the types of relationships between the tokens. If the node represents a function call, for example, the arguments and return values are the associated data that are contained in the node's children [9] [27].

2.2 Metamodeling

Metamodels are models that make statements about other models. On an abstraction scale, metamodels are one level above models. Every metamodel has its own modeling language that describes the possible structure of a model, which can be defined with this metamodel's language. It defines the model's abstract syntax [34]. An abstract syntax defines how a design can be decomposed into interconnected components, without being concerned with how a design is represented on paper or in a computer file [35]. Metamodels and models have a class-instance relationship. A model defined with a metamodel's language is a concrete instance of that metamodel [34].

Since we obtain a higher level of abstraction with metamodels, we can gain independence from the programming language used in the source code.

2.2.1 Eclipse Modeling Framework/Ecore

The Eclipse modeling framework (EMF) is a modeling framework and common standard for data models. The framework provides a metamodel, Ecore, for describing models [1]. Ecore is a language for defining metamodels.

2.2.2 Type Graph

The Ecore metamodel we use in our approach is introduced by Peldszus [29].
Peldszus also provides a tool for visualization, called GRaViTY. It can be found on GitHub [1]. The Ecore metamodel provides a high-level abstraction from source code for structural entities of object-oriented programs [29]. The metamodel, called type graph, is saved in XMI format and serves as a container for program elements. A visualization of the type graph structure can be seen in Figure 2.2.

The nodes in this metamodel represent first-class program entities, while the edges define a relation between these entities. There are different types of relations [29]. The root node is the type graph. From the root node, the packages, modules, classes, and methods contained in the type graph can be accessed directly via the same name attributes.

Object-oriented programs are structured into namespaces. They are useful for structuring programs hierarchically. In procedural languages these namespaces are simulated with naming patterns. Namespaces are represented by the node TPackage in the type graph. The package name is stored in the node. The hierarchy of packages is represented by two relations: parent- and subpackages. A TPackage node can contain an arbitrary number of classes and modules. The classes and modules are defined within the namespace of the package [29].

The node TModule represents modules in a program. They have a relation to the abstract node TContainableElement. The abstract node TAbstractType implements TContainableElement. Therefore, TModule can contain any arbitrary node type implementing TAbstractType [29].

There are two different kinds of TAbstractTypes in the type graph. One is TClass and the other one is TInterface.
TInterface represents interfaces and TClass represents concrete classes defined in a program. Since the concept of interfaces is not supported in Python, this node type

[1] https://github.com/GRaViTY-Tool/gravity-tool

is not used in our current tool. Inheritance between classes is represented by the relations of parent- and child classes. They are important for capturing a program's structure. Multiple inheritance in the type graph is possible to be as general as possible [29]. The class name is also stored in the TClass node.

The node TAbstractType defines an arbitrary number of TMember instances. TMember is an abstract node itself.
TMethodDefinition nodes implement TMember. They represent method definitions in a program. Methods in a program are split into multiple nodes in the type graph. Additionally to TMethodDefinition nodes, there are TMethod and TMethodSignature nodes. The TMethod node is connected to both its TMethodSignature node and to its TMethodDefinition node.
The method name is stored in the TMethod node. TMethodSignature has another relation to the node TParameter. TParameter represents parameters defined within a method. Since a method can have multiple parameters, the TParameter node has two relations, 'next' and 'previous', to connect multiple TParameter nodes to each other and to their method [29].

The abstract node TAccess has a relation to TMember. The node TCall implements TAccess and represents method calls in a program.
If a method calls another method in a program, the TMethodDefinition node representing the calling method is often referred to as the source of the call. The TMethodDefinition node representing the called method is often referred to as the target of the call. Both TMethodDefinition nodes are connected by a TCall node [29].

2.2.3 PyEcore Framework

The PyEcore framework is a model driven engineering (MDE) framework. It is an implementation of EMF/Ecore for Python and can be used to handle models and metamodels [2]. It is also compatible with the original Java implementation of EMF. The framework provides functionality for XMI serialization and deserialization, importing existing metamodel resources, and navigating in a model among other functions.

2.2.4 XML Metadata Interchange

XML metadata interchange (XMI) is a format specification that allows the interchange of objects and models through an XMI formatted file [14].

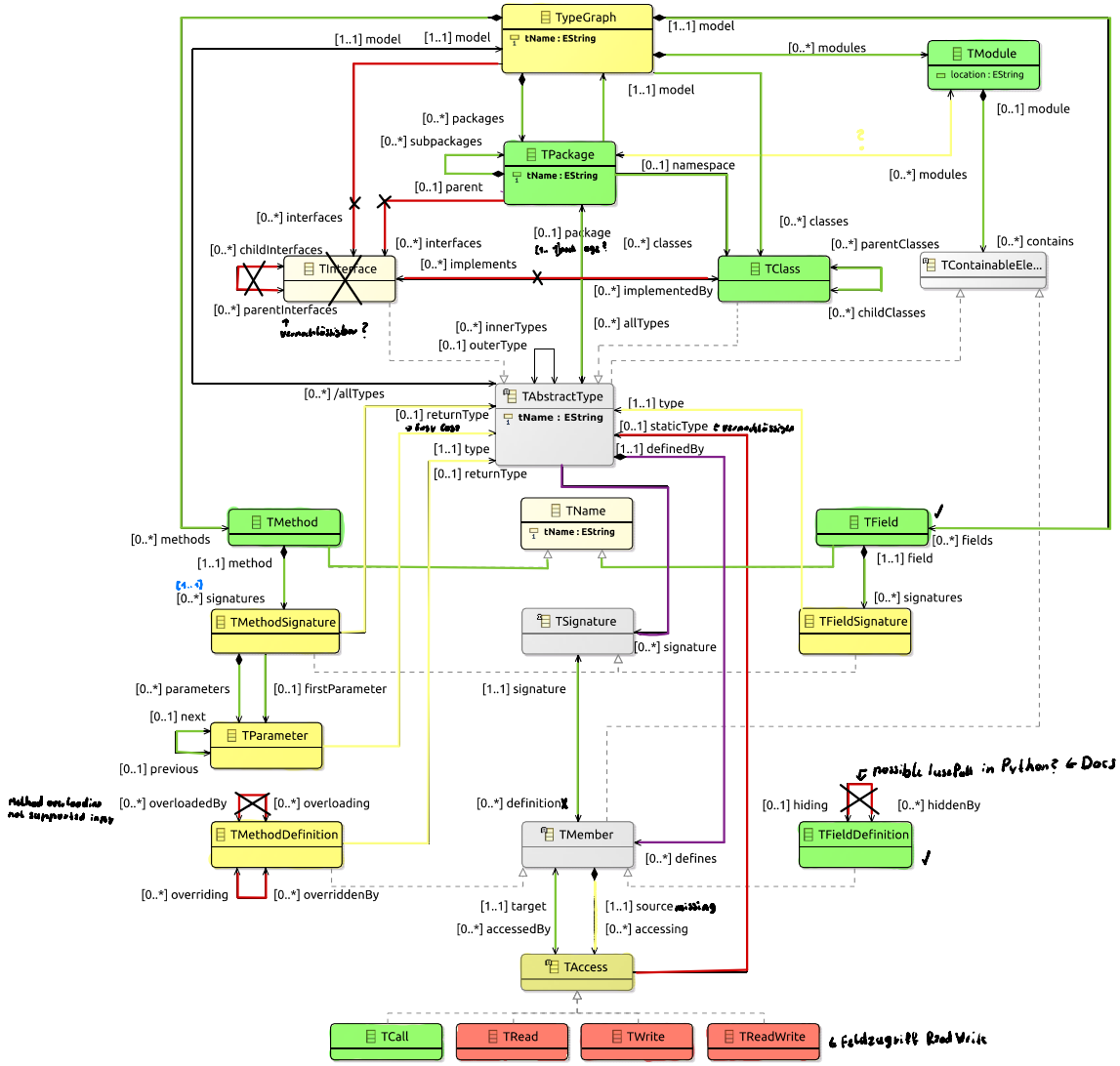


Figure 2.2: GRaViTY's metamodel for language independent object-oriented program models [29]

2.3 Multi-Label Classification

Labels for a dataset may not be mutually exclusive. That means multiple labels may be assigned to a single data instance in a dataset and multiple labels may be predicted for a single data input when performing a classification task.
This is called multi-label classification (MLC) [36].

2.4 Encoding

Datasets can contain text or non-numerical values. However, deep learning algorithms work better with numerical input. A conversion of non-numerical into numerical input is therefore necessary. There are multiple ways to encode data, each with its own advantages and disadvantages [38].

2.4.1 One-Hot Encoding

If the values have no hierarchy or order to them, one-hot encoding (OHE) is a good way to encode the data. With other encoding techniques, such as label encoding, the data could be misinterpreted by the DL algorithm. This ordering issue does not occur with one-hot encoding [38].

When used to encode labels for a dataset in deep learning, one-hot encoding maps each label to a binary vector [6]. Every category value in the dataset, that needs to be encoded, is converted into one column and is assigned a 1 for True or a 0 for False. The data encoded with OHE is represented in rows. If the data in one row fits in a category, the value in this row is assigned a 1 in the column of the category it fits into [38].

For example, if we want to encode the labels 'Application', 'Framework', and 'Library', their respective encodings are:

- Application: [1.0, 0.0, 0.0]
- Framework: [0.0, 1.0, 0.0]
- Library: [0.0, 0.0, 1.0]

A disadvantage of using OHE is the additional data overhead. With few category values, the overhead is manageable, but the more categorical values are in a dataset, the more dimensionality is added to the dataset with the encoding [38].

2.4.2 Multi-Hot Encoding

Multi-label classification requires special handling when encoding the labels. Multi-hot encoding (MHE) is a binary encoding of multiple tokens into a single vector [17]. There is no standard term for this type of encoding. We use the term multi-hot encoding, however, one may also refer to it as multi-label binarization [16]. MHE works like one-hot encoding, but supports multiple labels being assigned to one instance in a dataset. We use the example from OHE, but the labels are now not mutually exclusive. For example, if we assign the labels 'Application' and 'Library' to one sample, and the labels 'Framework' and 'Library' to a second one, their resulting MHE label vectors are:

- sample one: [1.0, 0.0, 1.0]
- sample two: [0.0, 1.0, 1.0]

2.5 Hashing

Hashing is the conversion of a string of characters into a fixed-size value by using a mathematical hash function. The output of the hash function is often in hexadecimal format. Hashing in data structures can be used to efficiently store data [39].

2.5.1 MD5

The message-digest algorithm 5 (MD5) is a hash algorithm that transforms input into a 128 bit hash. There are implementations readily available in libraries, like the Python 'hashlib' library [12].

2.6 Graph Convolutional Network

Graphs are very versatile data structures and useful for modeling real-world problems. There are multiple types of classification tasks when working with graphs, such as node classification. The one we perform is graph classification [10]. In graph classification, labels, either a single label or multiple, are assigned to the entire graph.
When facing classification problems, convolutional neural networks (CNNs) have proven to be a very powerful tool.
CNNs are very efficient at extracting features from complex data. A graph convolutional network is a special type of convolutional neural network, that operates on graphs. The graph is used as input and passed through the layers of a neural

network [10].
The graph is stored in two matrices. The nodes are stored in an $|N|xC$ matrix, for N nodes with C features. The edge information is contained in a sparse adjacency matrix with dimensions $|E|x2$, where E is the number of edges in the graph. The sparse adjacency matrix has the format $[node\_id_1, node\_id_2]$. GCNs apply filters to the graph to learn from its features.
The basic structure of a graph convolutional network is shown in Figure 2.3.

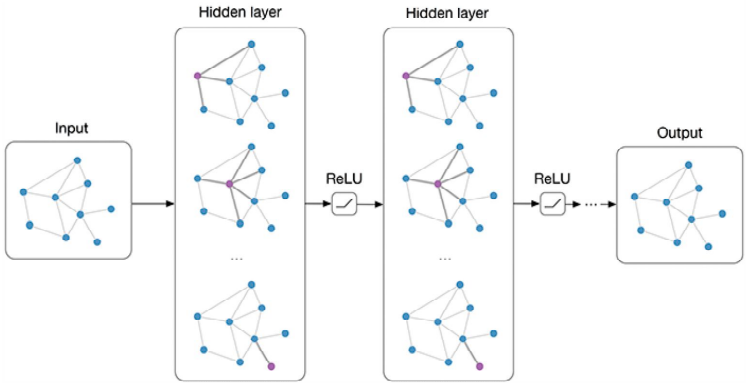


Figure 2.3: graph convolutional network with a series of alternating hidden layers and activation layers [26]

2.6.1 Batching

Batching is a technique for scaling the training of a deep learning model to huge amounts of input data [13]. Multiple samples, in our case graphs, are combined to be used as input at once.
Instead of taking one sample at a time, passing it through the layers of a GCN, and updating the model's weights based on the results, the entire batch is passed as input. Thus, the weights have to be updated less frequently, efficiently optimizing computation for the model. The number of samples in a batch is determined by the batch size.

Since graphs can hold any number of nodes, the sparse adjacency matrices are stacked diagonally resulting in a large graph consisting of multiple isolated subgraphs. The node feature matrices are concatenated in the node dimension [13].

2.6.2 GCN Layers

The layers of a graph convolutional network are a series of hidden layers using non-linear activation functions. Each hidden layer consists of a set of neurons. Each neuron is fully connected to all neurons of the previous layer [31]. The output of the

last layer is the final classification of the network. There are many different types of layers that can be used as hidden layers in a neural network.

The convolutional layers are where the filters are applied. The filters multiply neurons with weights to learn from data features. In graph convolutional networks, the features are learned from neighbouring nodes [3]. Therefore, the relations between nodes is important. The convolutional layers make up most of the hidden layers in the neural network.

An activation layer with a non-linear activation function, for example ReLU, is applied after every convolutional layer [31]. ReLU, or rectified linear unit, is a linear function that for every positive input value directly outputs the same value and for every negative input value outputs zero [7].

Pooling layers are used to reduce the size of the input. By reducing the input size, we can also reduce the number of parameters and therefore the amount of computation needed in a neural network [31].

Dropout layers are used for regularization to prevent the model from overfitting. Overfitting happens when a machine learning model models the training data too well. This results in the model performing badly on new, previously unseen data [8]. For every input, the dropout layer with probability p randomly disconnects input from the previous layer to the next layer. This reduces overfitting by explicitly altering the network architecture during training [31].

2.6.3 K-Fold Cross Validation

K-fold cross validation is a resampling technique in machine learning for evaluating the performance of a ML model. The dataset is split into k folds. The model is trained on $k - 1$ folds and evaluated on the remaining fold. The process is repeated k times [4]. With this technique, the model is trained and tested on every sample of the dataset at least once. This increases the amount of data the model is trained with, without using a larger dataset.

2.6.4 Functions for Training

In order to make predictions, the machine learning model has to learn from the given input data. After every training step, the output of the model is compared to the ground-truth labels, based on which the model's weights and parameters are updated so the model's predictions are closer to the ground-truth. This comparison is done by the loss function. It measures how well the GCN models the training data [40].

The updating of the model's weights and parameters is performed with an optimizer, which is a function. It updates the weights to reduce the loss during training, which improves the model's accuracy [18].

2.6.5 Evaluating Training Results

In multi-label classification, the accuracy is not a good metric to evaluate a ML model with [24]. It returns value 1.0, if all of the predicted labels of the output from the model match the ground-truth labels, and 0.0 if not.

Better metrics to evaluate a multi-label model with are the precision, the recall, and the f1-score. Intuitively, precision is the ability of the model not to label a sample as positive when it is negative. It is the ratio $\frac{tp}{tp+fp}$, where tp is the number of true positives and fp is the number of false positives [24].

Recall is the ability of the classifier to find all positive samples in a dataset. It is the ratio $\frac{tp}{tp+fn}$, where fn is the number of false negatives. tp stays the same as with the calculation of the precision [24].

The f1-score is the weighted average of precision and recall [24].

In order to evaluate a model with these metrics, one has to look at the development of their values during training. If the values of the f1-scores converge to a fixed value, that means the training process is reaching a stable state [30]. The model can make accurate predictions.

2.6.6 Comma-Separated Values

A comma-separated values (CSV) file is a simple text file, that stores data separated by commas [21].

2.6.7 PyTorch Framework

The PyTorch framework is an open-source machine learning framework based on Python and the torch library. It can be used to implement deep neural networks. PyTorch Geometric (PYG) is a library from PyTorch for deep learning on irregular input data, such as graphs [19].

3 Implementation

Our goal is to automate classification of arbitrary software systems by extracting patterns from the systems' source code. For this purpose, we build a tool with a pipeline that stores the source code structure of the software systems in Ecore metamodels and pipes the EMMs into a graph convolutional network for classification. First, we define our labels. Then we take a look at our tool pipeline, before looking at the different components of the tool. Additionally, we use a concrete example to show how the pipeline works in practice.

3.1 Classification Labels

We define our labels as not mutually exclusive. A software system can have multiple labels assigned to it.
The labels are part of the requirements for this classifier [1].

- **Applications** are executable software systems that provide end-user oriented functionality through some type of user interface, such as a GUI, web interface, or command line interface. They do not require the user to have programming skills or to interact with the source code.

  Inclusion Criteria:

  - T-A1: An application is executable and stand alone. It can operate independent of other software systems.
  - T-A2: An application is end-user oriented. We do not limit the target group from which a user originates. Who the user is depends on the context of the application.
  - T-A3: An application always has a form of user interface. This can be a graphical user interface or a basic command line tool.
  - T-A4: Using an application does not necessarily require programming skills.

[1] The labels are from another study of this chair, which has not yet been published, and thus cannot be referenced here.

  - T-A5: An application that is finished and being distributed comes with a stable release.

- **Libraries** provide functionality that is intended to be used in other programs via code-level APIs. Except for the functionality they provide, they contain no application logic and cannot run as standalone software.

  Inclusion Criteria:

  - T-L1: Libraries provide one or many functionalities to be used by other programs. A library is usually not suited to only extend one program, but rather are a way of encapsulating solutions for problems that can then be used by a variety of problems.
  - T-L2: Libraries interact with other programs on a code level. More advanced libraries define a dedicated API, but this is not always necessary. It is easy to reuse a Python project by importing functions and classes, therefore, we also consider systems without an API as libraries.
  - T-L3: A library cannot run as a stand alone system.

- **Frameworks** provide general application logic, but are designed to be extended by concrete functionality orchestrated by the framework. To this end, frameworks take control of the code that uses the framework, according to the principle of dependency inversion. Similar to libraries, frameworks usually provide generic helper functions that are frequently used in the context of the framework.

  Inclusion Criteria:

  - T-F1: Frameworks provide general application logic.
  - T-F2: Frameworks orchestrate how they can be extended with functionality. They take control of the code that uses the framework.
  - T-F3: Frameworks follow the principle of dependency inversion. They provide classes that can be extended by the user.

- **Plugins** are an extension of applications by providing additional functionality. The plugin interacts with an application through code-level APIs and with the user through a UI defined by the application it extends. The plugin is not executable by itself and is only designed to function with one application.

  Inclusion Criteria:

  - T-P1: Plugins are not executable and cannot run as a stand alone system. They can only be executed together with the application they extend.

- T-P2: Plugins interact with the application on a code-level API. The interface is defined by the application.
- T-P3: Plugins extend one application or a family of applications that have the same interface.

3.2 Example Software System

Real-world software systems from our dataset are very large, with thousands of program entities, so we use this small dummy example containing the different elements found in real-world systems for illustration.

The structure of the example software system is shown in Figure 3.1. The root folder of the directory tree of the software system contains the software system's name. Our example software system contains one package and four modules.

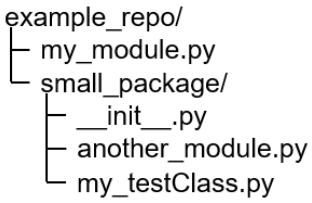


Figure 3.1: directory tree of the example software system

The full code of the example software system, along with the information which file the code snippets are from, is shown in Listing 1.

The third module '___init___.py' in the 'small_package' folder is empty.

3.3 Pipeline

The tool pipeline is shown in Figure 3.2. The input for the pipeline is either a single software system or a list of software systems that we want to classify. In case the systems are not already stored locally, they are automatically downloaded.

The files contained in the software systems are processed depending on the programming language used in them. Currently, the tool only processes Python files. These files are the input for the first component of the pipeline, the Ast-To-Ecore-Converter (ATE-Converter). This first converter parses the source code from each software system into an abstract syntax tree and converts this tree into an Ecore metamodel.

```python
#file my_module.py
import numpy as np

def convert_to_numpy_array(simple_array):
    numpy_array = np.array(simple_array)
    return numpy_array

#file my_testClass.py
class TestClass:
    def add_integers(num_1, num_2):
        sum = num_1 + num_2
        return sum

#file another_module.py
from my_testClass import TestClass

sum = TestClass.add_integers(3, 5)

print(sum)
```

Listing 1: full code of the example software system

The Ecore metamodels are piped into the second converter, the Ecore-To-Matrix-Converter (ETM-Converter). It converts each Ecore metamodel into a matrix structure, which is then used as input for the graph convolutional network.

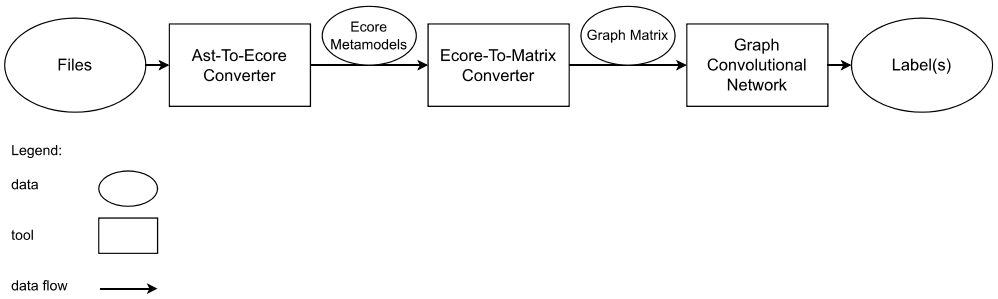


Figure 3.2: tool pipeline

If the tool is used to classify one software system, the type graph and the matrices are forwarded directly to the next component of the pipeline without writing them into files. If multiple software systems are converted, the graphs are written into files and then loaded again before being piped into the next pipeline component. This process is sped up with multi-processing. The tool parallelizes the conversion of the software systems and their graphs for optimization.

The output of the graph convolutional network, which is also the output of the entire tool, are the classification labels assigned to the software systems.

3.4 Ast-To-Ecore-Converter

The Ast-To-Ecore-Converter takes a software system and the information, if the output of the converter is supposed to be written in a file or returned to the calling function, as input.
The ATE-Converter walks the directory tree of the given repository path and creates a list of the Python files contained in the directory.
It also checks if an empty directory, or multiple, are contained in the software system.

The directories are represented as TPackage objects in the type graph. The name of the type graph matches the name of the software system, whose structure it contains. Every Python file is processed by the ATE-Converter. Files that cannot be parsed due to syntax errors are skipped. We log how often, and for which files, this occurs in our dataset.
Each Python file is represented by a TModule object. For each file, we recursively determine whether it is contained in a package, and if it is, we determine the package hierarchy by using the file's path. Afterwards the file itself is processed.

3.4.1 AST-Visitor

The Ast-To-Ecore-Converter uses the Python 'ast' module to parse a file's source code into an abstract syntax tree. The AST-Visitor defines how to visit the AST's nodes.
The ATE-Converter visits every node of the AST, however, we only process the nodes that can be represented in our type graph. These node types are defined in Chapter 2.
The converter checks before converting each node, if that node is already created in the type graph. Due to the different ways to access nodes in the AST, they can be visited multiple times.
For every node, the Ast-To-Ecore-Converter stores the node name, node type, and the information, whether it belongs to an external library or not, in the type graph.

One node type visited by the AST-Visitor are class definitions. When a ClassDef node in the AST is visited, the converter creates a TClass object in the type graph.
The TClass node is appended to both the type graph root and the TModule node representing the file the class is defined in.

The second step, when processing ClassDef nodes, is creating an inheritance structure for existing child classes. For every child class, a TClass object is created and, along with its name, appended to the type graph. The child TClass node is connected to its parent TClass node.

The third, and last, step, when processing ClassDef nodes, is visiting the AST nodes representing the program entities defined in the class.
The program entities we want to convert are function definitions. If a FunctionDef node is contained in the ClassDef node's body, the ATE-Converter creates a TMethodDefinition node to represent the FunctionDef node in the type graph.
The ATE-Converter creates TMethod and TMethodSignature nodes for the defined function. While the TMethod is appended to the type graph root and contains the name of the defined function, the TMethodDefinition is appended to the TClass node, specifying that the function is defined inside this class.

Another node type we want to convert are the aforementioned FunctionDef nodes. These can also be defined in a module outside of a class. The Ast-To-Ecore-Converter, after checking that the function does not already exist in the type graph, creates the nodes representing the function as described before.
The TMethodDefinition node is then appended to the current file's TModule node.

If a function has a parameter, or multiple, they are retrieved from where they are stored in the FunctionDef node. These parameters are represented by the TParameter nodes the ATE-Converter creates and appends to the TMethodSignature node.

The next node types in the abstract syntax tree, that the Ast-To-Ecore-Converter visits, are Import and ImportFrom nodes.
These nodes are important for representing function calls later. The ATE-Converter stores their information internally.

The Ast-To-Ecore-Converter also visits Assign nodes and checks if the node stored inside the Assign node is a Call node. If it is, the converter stores the name of the called function to convert the call later.

When visiting a Call node, we need to know what function the source of the call is. This information is important to represent the call in the type graph.
To get the source of the call, we retrieve the TMethodDefinition node that was visited last.
If there is no TMethodDefinition node stored, the ATE-Converter retrieves the location of the current module, which is its file path as a string. The last element of the file path, which is also the module name, is used as method name for the calling function. The ATE-Converter then creates a new TMethodDefinition, TMethodSignature, and TMethod node. The nodes are appended to the type graph.

An example of this is shown in the file 'another_module.py' of our example software system. The code of the file is shown in Listing 1. Since the, in the example imported, method 'TestClass.add_integers' is not called inside of a function defined in the module, a new method sharing the same name as the module is created in the type graph. A TModule node cannot be set as the source of a call in the type graph.

The target, which is the method that is being called, is retrieved from the Call node. The Ast-To-Ecore-Converter checks if the target matches an import that was stored in the converter. This information, along with the target, the source, and the current module are stored in the ATE-Converter.

3.4.2 Missing Classes

Some classes are missing in the type graph after every Python file in the software system has been processed.
The Ast-To-Ecore-Converter stores every class, along with its defined methods, to check if they are appended to the type graph after the conversion is done.
If they are missing, it is because the modules they belong to are missing. This is probably because the converter currently only processes Python files and skips Python files with syntax errors. However, the classes and methods from the files that cannot be processed can be imported by other modules, that are processed.
The ATE-Converter retrieves the information, which module they are imported from, and uses that information to create the appropriate TModule node, and, if found in the import information, the TPackage node.
These newly created nodes are then appended to the type graph, along with the TClass node that was missing previously.

3.4.3 Call Conversion

We convert three different types of function calls.

The first type are calls made from one method to another method within the same module.
The information needed to represent these calls in the type graph is stored in the Ast-To-Ecore-Converter. From every stored call information, ATE-Converter retrieves the target, the source, and the module nodes that represent the program entities in the call.
The target can either be a method within the module or a method within a class that is defined in the module. The target contains the name of the method, or the names of the class and the method concatenated with a dot. The TMethodDefinition node representing the target method is contained either in the TModule node,

or, if there is a class name stored in the target, in the TClass node.

The ATE-Converter checks if a TCall node with the same source and target nodes already exists. If it does not exist, a TCall node is created and the source and target are set accordingly.

The second type of function calls are calls made to methods imported from another module in the software system. The ATE-Converter retrieves the imported instance and the caller TMethodDefinition node.
The imported instance, which is the target of the call, has the structure of an arbitrary number of packages and subpackages, followed by a module, and either a class name and a method name, or a method name. These names are all concatenated by dots. The converter splits the imported instance into an array containing these names. The node type each name belongs to is determined by relying on naming conventions and the position in the array.

The last name in the array always belongs to the method that is being called. The second to last name is either a class, or a module. If the name starts with a capital letter, it is a class name. If it does not start with a capital letter, it is a module name. The converter relies on this naming convention to determine the node type. If the second to last name in the array is a class name, the name at the position before the class has to be a module.

In our example software system the call to 'add_integers' is this type of call. The code containing the call is shown in Listing 1. The called method is defined in class 'TestClass', which is defined in 'my_testClass.py'. The imported instance for this call is 'my_testClass.TestClass.add_integers'. The node types these names belong to after splitting is determined as described above, starting with the name at the last position.

In the case of our example, both the module with the method making the call and the module containing the called method are in the same package. Thus, the first name in our imported instance is a module, and not a package.
However, the Ast-To-Ecore-Converter checks the length of the array with the names of the imported instance, and if it has a length greater than three, the names at the positions before the module are determined to be packages and subpackages.

The converter searches for the TMethodDefinition node in the given class or module, and sets it as the target of the newly created TCall node. The caller TMethodDefinition node is set as source.
If the TModule node with the given name cannot be found in the type graph, it is a call to an external library.

The ATE-Converter excludes calls to program entities from other modules, or ex-

ternal libraries, with length one. These can be calls to single modules, classes, or methods. They could be the result of Import nodes, such as 'import torch' for example, or other import statements to modules in the software system, which are not Python files, and are therefore not converted currently.

The calls to external libraries, such as 'torch' or 'numpy', are the third and last type of calls. The imported instance storing the information for the target is split into its constituent parts and the node types of the names are determined as described before. In our example software system, a call to an external library is made, as shown in Listing 1. The imported instance for this call is 'numpy.array'. The method name is 'array', while 'numpy' is the module name.

For every external library, a package hierarchy is created. The Ast-To-Ecore-Converter creates this hierarchy based on the length of the name array. In our example software system, the first name is determined to be the module name. Since there are no other names that could be packages, the module name is also used as package name. The converter creates a TPackage node 'numpy' and appends a TModule node 'numpy' to its namespace.

If the array with the names of the imported instance has names at positions before the determined module name, these names are used to create the package hierarchy. The name at the first position in the array is the parent package, while the remaining names are subpackages of the parent package.

Like with every other node in the type graph, before creating new nodes for the external libraries, the ATE-Converter checks if these nodes already exist. This way, there is only one TPackage node created for every imported external library. If other imported instances share a parent package, which already exists, they are appended as subpackages and modules.
The TClass and TMethodDefinition nodes within the TModule nodes are created and set as call targets as described before.

3.4.4 Library Flag

The information, whether a node is from an external library or not, is stored in the type graph. For TClass nodes, the Ast-To-Ecore-Converter sets the node's 'tLib' attribute to True. For the other node types, this attribute does not show in the XMI file after writing the type graph in the file for storage. For these nodes, the ATE-Converter appends the string '_ExternalLibrary' to the node's name, or in case of a TModule node, the node's location. This is shown in the XMI file of our example in Listings 3 (line 17) and 4 (line 1), or in the visualized type graph in Figure 3.3.

3.4.5 Output

The output of the Ast-To-Ecore-Converter is the type graph. It is stored in an XMI file.
A part of the visualized type graph of the example software system is shown in Figure 3.3. The full XMI file is in Listings 3 and 4.

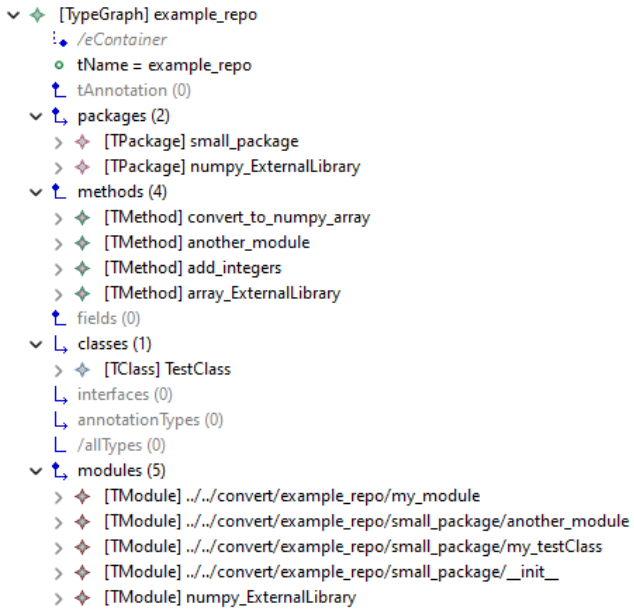


Figure 3.3: excerpt from the type graph of the example software system

3.5 Ecore-To-Matrix-Converter

The Ecore-To-Matrix-Converter takes a type graph as input, converts it into a matrix structure consisting of three matrices, and either returns them to the calling function, or writes them into three separate CSV files.

3.5.1 Node Feature Matrix

The Ecore-To-Matrix-Converter converts the nodes and their features first. It stores them in a node feature matrix. Every node in the matrix has a distinct key. This key is the index at which the node is stored in the node feature matrix. The node feature matrix has the shape $[num\_nodes, num\_node\_features]$. Each node in the graph has its node type, its name, and the library flag as features. The node types are the aforementioned types in the type graph.

The node name is hashed with message-digest algorithm 5 to convert it into a numerical value. We chose MD5 because out of the available hash functions in the Python module 'hashlib', MD5 has the shortest output length.
The library flag is stored in the node. Both the type and the library flag are one-hot encoded. Due to one-hot encoding the features, the total number of node features is 11. There are eight types, the name, and two values, True and False, for the library flag. The ETM-Converter checks before converting each node, if that node is already created in the matrix structure.
Due to the different ways to access nodes in the type graph, they can be visited multiple times.
For every converted node, the ETM-Converter appends the type, name, and library flag values to the node feature matrix.
The nodes are also stored in the Ecore-To-Matrix-Converter, along with the relations they have to their neighbouring nodes. This information is used to create the sparse edge matrix later.

The ETM-Converter converts nodes by type. The first type it converts are the TPackage nodes in the type graph.
The converter checks for every TPackage node if it has subpackages. They are converted recursively, since a subpackage can have a subpackage, or multiple, themself.

The next node type converted by the Ecore-To-Matrix-Converter are modules. After the TModule node itself is converted as described before, the nodes contained in the module are converted.
If the contained node is a TClass node, the ETM-Converter checks, after converting the class node itself, if the class has defined child classes and methods defined in the class. These nodes are converted next.
If the contained node is a method definition, it is converted as described before.

The next node type are TMethod nodes. The Ecore-To-Matrix-Converter accesses them via the type graph root. Together with the TMethod nodes, the corresponding TMethodSignature nodes, and the related TParameter nodes, are converted.

Since the TMethodDefinition, TMethodSignature, and TParameter nodes do not have names in the type graph, we create them in the node feature matrix so that all nodes have this feature. They share the name with their related TMethod node. To differentiate between them, their respective types are appended to their names.

In our example software system, the method 'add_integers' has a TMethod, TMethodDefinition, and TMethodSignature node representing it both in the type graph and in the node feature matrix. The method also has two parameters. In the node feature matrix, the three nodes representing the method have the names 'add_integers', 'add_integers_definition', and 'add_integers_signature' respectively. The parameters belonging to this method are assigned the names

'add_integers_signature_param_1' and 'add_integers_signature_param_2'.

For every converted TMethodDefinition node, the Ecore-To-Matrix-Converter checks if it has TCall nodes appended to it. If there are TCall nodes, these are converted next. The call nodes are also assigned names. They are constructed similarly to before, by using the name of the TMethodDefinition node they are appended to. In our example software system, the node representing the call to the method 'add_integers' is assigned the name 'add_integers_definition_call1'. Like the parameter nodes, the call nodes are consecutively numbered to differentiate between several of these nodes belonging to the same method.

The Ecore-To-Matrix-Converter checks if every class in the type graph is converted by additionally accessing each class via the type graph root.

3.5.2 Sparse Edge Matrix

The information on node neighbours stored in the Ecore-To-Matrix-Converter during the construction of the node feature matrix is used to construct the sparse edge matrix, which contains the edge information of our graph. The edge matrix has the shape $[num\_edges, 2]$ with the entry structure $[node\_id_1, node\_id_2]$.

If we take a look at the module 'my_testClass' in our example software system, it has edges to the package 'small_package' and to the contained class definition 'TestClass'. Module 'my_testClass' has ID eight in the node feature matrix, 'small_package' has ID zero, and 'TestClass' has ID nine.
The edge information pertaining these nodes is stored in the sparse edge matrix, as shown in Listing 2.

```
1 0, 8
2 8, 0
3 8, 9
```

Listing 2: excerpt from the sparse edge matrix of our example software system

3.5.3 Edge Attribute Matrix

The type graph stores the type of relation between nodes for every edge it contains. These relations are meta information on the previously explained neighbouring nodes. This information is retained in the edge attribute matrix. For every edge converted by the Ecore-To-Matrix-Converter, the relation between the nodes is

stored at the same position in the edge attribute matrix as the position of the corresponding edge information in the sparse edge matrix. The edge attribute matrix has the shape $[num\_edges, num\_edge\_attributes]$. The edge attributes are one-hot encoded, which results in a total number of 17 attributes.

3.5.4 Output

The output of the Ecore-To-Matrix-Converter is three matrices containing our graph structure.
The full CSV files of the example software system are shown in Listings 5, 6, 7, and 8.

3.6 Repository Dataset

The graphs can be loaded from the CSV files as a custom dataset. They can be loaded both with labels, when used for training, and without labels, when used for classification.
The tool checks for every graph in the dataset if all three matrices can be loaded from the files. In case of an empty file, for example if a software system consists of only one empty package, and therefore has no edges and edge attributes, the tool removes this sample from the dataset because there is no information to classify the software system with. When a sample is retrieved from the dataset, this sample is loaded directly from the files.
The matrices, after loading them, are converted into torch tensors. The dimensions of the sparse edge matrix are permuted, resulting in the tensor having the shape $[2, num\_edges]$.

The hexadecimal encoded hashed names are also transformed into completely numerical values. The non-integer characters in the strings are stripped, and a modulo operation with value 16 is performed on the remaining integer characters. This results in the hashed names being represented by an integer value.
The hexadecimal hash in the originally loaded node features is replaced with this computed value. This enables us to convert the node feature matrix into a torch tensor, which has limitations in its content size.

If the dataset is used for training, the labels are loaded from where they are stored in an excel file. The tool expects the excel file to contain the name of the software system and the assigned labels. Since we define our labels as not mutually exclusive, the loaded labels are multi-hot encoded. The multi-hot encoded labels are stored in a torch tensor.

3.7 Graph Convolutional Network

The input of the graph convolutional network is a graph consisting of three torch tensors. The GCN expects its input to be numerical torch tensors and to have a certain shape.
This is the reason for choosing to convert the graph into the aforementioned three matrices with the specific dimensions, while encoding any non-numerical values into numerical values using one-hot encoding and multi-hot encoding.

The GCN consists of three convolutional layers, each followed by an activation layer. The convolutional layer we use is GATConv. This layer operates on graphs and supports edge attributes as input.
The activation layer we use is ReLU, which is explained in Chapter 2.

The series of convolution and activation layers is followed by a pooling layer. We use global mean pool. Afterwards the GCN applies a dropout layer for regularization. We set 0.5 as the probability p.

The last layer in the graph convolutional network is a non-linear activation function specifically for multi-label classification. We use the sigmoid layer, which applies the element-wise function $Sigmoid(x) = \sigma(x) = \frac{1}{1+e^{-x}}$.

3.8 Training the GCN

We use the DataLoader from the PyTorch Geometric library for batching. Both the DataLoader and the graph convolutional network expect their input to be torch tensors with a specific shape. The RepositoryDataset transforms the torch tensors containing the graphs from our dataset to have this shape after loading them from their respective CSV files. We set a batch size of 32 for training.

The dataset is split with a ratio of 0.9 to 0.1 into a train set and a test set. Before we can start training our tool, we have to define some hyperparameters, which are the number of epochs and the number of folds for the k-fold cross validation. Due to the small size of our current dataset, we use k-fold cross validation, as explained in Chapter 2, to increase the number of different samples in the train set. We use four folds, with 100 epochs each, for training.

Common values for the number of hidden channels in a convolutional neural network are 32 and 64. We initially trained our tool with 64 hidden channels, however, we did not achieve better results than we did with 32 hidden channels. The only difference was a longer computation time. Therefore, we set the number of hidden channels to 32.

The loss function we use for training has to be one that is suited for multi-label classification. That loss function is MultiLabelSoftMarginLoss. We use the optimizer Adam and a learning rate of 0.001 to reduce the loss during training.

Furthermore, the tool checks if a GPU is available to speed up computation. If one is available, the tool moves the torch tensors to the GPU for faster training.

The output of the model is an array with four values. One value for every class in our dataset. We have to define a threshold, a value above which a label is considered to be predicted by the model. This value is set to 0.5 in our training loop.

We save the model with the best weighted average f1-score across all folds.

4 Results

Our tool can process and classify arbitrary Python software systems. It can be found on GitHub [1]. The dataset we trained the tool with is contained in an excel file in the data folder of the repository [2].

4.1 Our Dataset

We created a dataset of 160 labeled Python software systems. The number of samples in each of our four classes is shown in Figure 4.1. The total number of samples displayed in the figure adds up to more than 160 because some software systems are assigned multiple labels, as defined with multi-label classification. We used multiple

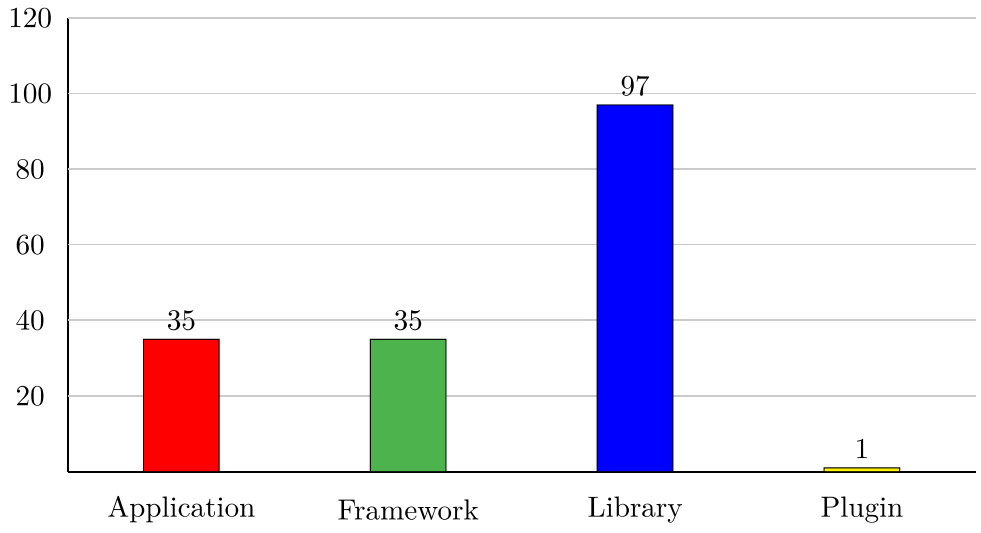


Figure 4.1: number of samples in each class of our dataset

filter criteria to select software systems from GitHub for our dataset. One criterion is the number of stars a repository has on GitHub. Another is the programming

[1] https://github.com/isselab/github-classifier
[2] https://github.com/isselab/github-classifier/blob/main/data/labeled_dataset_repos.xlsx

language Python. The software systems also have to have a dependency on at least one ML library.

4.2 Training with our Dataset

We trained our tool with our current dataset. Unfortunately, our dataset is too small to show good results and to prove that a classification only by analyzing the code structure is possible. The only class, that possibly contains enough samples for training, is the 'Library' class. As explained in Section 3.8, we defined a batch size of 32 and used a ratio of 0.9 to 0.1 to split our dataset into train and test sets. Therefore, our train set contains five batches with a total of 144 samples and our test set contains only 16 samples for evaluation. By looking at the f1-scores of the classes, we can assess the performance of our model. A visualization of the best f1-scores achieved during training are shown in Figure 4.2, along with the computed loss. The best scores are achieved in fold three of training. The computed loss during training

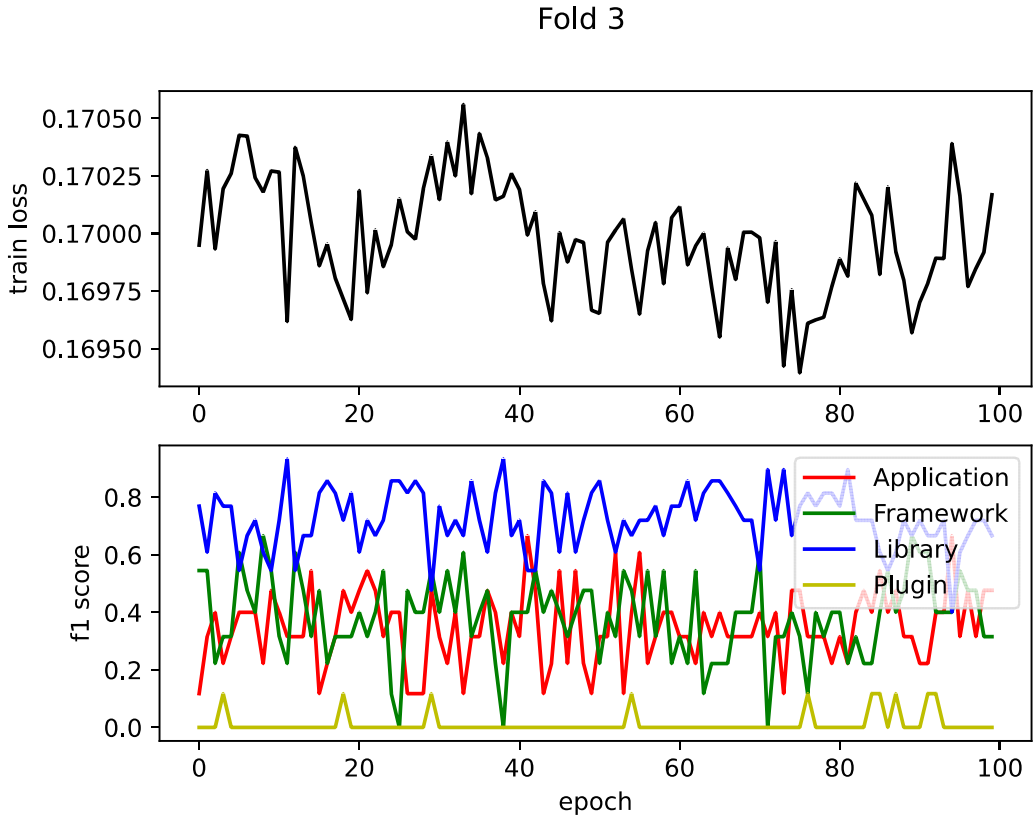


Figure 4.2: training results of fold 3 of our dataset

in fold three starts at approximately 0.1699 and ends with value 0.1701. The highest loss value is computed in epoch 33 with a loss of approximately 0.1705. The lowest loss value is approximately 0.1693 in epoch 75. The loss fluctuates between these values without permanently dropping.

The class 'Library' achieved the best f1-scores out of all classes. This class also has the most samples in our dataset. The f1-score starts at approximately 0.769. The best f1-score, with a value of 0.933, is achieved twice during training in epochs 11 and 38. The lowest f1-score for the 'Library' class is achieved in epoch 94 with a value of 0.400. The f1-score in the last epoch is 0.667.

For class 'Application', the f1-score starts with a value of 0.118 and ends with a value of 0.476. The highest f1-score is achieved in epochs 41 and 94 in training with a value of 0.667. The lowest score, with a value of 0.118, is achieved multiple times during training, for example in epochs 33 or 49.

The f1-score for class 'Framework' starts with value 0.545 and ends with value 0.316. The highest f1-score for this class is achieved twice, in epochs 8 and 89, with a value of 0.667. The lowest is 0.

Class 'Plugin' only has one sample in our dataset. The f1-score of this class is 0, except for a few epochs, where the highest value is achieved with 0.118.

The f1-scores fluctuate between these values for all classes during training. They do not converge, which means that the training process is not reaching a stable state. The model cannot make accurate predictions. The best f1-scores and loss during testing are visualized in Figure 4.3. The computed loss during testing is higher than during training. It starts with a value of 0.8381 and ends with a value of 0.8366. The highest loss is computed in epoch 34 with a value of 0.8437. The lowest is 0.8278 in epoch 74.

The f1-scores of all classes remain constant across all 100 epochs during testing. For class 'Application' the f1-score is 0.316. Class 'Framework' has an f1-score of 0.118, while 'Plugin' has an f1-score of 0. The class 'Library' has, like in training, the highest f1-score with a value of 0.897.

The precision, the recall, and the f1-scores during training and testing are in Tables A.1 and A.2 for more detailed results.

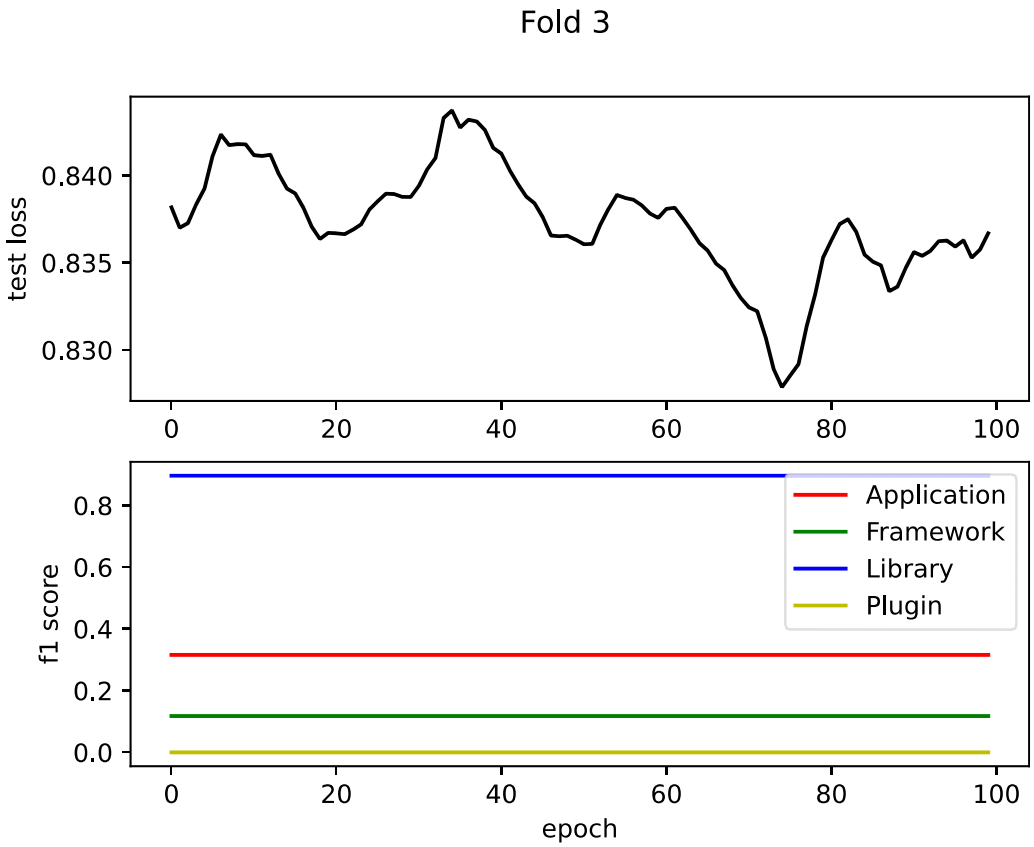


Figure 4.3: test results of fold 3 of our dataset

4.3 Applying our Tool to Other Problems

We tested training our tool with another dataset. The dataset has to consist of labeled software systems that are not preprocessed. The labels are arbitrary. Additionally, datasets that only contain software systems with a programming language other than Python cannot be used for training our current tool.
The only dataset we were able to find that fits these criteria is the ClassifyHub dataset [3]. The classes in this dataset are 'Development', 'Education', 'Homework', and 'Other'. Each of these classes has 70 samples in it.
We exclude the classes 'Data', 'Docs', and 'Web' because we do not expect the software systems belonging to these classes to contain any Python files.
The model is trained on three folds with 100 epochs each. Training our tool with the ClassifyHub dataset did not yield good results. The most likely reason for this is the small size of the dataset. Furthermore, most of the software systems did not contain

[3] https://github.com/Top-Ranger/ClassifyHub-data

any Python files. In these cases, their graphs only contain the package structure in the software systems. The best results are achieved in fold one. The loss and the f1-scores of the classes during training are shown in Figure 4.4. The computed loss

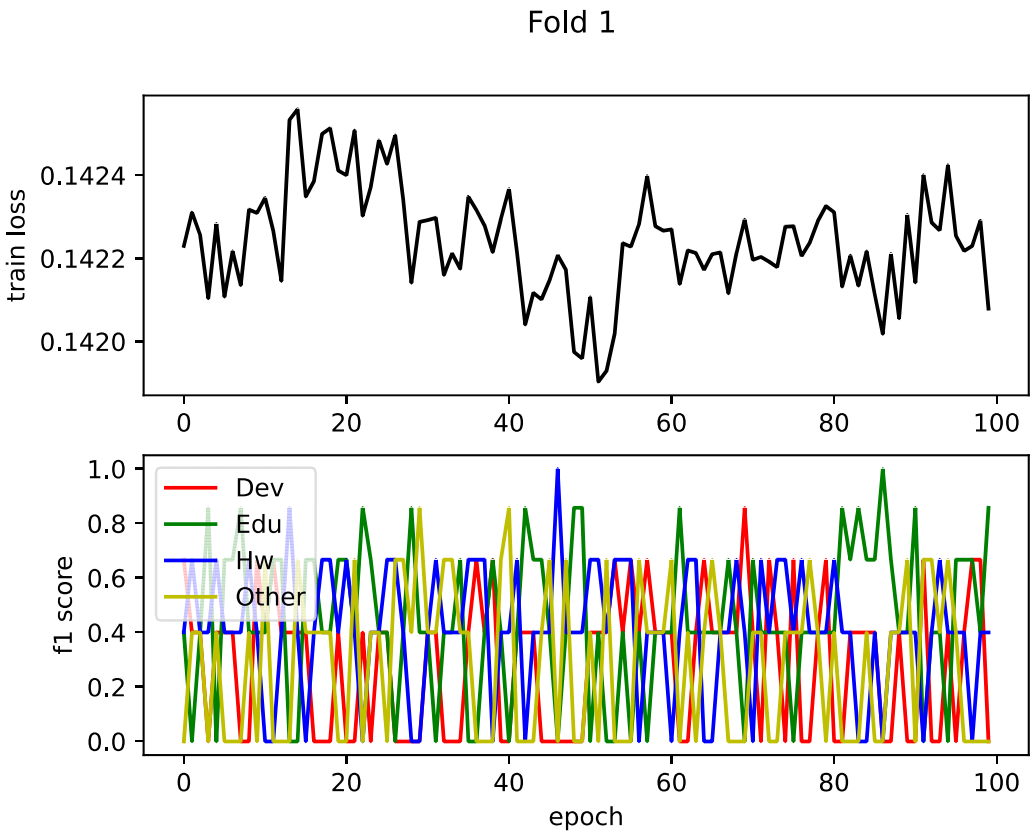


Figure 4.4: training results of fold 1 of ClassifyHub dataset

during training starts at 0.1422 in epoch 0 and ends at 0.1420 in epoch 99. The highest loss is computed in epoch 14 with a value of 0.1425 and the lowest is 0.1419 in epoch 51.

The best f1-scores across all four classes in training are achieved by class 'Education' in epoch 86 and by class 'Homework' in epoch 46, both with an f1-score of 1.00. Classes 'Development' and 'Other' achieve their highest f1-score with value 0.857 in epoch 69, for class 'Development', and epochs 29 and 40, for class 'Other'. The lowest f1-score of all classes is 0.

The values do not converge, which means that with the current data the graph convolutional network is not able to learn from the training data. The model most likely needs more training data. The values during testing are shown in Figure 4.5.

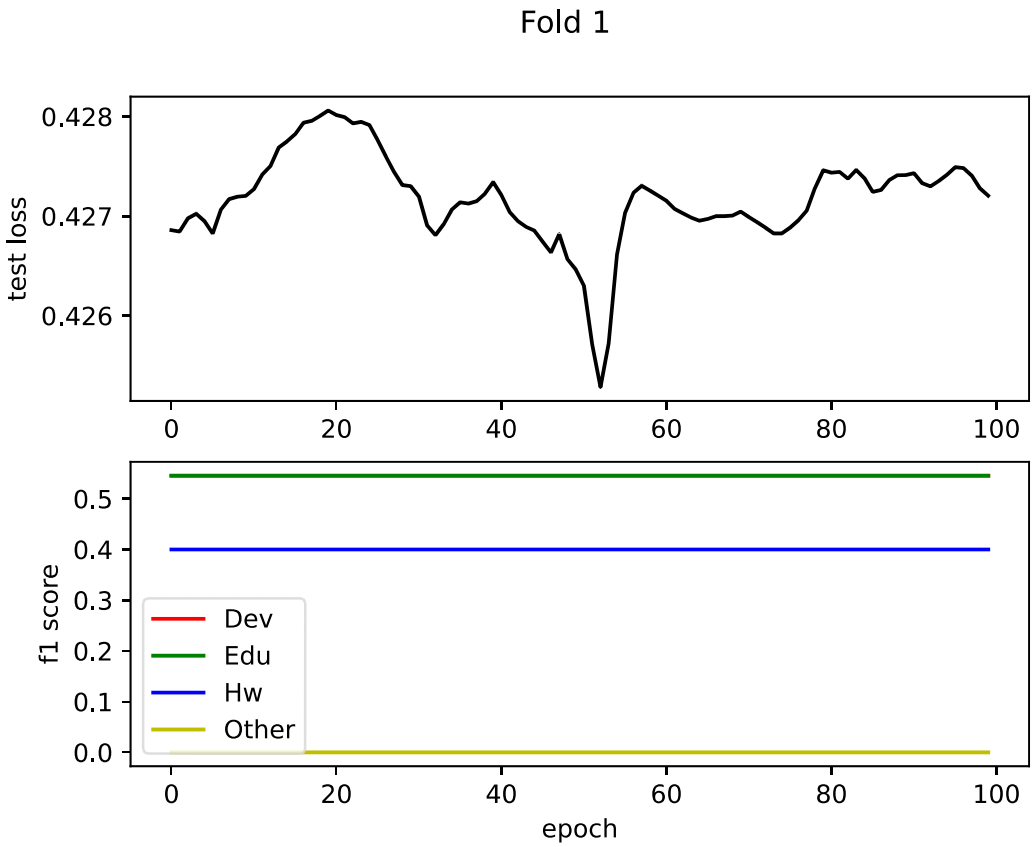


Figure 4.5: test results of fold 1 of ClassifyHub dataset

The loss during testing starts at 0.4268 and ends at 0.4272. It achieves the highest score in epoch 19 with a value of 0.4280. The lowest is computed with a value of 0.4252 in epoch 52. Like with our own dataset, the f1-scores of every class remain constant across all 100 epochs. Classes 'Education' and 'Development' achieve the highest f1-score with a value of 0.545 each. Two lines plotted with the same values do not show in the plot. One line overlays the other. Class 'Homework' achieves an f1-score of 0.400, while the f1-score of class 'Other' remains 0.

The precision, the recall, and the f1-scores of fold one are also shown in Tables A.3 and A.4.

5 Conclusion

The task of manually classifying software systems is challenging, due to the time and effort the task requires. Our tool can classify arbitrary software systems by converting them into graphs and piping them into a graph convolutional network for classification.

We created a dataset for training and evaluating our tool. The results from training indicate that a classification of software systems only by analyzing the structure of their source code could be possible. However, in order to determine that we need a bigger dataset, probably due to the complexity of the problem. The class 'Library' has the best results and also the most samples. This is supported by comparing our dataset size to Dang et al.'s [15], who used a dataset of 15,262 software systems to train their tool. Unfortunately, their dataset contained the already processed software systems, which is why we were not able to test our own tool with it.

If trained properly, our tool serves as a solution to the problem of classifying arbitrary software systems, when conducting mining studies.

5.1 Threats to Validity

There are multiple threats to validity for our current tool. The current dataset is too small to train the tool properly on it. Furthermore, the dataset has a huge bias towards libraries, as the majority of the software systems in our dataset belong to this class. Additionally, the dataset was originally created for a different study, which influenced the filter criteria for software systems in the dataset. Every software system is required to have at least one dependency on a ML library. This is an additional bias in the dataset.

Another threat to validity is that not the complete software systems are converted. Not only does the tool only process Python files, it also skips some of these Python files if there are syntax errors in them. This, however, is negligible because out of the several ten thousand Python files in our current dataset, only 51 of them were skipped.

5.2 Future Work

Future work for this tool should address and hopefully resolve the threats to validity. The tool can be extended to process files written in other programming languages, such as Java or C++. This way, the graph convolutional network would have more data which it could be trained on.

Another task could also be to extend our dataset. With a larger dataset, we could possibly train our tool so that we can use it to make accurate predictions when classifying arbitrary software systems for mining studies.

Due to the modular design of our tool, the components of the tool's pipeline could be reused for other software projects that rely on analyzing the structure of source code.

Additionally, further looking into the possibility of training the tool on other datasets and tasks could be useful. Our tool could be, for example, used to solve classification tasks with different labels, like we did with the ClassifyHub dataset.

List of Figures

List of Tables

List of Source Codes

Bibliography

[1] Copyright © Eclipse Foundation AISBL. *Eclipse Modeling Framework (EMF)*. Accessed 2024, June 11. URL: https://eclipse.dev/modeling/emf/.

[2] Vincent Aranega. *PyEcore Documentation*. Accessed 2024, April 09. 2017. URL: https://pyecore.readthedocs.io/en/latest/.

[3] Abid Ali Awan. *A Comprehensive Introduction to Graph Neural Networks (GNNs)*. Accessed 2024, July 09. 2022. URL: https://www.datacamp.com/tutorial/comprehensive-introduction-graph-neural-networks-gnns-tutorial.

[4] Avijit Bhattacharjee. *Implementing K-Fold Cross-Validation from Scratch in Python*. Accessed 2024, June 27. 2023. URL: https://medium.com/@avijit.bhattacharjee1996/implementing-k-fold-cross-validation-from-scratch-in-python-ae413b41c80d.

[5] Daniel Boadzie. *Introduction to Abstract Syntax Trees in Python*. Accessed 2024, June 10. 2023. URL: https://earthly.dev/blog/python-ast/.

[6] Jason Brownlee. *3 Ways to Encode Categorical Variables for Deep Learning*. Accessed 2024, June 17. 2020. URL: https://machinelearningmastery.com/how-to-prepare-categorical-data-for-deep-learning-in-python/.

[7] Jason Brownlee. *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. Accessed 2024, July 09. 2020. URL: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

[8] Jason Brownlee. *Overfitting and Underfitting With Machine Learning Algorithms*. Accessed 2024, August 05. 2019. URL: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/.

[9] Bala Priya C. *Abstract Syntax Tree (AST) - Explained in Plain English*. Accessed 2024, March 15. 2022. URL: https://dev.to/balapriya/abstract-syntax-tree-ast-explained-in-plain-english-1h38.

[10] Francesco Casalegno. *Graph Convolutional Networks — Deep Learning on Graphs*. Accessed 2024, July 09. 2021. URL: https://towardsdatascience.com/graph-convolutional-networks-deep-99d7fee5706f.

[11] Christopher Clifton. *data mining*. Accessed 2024, August 06. 2024. URL: https://www.britannica.com/technology/data-mining.

[12] Python Software Foundation Copyright 2001-2024. *hashlib — Secure hashes and message digests*. Accessed 2024, July 09. 2024. URL: https://docs.python.org/3/library/hashlib.html#hashlib.md5.

[13] PyG Team. Revision fbafbc4f Copyright 2024. *Advanced Mini-Batching*. Accessed 2024, July 09. URL: https://pytorch-geometric.readthedocs.io/en/latest/advanced/batching.html.

[14] 2023 Copyright IBM Corporation 1997. *Exchanging model data by using XMI*. Accessed 2024, April 11. 2023. URL: https://www.ibm.com/docs/en/engineering-lifecycle-management-suite/design-rhapsody/9.0.2?topic=tools-exchanging-model-data-by-using-xmi.

[15] Yen-Trang Dang, Thanh-Le Cong, Phuc-Thanh Nguyen, Anh M. T. Bui, Phuong T. Nguyen, Bach Le, and Quyet-Thang Huynh. "LEGION: Harnessing Pre-trained Language Models for GitHub Topic Recommendations with Distribution-Balance Loss". In: *Software Engineering (cs.SE)* (2024). URL: https://arxiv.org/abs/2403.05873.

[16] scikit-learn developers. *MultiLabelBinarizer*. Accessed 2024, June 17. 2024. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MultiLabelBinarizer.html.

[17] TensorFlow developers. *tf.keras.layers.CategoryEncoding*. Accessed 2024, June 17. 2024. URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers/CategoryEncoding#args.

[18] Sanket Doshi. *Various Optimization Algorithms For Training Neural Network*. Accessed 2024, August 04. 2019. URL: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6.

[19] PyG Team. Revision e213c297. *PyG Documentation*. Accessed 2024, April 09. 2024. URL: https://pytorch-geometric.readthedocs.io/en/latest/.

[20] Python Software Foundation. *ast — Abstract Syntax Trees*. Accessed 2024, June 27. URL: https://docs.python.org/3/library/ast.html#node-classes.

[21] Chris Hoffman. *What Is a CSV File, and How Do I Open It?* Accessed 2024, April 22. 2018. URL: https://www.howtogeek.com/348960/what-is-a-csv-file-and-how-do-i-open-it/.

[22] Samuel Idowu, Yorick Sens, Thorsten Berger, Jacob Krueger, and Michael Vierhauser. "A Large-Scale Study of ML-Related Python Projects". In: *Symposium on Applied Computing (SAC)* (2024). URL: https://research.tue.nl/en/publications/a-large-scale-study-of-ml-related-python-projects.

[23] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. "The Promises and Perils of Mining GitHub". In: *MSR 2014: Proceedings of the 11th Working Conference on Mining Software Repositories* (2014), pp. 92–101. URL: https://doi.org/10.1145/2597073.2597074.

[24] Murat Karakaya. *Multi Label Model Evaluation*. Accessed 2024, July 13. 2020. URL: https://www.kaggle.com/code/kmkarakaya/multi-label-model-evaluation.

[25] Richard Kennard and John Leaney. *An Introduction to Software Mining*. Accessed 2024, July 20. 2012. URL: https://www.researchgate.net/publication/232415957_An_Introduction_to_Software_Mining.

[26] Thomas Kipf. *Graph Convolutional Networks*. Accessed 2024, July 02. 2016. URL: https://tkipf.github.io/graph-convolutional-networks/.

[27] Dominik Kundel. *ASTs - What are they and how to use...* Accessed 2024, February 29. 2020. URL: https://www.twilio.com/en-us/blog/abstract-syntax-trees.

[28] Nadia Nahar, Haoran Zhang, Grace Lewis, Shurui Zhou, and Christian Kästner. "A Dataset and Analysis of Open-Source Machine Learning Products". In: *Software Engineering (cs.SE)* (2023). URL: https://doi.org/10.48550/arXiv.2308.04328.

[29] Sven Matthias Peldszus. *Security Compliance in Model-driven Development of Software Systems in Presence of Long-Term Evolution and Variants*. Springer Vieweg, 2022, pp. 78–88. ISBN: 978-3-658-37664-2. DOI: https://doi.org/10.1007/978-3-658-37665-9.

[30] om pramod. *Convergence in deep learning*. Accessed 2024, August 05. 2023. URL: https://medium.com/@ompramod9921/convergence-in-deep-learning-f96568923d43.

[31] Adrian Rosebrock. *Convolutional Neural Networks (CNNs) and Layer Types*. Accessed 2024, July 09. 2021. URL: https://pyimagesearch.com/2021/05/14/convolutional-neural-networks-cnns-and-layer-types/.

[32] Shanshan. *Intro to Python ast Module*. Accessed 2024, June 10. 2022. URL: https://medium.com/@wshanshan/intro-to-python-ast-module-bbd22cd505f7.

[33] Marcus Soll and Malte Vosgerau. "ClassifyHub: An Algorithm to Classify GitHub Repositories". In: *Conference: Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)* (2017), pp. 374–379. URL: https://rdcu.be/dwN3u.

[34] Thomas Stahl, Markus Völter, Jorn Bettin, Arno Haase, Simon Helsen, Krzysztof Czarnecki, and Bettina von Stockfleth. *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, 2006, pp. 85–88. ISBN: 9780470025703.

[35] Bedir Tekinerdogan, Dominique Blouin, Hans Vangheluwe, Miguel Goulão, Paulo Carreira, and Vasco Amaral, eds. *Multi-Paradigm Modelling Approaches for Cyber-Physical Systems*. Copyright © 2021 Elsevier Inc, 2021. ISBN: 978-0-12-819105-7. DOI: https://doi.org/10.1016/C2018-0-04990-7. URL: https://www.sciencedirect.com/topics/computer-science/abstract-syntax.

[36] Turing. *How Machine Learning Can Be Used for Multiclass Classification in Python*. Accessed 2024, June 17. 2024. URL: https://www.turing.com/kb/how-machine-learning-can-be-used-for-multiclass-classification-in-python.

[37] Alexandra Twin. *What Is Data Mining? How It Works, Benefits, Techniques, and Examples*. Accessed 2024, August 06. 2024. URL: https://www.investopedia.com/terms/d/datamining.asp.

[38] Dinesh Yadav. *Categorical encoding using Label-Encoding and One-Hot-Encoder*. Accessed 2024, June 14. 2019. URL: https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd.

[39] Kinza Yasar and Andrew Zola. *Definition hashing*. Accessed 2024, July 09. 2024. URL: https://www.techtarget.com/searchdatamanagement/definition/hashing.

[40] Vishal Yathish. *Loss Functions and Their Use In Neural Networks*. Accessed 2024, August 04. 2022. URL: https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9.

A Appendix

A.1 XMI File

```
<?xml version='1.0' encoding='UTF-8'?>
<org.gravity.typegraph.basic:TypeGraph xmlns:xmi="http://www.omg.org/XMI"
↪ xmlns:org.gravity.typegraph.basic="http://www.gravity-tool
↪ .org/typegraph/basic" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
↪ xmi:id="42db76d1-259b-4b9f-ac84-b34ad4687152" tName="example_repo"
↪ classes="c13756b3-b218-47d6-9e23-d91bff4d9245" xmi:version="2.0">
  <methods xmi:id="f5b9adf6-ffc4-427b-8106-174e9f82c91b"
  ↪ tName="convert_to_numpy_array">
    <signatures xmi:id="03ede3ac-be6e-4b37-a267-505c74248200"
    ↪ definitions="6ec16727-7aea-46e4-9f68-9b3bacb8ed78">
      <parameters xmi:id="d8ca36db-1cfb-4748-9450-b1f683958700"/>
    </signatures>
  </methods>
  <methods xmi:id="d69a660c-8546-48d1-b57a-5c69ef56d646" tName="another_module">
    <signatures xmi:id="923a9e94-bbc1-48e1-810a-2bbc0cb97500"
    ↪ definitions="9f0fa3b9-e266-4d0c-83a3-0656f754bb1f"/>
  </methods>
  <methods xmi:id="def15cf1-8aee-457e-8e05-b230e1c382a3" tName="add_integers">
    <signatures xmi:id="2d6e6585-3dff-4226-ba66-8e64fdb8f2f5"
    ↪ definitions="4a845705-5cb6-40c4-b14a-89e6225bd085">
      <parameters xmi:id="2a41f2fd-e5cf-48ef-894d-81cc8e38425d"
      ↪ next="5c30066a-724b-4528-bad2-d7593030bc7c"/>
      <parameters xmi:id="5c30066a-724b-4528-bad2-d7593030bc7c"
      ↪ previous="2a41f2fd-e5cf-48ef-894d-81cc8e38425d"/>
    </signatures>
  </methods>
  <methods xmi:id="d92be422-66f8-46f2-8577-15e9fcdcdfa7"
  ↪ tName="array_ExternalLibrary">
    <signatures xmi:id="4504671e-1c4f-4b32-9fbe-deca18ff2553"
    ↪ definitions="50a7e865-6b8a-4120-8a4e-f2c3cc7a1559"/>
  </methods>
  <packages xmi:id="a9a83c92-0412-4f16-847c-5c07fef9897b" tName="small_package"
  ↪ modules="40300fee-f614-4419-b36c-dbb65f75f07b
  ↪ 733a1efa-3ef5-4bff-9d35-eb9c775faa53 dd791c86-99dd-4eaf-9f64-9c776080554c"/>
```

Listing 3: first part of the XMI file of the example software system

```
<packages xmi:id="fa64b54d-22cf-43ad-a01a-32c54fd85bce"
↪  tName="numpy_ExternalLibrary" modules="331559c1-7bcf-4beb-82bc-793e7675b96c"/>
  <modules xmi:id="4b1c01b7-31e4-4727-ac20-6326a93da86f"
  ↪  location="../../convert/example_repo/my_module">
    <contains xsi:type="org.gravity.typegraph.basic:TMethodDefinition"
    ↪  xmi:id="6ec16727-7aea-46e4-9f68-9b3bacb8ed78"
    ↪  signature="03ede3ac-be6e-4b37-a267-505c74248200">
      <accessing xsi:type="org.gravity.typegraph.basic:TCall"
      ↪  xmi:id="f0d30b79-1a86-4a59-bac4-6b752db927b9"
      ↪  target="50a7e865-6b8a-4120-8a4e-f2c3cc7a1559"/>
    </contains>
  </modules>
  <modules xmi:id="40300fee-f614-4419-b36c-dbb65f75f07b"
  ↪  location="../../convert/example_repo/small_package/another_module"
  ↪  namespace="a9a83c92-0412-4f16-847c-5c07fef9897b">
    <contains xsi:type="org.gravity.typegraph.basic:TMethodDefinition"
    ↪  xmi:id="9f0fa3b9-e266-4d0c-83a3-0656f754bb1f"
    ↪  signature="923a9e94-bbc1-48e1-810a-2bbc0cb97500">
      <accessing xsi:type="org.gravity.typegraph.basic:TCall"
      ↪  xmi:id="1d652c58-9db2-4e4a-a459-010b21fff09d"
      ↪  target="4a845705-5cb6-40c4-b14a-89e6225bd085"/>
    </contains>
  </modules>
  <modules xmi:id="733a1efa-3ef5-4bff-9d35-eb9c775faa53"
  ↪  location="../../convert/example_repo/small_package/my_testClass"
  ↪  namespace="a9a83c92-0412-4f16-847c-5c07fef9897b">
    <contains xsi:type="org.gravity.typegraph.basic:TClass"
    ↪  xmi:id="c13756b3-b218-47d6-9e23-d91bff4d9245" tName="TestClass">
      <defines xsi:type="org.gravity.typegraph.basic:TMethodDefinition"
      ↪  xmi:id="4a845705-5cb6-40c4-b14a-89e6225bd085"
      ↪  signature="2d6e6585-3dff-4226-ba66-8e64fdb8f2f5"
      ↪  accessedBy="1d652c58-9db2-4e4a-a459-010b21fff09d"/>
    </contains>
  </modules>
  <modules xmi:id="dd791c86-99dd-4eaf-9f64-9c776080554c"
  ↪  location="../../convert/example_repo/small_package/__init__"
  ↪  namespace="a9a83c92-0412-4f16-847c-5c07fef9897b"/>
  <modules xmi:id="331559c1-7bcf-4beb-82bc-793e7675b96c"
  ↪  location="numpy_ExternalLibrary"
  ↪  namespace="fa64b54d-22cf-43ad-a01a-32c54fd85bce">
    <contains xsi:type="org.gravity.typegraph.basic:TMethodDefinition"
    ↪  xmi:id="50a7e865-6b8a-4120-8a4e-f2c3cc7a1559"
    ↪  signature="4504671e-1c4f-4b32-9fbe-deca18ff2553"
    ↪  accessedBy="f0d30b79-1a86-4a59-bac4-6b752db927b9"/>
  </modules>
</org.gravity.typegraph.basic:TypeGraph>
```

Listing 4: second part of the XMI file of the example software system

A.2 CSV Files

```
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 2f003fcb75409900d17fd6a2c797bfe2, 1.0, 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, ecc19e0f13b343ae6b7d99d4d69981f2, 0.0, 1.0
0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 3035a5538e5e13eb79d7456488341782, 1.0, 0.0
0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 7a03a063d68ce7cc726760e335ce3dfd, 1.0, 0.0
1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 5cddd9f543b8b297839b139a63652279, 0.0, 1.0
0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, ebddb1179efc1814ce2bc00614c2f944, 1.0, 0.0
0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 8d82b2e5c9724c90b4a1bf8492f478d0, 1.0, 0.0
1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, e19510d1dc287482984e40a9ba6e46c0, 1.0, 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 57b3dadc4e5cc2bbd8ea1718f84680ca, 1.0, 0.0
0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3b5abaddc0344111a05bc707222d4f00, 1.0, 0.0
0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, f15357fe9729605d01dbf19ddd8bda0b, 1.0, 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 13903e336ea74b2e4b4abc263d23e24d, 1.0, 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, ecc19e0f13b343ae6b7d99d4d69981f2, 0.0, 1.0
0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 7bb0084f6ef9977bdcf599a1cc02f704, 0.0, 1.0
0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 86d0d80f77ea0c0601c4f9332bc6713c, 1.0, 0.0
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 4f989a50838bfb7c4ec18b114075f8b9, 1.0, 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 4cf3f1f8df08766bde65f65a2de84d1f, 1.0, 0.0
0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 6a740f8686ff86953c548b39b159abe4, 1.0, 0.0
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 673cc5bf16d4b07c28154bf0376d74d7, 1.0, 0.0
0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, cb0836479f8369b23002e4709f79f393, 1.0, 0.0
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 8a991deb4d00f6945b4479bc27da889a, 1.0, 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 491c53f2aaa1de88c20727f3f4b272ae, 1.0, 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 125a0472bcc1c096e19ce30f6a63ae1a, 1.0, 0.0
0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, e829395653e9ccbc5b02fd0d2ccf859e, 0.0, 1.0
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1c6e574cde3d50b7fb43a586a0828fb8, 0.0, 1.0
```

Listing 5: CSV file of the example software system's node feature matrix

```
2, 3
3, 4
4, 3
13, 4
4, 13
0, 5
5, 0
5, 6
6, 7
7, 6
10, 7
7, 10
0, 8
8, 0
8, 9
9, 10
0, 11
11, 0
1, 12
12, 1
12, 13
14, 3
14, 15
15, 16
17, 6
17, 18
19, 10
19, 20
20, 21
21, 22
22, 21
20, 22
23, 13
23, 24
```

Listing 6: CSV file of the example software system's sparse edge matrix

```
0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0
0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
↪  0.0
1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  1.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0
0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0
0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
↪  0.0
1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  1.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0
0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0
0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪  0.0
```

Listing 7: first part of the CSV file of the example software system's edge attribute matrix

```
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0
0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,
↪ 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,
↪ 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,
↪ 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0,
↪ 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
↪ 0.0
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,
↪ 0.0
```

Listing 8: second part of the CSV file of the example software system's edge attribute matrix

A.3 Training Results of our Dataset

Table A.1: Training results of fold 3 of our dataset

Epoch	Application	Framework	Library	Plugin
0	0.062 1.000 0.118	0.375 1.000 0.545	0.625 1.000 0.769	0.000 0.000 0.000
1	0.188 1.000 0.316	0.375 1.000 0.545	0.438 1.000 0.609	0.000 0.000 0.000
2	0.250 1.000 0.400	0.125 1.000 0.222	0.688 1.000 0.815	0.000 0.000 0.000
3	0.125 1.000 0.222	0.188 1.000 0.316	0.625 1.000 0.769	0.062 1.000 0.118
4	0.188 1.000 0.316	0.188 1.000 0.316	0.625 1.000 0.769	0.000 0.000 0.000
5	0.250 1.000 0.400	0.438 1.000 0.609	0.375 1.000 0.545	0.000 0.000 0.000
6	0.250 1.000 0.400	0.312 1.000 0.476	0.500 1.000 0.667	0.000 0.000 0.000
7	0.250 1.000 0.400	0.250 1.000 0.400	0.562 1.000 0.720	0.000 0.000 0.000
8	0.125 1.000 0.222	0.500 1.000 0.667	0.438 1.000 0.609	0.000 0.000 0.000
9	0.312 1.000 0.476	0.375 1.000 0.545	0.375 1.000 0.545	0.000 0.000 0.000
10	0.250 1.000 0.400	0.188 1.000 0.316	0.562 1.000 0.720	0.000 0.000 0.000
11	0.188 1.000 0.316	0.125 1.000 0.222	0.875 1.000 0.933	0.000 0.000 0.000
12	0.188 1.000 0.316	0.438 1.000 0.609	0.375 1.000 0.545	0.000 0.000 0.000
13	0.188 1.000 0.316	0.312 1.000 0.476	0.500 1.000 0.667	0.000 0.000 0.000
14	0.375 1.000 0.545	0.188 1.000 0.316	0.500 1.000 0.667	0.000 0.000 0.000
15	0.062 1.000 0.118	0.312 1.000 0.476	0.688 1.000 0.815	0.000 0.000 0.000
16	0.125 1.000 0.222	0.125 1.000 0.222	0.750 1.000 0.857	0.000 0.000 0.000
17	0.188 1.000 0.316	0.188 1.000 0.316	0.688 1.000 0.815	0.000 0.000 0.000
18	0.312 1.000 0.476	0.188 1.000 0.316	0.562 1.000 0.720	0.062 1.000 0.118
19	0.250 1.000 0.400	0.188 1.000 0.316	0.688 1.000 0.815	0.000 0.000 0.000
20	0.312 1.000 0.476	0.250 1.000 0.400	0.438 1.000 0.609	0.000 0.000 0.000
21	0.375 1.000 0.545	0.188 1.000 0.316	0.562 1.000 0.720	0.000 0.000 0.000
22	0.312 1.000 0.476	0.250 1.000 0.400	0.500 1.000 0.667	0.000 0.000 0.000
23	0.188 1.000 0.316	0.375 1.000 0.545	0.562 1.000 0.720	0.000 0.000 0.000
24	0.250 1.000 0.400	0.062 1.000 0.118	0.750 1.000 0.857	0.000 0.000 0.000
25	0.250 1.000 0.400	0.000 0.000 0.000	0.750 1.000 0.857	0.000 0.000 0.000
26	0.062 1.000 0.118	0.312 1.000 0.476	0.688 1.000 0.815	0.000 0.000 0.000
27	0.062 1.000 0.118	0.250 1.000 0.400	0.750 1.000 0.857	0.000 0.000 0.000
28	0.062 1.000 0.118	0.250 1.000 0.400	0.688 1.000 0.815	0.000 0.000 0.000
29	0.312 1.000 0.476	0.375 1.000 0.545	0.312 1.000 0.476	0.062 1.000 0.118
30	0.188 1.000 0.316	0.250 1.000 0.400	0.625 1.000 0.769	0.000 0.000 0.000
31	0.125 1.000 0.222	0.375 1.000 0.545	0.500 1.000 0.667	0.000 0.000 0.000
32	0.250 1.000 0.400	0.250 1.000 0.400	0.562 1.000 0.720	0.000 0.000 0.000
33	0.062 1.000 0.118	0.438 1.000 0.609	0.500 1.000 0.667	0.000 0.000 0.000
34	0.188 1.000 0.316	0.188 1.000 0.316	0.750 1.000 0.857	0.000 0.000 0.000
35	0.188 1.000 0.316	0.250 1.000 0.400	0.562 1.000 0.720	0.000 0.000 0.000

Note: the values in each cell are, in this order, precision recall f1-score

Epoch	Application	Framework	Library	Plugin
36	0.312 1.000 0.476	0.312 1.000 0.476	0.438 1.000 0.609	0.000 0.000 0.000
37	0.250 1.000 0.400	0.125 1.000 0.222	0.688 1.000 0.815	0.000 0.000 0.000
38	0.125 1.000 0.222	0.000 0.000 0.000	0.875 1.000 0.933	0.000 0.000 0.000
39	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667	0.000 0.000 0.000
40	0.188 1.000 0.316	0.250 1.000 0.400	0.562 1.000 0.720	0.000 0.000 0.000
41	0.500 1.000 0.667	0.250 1.000 0.400	0.375 1.000 0.545	0.000 0.000 0.000
42	0.312 1.000 0.476	0.375 1.000 0.545	0.375 1.000 0.545	0.000 0.000 0.000
43	0.062 1.000 0.118	0.250 1.000 0.400	0.750 1.000 0.857	0.000 0.000 0.000
44	0.125 1.000 0.222	0.312 1.000 0.476	0.688 1.000 0.815	0.000 0.000 0.000
45	0.375 1.000 0.545	0.250 1.000 0.400	0.438 1.000 0.609	0.000 0.000 0.000
46	0.125 1.000 0.222	0.188 1.000 0.316	0.688 1.000 0.815	0.000 0.000 0.000
47	0.375 1.000 0.545	0.250 1.000 0.400	0.438 1.000 0.609	0.000 0.000 0.000
48	0.125 1.000 0.222	0.312 1.000 0.476	0.562 1.000 0.720	0.000 0.000 0.000
49	0.062 1.000 0.118	0.312 1.000 0.476	0.688 1.000 0.815	0.000 0.000 0.000
50	0.188 1.000 0.316	0.125 1.000 0.222	0.750 1.000 0.857	0.000 0.000 0.000
51	0.188 1.000 0.316	0.250 1.000 0.400	0.562 1.000 0.720	0.000 0.000 0.000
52	0.438 1.000 0.609	0.188 1.000 0.316	0.438 1.000 0.609	0.000 0.000 0.000
53	0.062 1.000 0.118	0.375 1.000 0.545	0.562 1.000 0.720	0.000 0.000 0.000
54	0.250 1.000 0.400	0.312 1.000 0.476	0.500 1.000 0.667	0.062 1.000 0.118
55	0.438 1.000 0.609	0.188 1.000 0.316	0.562 1.000 0.720	0.000 0.000 0.000
56	0.125 1.000 0.222	0.375 1.000 0.545	0.562 1.000 0.720	0.000 0.000 0.000
57	0.188 1.000 0.316	0.188 1.000 0.316	0.625 1.000 0.769	0.000 0.000 0.000
58	0.250 1.000 0.400	0.375 1.000 0.545	0.500 1.000 0.667	0.000 0.000 0.000
59	0.250 1.000 0.400	0.125 1.000 0.222	0.625 1.000 0.769	0.000 0.000 0.000
60	0.188 1.000 0.316	0.188 1.000 0.316	0.625 1.000 0.769	0.000 0.000 0.000
61	0.188 1.000 0.316	0.125 1.000 0.222	0.750 1.000 0.857	0.000 0.000 0.000
62	0.125 1.000 0.222	0.375 1.000 0.545	0.562 1.000 0.720	0.000 0.000 0.000
63	0.250 1.000 0.400	0.062 1.000 0.118	0.688 1.000 0.815	0.000 0.000 0.000
64	0.188 1.000 0.316	0.125 1.000 0.222	0.750 1.000 0.857	0.000 0.000 0.000
65	0.250 1.000 0.400	0.125 1.000 0.222	0.750 1.000 0.857	0.000 0.000 0.000
66	0.188 1.000 0.316	0.125 1.000 0.222	0.688 1.000 0.815	0.000 0.000 0.000
67	0.188 1.000 0.316	0.250 1.000 0.400	0.625 1.000 0.769	0.000 0.000 0.000
68	0.188 1.000 0.316	0.250 1.000 0.400	0.562 1.000 0.720	0.000 0.000 0.000
69	0.188 1.000 0.316	0.250 1.000 0.400	0.562 1.000 0.720	0.000 0.000 0.000
70	0.250 1.000 0.400	0.438 1.000 0.609	0.375 1.000 0.545	0.000 0.000 0.000
71	0.188 1.000 0.316	0.000 0.000 0.000	0.812 1.000 0.897	0.000 0.000 0.000
72	0.250 1.000 0.400	0.188 1.000 0.316	0.562 1.000 0.720	0.000 0.000 0.000
73	0.062 1.000 0.118	0.188 1.000 0.316	0.812 1.000 0.897	0.000 0.000 0.000
74	0.312 1.000 0.476	0.250 1.000 0.400	0.500 1.000 0.667	0.000 0.000 0.000
75	0.312 1.000 0.476	0.188 1.000 0.316	0.625 1.000 0.769	0.000 0.000 0.000
Note: the values in each cell are, in this order, precision recall f1-score				

Epoch	Application	Framework	Library	Plugin
76	0.188 1.000 0.316	0.062 1.000 0.118	0.688 1.000 0.815	0.062 1.000 0.118
77	0.188 1.000 0.316	0.250 1.000 0.400	0.625 1.000 0.769	0.000 0.000 0.000
78	0.188 1.000 0.316	0.188 1.000 0.316	0.688 1.000 0.815	0.000 0.000 0.000
79	0.125 1.000 0.222	0.250 1.000 0.400	0.688 1.000 0.815	0.000 0.000 0.000
80	0.188 1.000 0.316	0.250 1.000 0.400	0.625 1.000 0.769	0.000 0.000 0.000
81	0.125 1.000 0.222	0.125 1.000 0.222	0.812 1.000 0.897	0.000 0.000 0.000
82	0.250 1.000 0.400	0.188 1.000 0.316	0.562 1.000 0.720	0.000 0.000 0.000
83	0.312 1.000 0.476	0.125 1.000 0.222	0.562 1.000 0.720	0.000 0.000 0.000
84	0.250 1.000 0.400	0.125 1.000 0.222	0.562 1.000 0.720	0.062 1.000 0.118
85	0.375 1.000 0.545	0.250 1.000 0.400	0.438 1.000 0.609	0.062 1.000 0.118
86	0.250 1.000 0.400	0.375 1.000 0.545	0.375 1.000 0.545	0.000 0.000 0.000
87	0.312 1.000 0.476	0.250 1.000 0.400	0.438 1.000 0.609	0.062 1.000 0.118
88	0.188 1.000 0.316	0.312 1.000 0.476	0.562 1.000 0.720	0.000 0.000 0.000
89	0.188 1.000 0.316	0.500 1.000 0.667	0.500 1.000 0.667	0.000 0.000 0.000
90	0.125 1.000 0.222	0.438 1.000 0.609	0.562 1.000 0.720	0.000 0.000 0.000
91	0.125 1.000 0.222	0.438 1.000 0.609	0.500 1.000 0.667	0.062 1.000 0.118
92	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667	0.062 1.000 0.118
93	0.250 1.000 0.400	0.250 1.000 0.400	0.562 1.000 0.720	0.000 0.000 0.000
94	0.500 1.000 0.667	0.250 1.000 0.400	0.250 1.000 0.400	0.000 0.000 0.000
95	0.188 1.000 0.316	0.375 1.000 0.545	0.438 1.000 0.609	0.000 0.000 0.000
96	0.312 1.000 0.476	0.312 1.000 0.476	0.500 1.000 0.667	0.000 0.000 0.000
97	0.188 1.000 0.316	0.312 1.000 0.476	0.562 1.000 0.720	0.000 0.000 0.000
98	0.312 1.000 0.476	0.188 1.000 0.316	0.562 1.000 0.720	0.000 0.000 0.000
99	0.312 1.000 0.476	0.188 1.000 0.316	0.500 1.000 0.667	0.000 0.000 0.000
Note: the values in each cell are, in this order, precision recall f1-score				

Table A.2: Test results of fold 3 of our dataset

Epoch	Application	Framework	Library	Plugin
0	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
1	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
2	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
3	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
4	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
5	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
6	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
7	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
8	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
9	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
10	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
Note: the values in each cell are, in this order, precision recall f1-score				

Epoch	Application	Framework	Library	Plugin
11	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
12	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
13	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
14	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
15	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
16	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
17	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
18	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
19	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
20	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
21	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
22	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
23	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
24	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
25	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
26	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
27	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
28	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
29	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
30	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
31	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
32	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
33	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
34	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
35	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
36	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
37	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
38	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
39	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
40	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
41	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
42	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
43	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
44	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
45	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
46	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
47	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
48	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
49	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
50	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
Note: the values in each cell are, in this order, precision recall f1-score				

Epoch	Application	Framework	Library	Plugin
51	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
52	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
53	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
54	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
55	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
56	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
57	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
58	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
59	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
60	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
61	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
62	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
63	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
64	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
65	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
66	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
67	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
68	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
69	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
70	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
71	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
72	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
73	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
74	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
75	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
76	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
77	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
78	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
79	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
80	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
81	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
82	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
83	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
84	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
85	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
86	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
87	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
88	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
89	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
90	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
Note: the values in each cell are, in this order, precision recall f1-score				

Epoch	Application	Framework	Library	Plugin
91	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
92	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
93	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
94	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
95	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
96	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
97	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
98	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
99	0.188 1.000 0.316	0.062 1.000 0.118	0.812 1.000 0.897	0.000 0.000 0.000
Note: the values in each cell are, in this order, precision recall f1-score				

A.4 Training Results of ClassifyHub Dataset

Table A.3: Training results of fold 1 of ClassifyHub dataset

Epoch	Development	Education	Homework	Other
0	0.500 1.000 0.667	0.250 1.000 0.400	0.250 1.000 0.400	0.000 0.000 0.000
1	0.250 1.000 0.400	0.000 0.000 0.000	0.500 1.000 0.667	0.250 1.000 0.400
2	0.250 1.000 0.400	0.250 1.000 0.400	0.250 1.000 0.400	0.250 1.000 0.400
3	0.000 0.000 0.000	0.750 1.000 0.857	0.250 1.000 0.400	0.000 0.000 0.000
4	0.250 1.000 0.400	0.000 0.000 0.000	0.500 1.000 0.667	0.250 1.000 0.400
5	0.250 1.000 0.400	0.500 1.000 0.667	0.250 1.000 0.400	0.000 0.000 0.000
6	0.250 1.000 0.400	0.500 1.000 0.667	0.250 1.000 0.400	0.000 0.000 0.000
7	0.000 0.000 0.000	0.750 1.000 0.857	0.250 1.000 0.400	0.000 0.000 0.000
8	0.000 0.000 0.000	0.250 1.000 0.400	0.500 1.000 0.667	0.250 1.000 0.400
9	0.500 1.000 0.667	0.250 1.000 0.400	0.250 1.000 0.400	0.000 0.000 0.000
10	0.250 1.000 0.400	0.250 1.000 0.400	0.000 0.000 0.000	0.500 1.000 0.667
11	0.500 1.000 0.667	0.500 1.000 0.667	0.000 0.000 0.000	0.000 0.000 0.000
12	0.250 1.000 0.400	0.500 1.000 0.667	0.250 1.000 0.400	0.000 0.000 0.000
13	0.250 1.000 0.400	0.000 0.000 0.000	0.750 1.000 0.857	0.000 0.000 0.000
14	0.250 1.000 0.400	0.000 0.000 0.000	0.250 1.000 0.400	0.500 1.000 0.667
15	0.250 1.000 0.400	0.500 1.000 0.667	0.000 0.000 0.000	0.250 1.000 0.400
16	0.000 0.000 0.000	0.500 1.000 0.667	0.250 1.000 0.400	0.250 1.000 0.400
17	0.000 0.000 0.000	0.250 1.000 0.400	0.500 1.000 0.667	0.250 1.000 0.400
18	0.000 0.000 0.000	0.250 1.000 0.400	0.500 1.000 0.667	0.250 1.000 0.400
19	0.250 1.000 0.400	0.500 1.000 0.667	0.250 1.000 0.400	0.000 0.000 0.000
20	0.000 0.000 0.000	0.500 1.000 0.667	0.500 1.000 0.667	0.000 0.000 0.000
21	0.000 0.000 0.000	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667
22	0.250 1.000 0.400	0.750 1.000 0.857	0.000 0.000 0.000	0.000 0.000 0.000
23	0.000 0.000 0.000	0.500 1.000 0.667	0.250 1.000 0.400	0.250 1.000 0.400
24	0.250 1.000 0.400	0.250 1.000 0.400	0.250 1.000 0.400	0.250 1.000 0.400
25	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667	0.000 0.000 0.000
26	0.000 0.000 0.000	0.000 0.000 0.000	0.500 1.000 0.667	0.500 1.000 0.667
27	0.000 0.000 0.000	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667
28	0.000 0.000 0.000	0.750 1.000 0.857	0.000 0.000 0.000	0.250 1.000 0.400
29	0.000 0.000 0.000	0.250 1.000 0.400	0.000 0.000 0.000	0.750 1.000 0.857
30	0.250 1.000 0.400	0.250 1.000 0.400	0.250 1.000 0.400	0.250 1.000 0.400
31	0.250 1.000 0.400	0.000 0.000 0.000	0.500 1.000 0.667	0.250 1.000 0.400
32	0.000 0.000 0.000	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667
33	0.000 0.000 0.000	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667
34	0.000 0.000 0.000	0.500 1.000 0.667	0.250 1.000 0.400	0.250 1.000 0.400
35	0.250 1.000 0.400	0.000 0.000 0.000	0.500 1.000 0.667	0.250 1.000 0.400
Note: the values in each cell are, in this order, precision recall f1-score				

Epoch	Development	Education	Homework	Other
36	0.500 1.000 0.667	0.000 0.000 0.000	0.500 1.000 0.667	0.000 0.000 0.000
37	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667	0.000 0.000 0.000
38	0.500 1.000 0.667	0.500 1.000 0.667	0.000 0.000 0.000	0.000 0.000 0.000
39	0.000 0.000 0.000	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667
40	0.000 0.000 0.000	0.000 0.000 0.000	0.250 1.000 0.400	0.750 1.000 0.857
41	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667	0.000 0.000 0.000
42	0.250 1.000 0.400	0.750 1.000 0.857	0.000 0.000 0.000	0.000 0.000 0.000
43	0.250 1.000 0.400	0.500 1.000 0.667	0.250 1.000 0.400	0.000 0.000 0.000
44	0.000 0.000 0.000	0.500 1.000 0.667	0.250 1.000 0.400	0.250 1.000 0.400
45	0.000 0.000 0.000	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667
46	0.000 0.000 0.000	0.000 0.000 0.000	1.000 1.000 1.000	0.000 0.000 0.000
47	0.000 0.000 0.000	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667
48	0.000 0.000 0.000	0.750 1.000 0.857	0.250 1.000 0.400	0.000 0.000 0.000
49	0.000 0.000 0.000	0.750 1.000 0.857	0.250 1.000 0.400	0.000 0.000 0.000
50	0.250 1.000 0.400	0.000 0.000 0.000	0.500 1.000 0.667	0.250 1.000 0.400
51	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667	0.000 0.000 0.000
52	0.250 1.000 0.400	0.000 0.000 0.000	0.250 1.000 0.400	0.500 1.000 0.667
53	0.500 1.000 0.667	0.000 0.000 0.000	0.500 1.000 0.667	0.000 0.000 0.000
54	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667	0.000 0.000 0.000
55	0.500 1.000 0.667	0.000 0.000 0.000	0.500 1.000 0.667	0.000 0.000 0.000
56	0.250 1.000 0.400	0.250 1.000 0.400	0.000 0.000 0.000	0.500 1.000 0.667
57	0.500 1.000 0.667	0.000 0.000 0.000	0.250 1.000 0.400	0.250 1.000 0.400
58	0.250 1.000 0.400	0.250 1.000 0.400	0.250 1.000 0.400	0.250 1.000 0.400
59	0.250 1.000 0.400	0.250 1.000 0.400	0.250 1.000 0.400	0.250 1.000 0.400
60	0.250 1.000 0.400	0.250 1.000 0.400	0.000 0.000 0.000	0.500 1.000 0.667
61	0.000 0.000 0.000	0.750 1.000 0.857	0.250 1.000 0.400	0.000 0.000 0.000
62	0.000 0.000 0.000	0.250 1.000 0.400	0.500 1.000 0.667	0.250 1.000 0.400
63	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667	0.000 0.000 0.000
64	0.500 1.000 0.667	0.250 1.000 0.400	0.000 0.000 0.000	0.250 1.000 0.400
65	0.250 1.000 0.400	0.250 1.000 0.400	0.000 0.000 0.000	0.500 1.000 0.667
66	0.250 1.000 0.400	0.250 1.000 0.400	0.250 1.000 0.400	0.250 1.000 0.400
67	0.250 1.000 0.400	0.500 1.000 0.667	0.250 1.000 0.400	0.000 0.000 0.000
68	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667	0.000 0.000 0.000
69	0.750 1.000 0.857	0.000 0.000 0.000	0.250 1.000 0.400	0.000 0.000 0.000
70	0.250 1.000 0.400	0.500 1.000 0.667	0.000 0.000 0.000	0.250 1.000 0.400
71	0.000 0.000 0.000	0.250 1.000 0.400	0.500 1.000 0.667	0.250 1.000 0.400
72	0.500 1.000 0.667	0.250 1.000 0.400	0.250 1.000 0.400	0.000 0.000 0.000
73	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667	0.000 0.000 0.000
74	0.000 0.000 0.000	0.250 1.000 0.400	0.500 1.000 0.667	0.250 1.000 0.400
75	0.500 1.000 0.667	0.000 0.000 0.000	0.250 1.000 0.400	0.250 1.000 0.400
Note: the values in each cell are, in this order, precision recall f1-score				

Epoch	Development	Education	Homework	Other
76	0.000 0.000 0.000	0.250 1.000 0.400	0.500 1.000 0.667	0.250 1.000 0.400
77	0.000 0.000 0.000	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667
78	0.250 1.000 0.400	0.250 1.000 0.400	0.250 1.000 0.400	0.250 1.000 0.400
79	0.500 1.000 0.667	0.250 1.000 0.400	0.250 1.000 0.400	0.000 0.000 0.000
80	0.000 0.000 0.000	0.250 1.000 0.400	0.500 1.000 0.667	0.250 1.000 0.400
81	0.000 0.000 0.000	0.750 1.000 0.857	0.250 1.000 0.400	0.000 0.000 0.000
82	0.250 1.000 0.400	0.500 1.000 0.667	0.250 1.000 0.400	0.000 0.000 0.000
83	0.250 1.000 0.400	0.750 1.000 0.857	0.000 0.000 0.000	0.000 0.000 0.000
84	0.250 1.000 0.400	0.500 1.000 0.667	0.000 0.000 0.000	0.250 1.000 0.400
85	0.250 1.000 0.400	0.500 1.000 0.667	0.250 1.000 0.400	0.000 0.000 0.000
86	0.000 0.000 0.000	1.000 1.000 1.000	0.000 0.000 0.000	0.000 0.000 0.000
87	0.000 0.000 0.000	0.500 1.000 0.667	0.250 1.000 0.400	0.250 1.000 0.400
88	0.250 1.000 0.400	0.250 1.000 0.400	0.250 1.000 0.400	0.250 1.000 0.400
89	0.000 0.000 0.000	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667
90	0.000 0.000 0.000	0.750 1.000 0.857	0.250 1.000 0.400	0.000 0.000 0.000
91	0.500 1.000 0.667	0.000 0.000 0.000	0.000 0.000 0.000	0.500 1.000 0.667
92	0.000 0.000 0.000	0.250 1.000 0.400	0.250 1.000 0.400	0.500 1.000 0.667
93	0.000 0.000 0.000	0.250 1.000 0.400	0.500 1.000 0.667	0.250 1.000 0.400
94	0.250 1.000 0.400	0.000 0.000 0.000	0.250 1.000 0.400	0.500 1.000 0.667
95	0.000 0.000 0.000	0.500 1.000 0.667	0.250 1.000 0.400	0.250 1.000 0.400
96	0.250 1.000 0.400	0.500 1.000 0.667	0.250 1.000 0.400	0.000 0.000 0.000
97	0.500 1.000 0.667	0.500 1.000 0.667	0.000 0.000 0.000	0.000 0.000 0.000
98	0.500 1.000 0.667	0.250 1.000 0.400	0.250 1.000 0.400	0.000 0.000 0.000
99	0.000 0.000 0.000	0.750 1.000 0.857	0.250 1.000 0.400	0.000 0.000 0.000
Note: the values in each cell are, in this order, precision recall f1-score				

Table A.4: Test results of fold 1 of ClassifyHub dataset

Epoch	Development	Education	Homework	Other
0	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
1	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
2	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
3	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
4	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
5	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
6	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
7	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
8	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
9	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
10	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
Note: the values in each cell are, in this order, precision recall f1-score				

Epoch	Development	Education	Homework	Other
11	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
12	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
13	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
14	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
15	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
16	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
17	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
18	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
19	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
20	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
21	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
22	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
23	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
24	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
25	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
26	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
27	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
28	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
29	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
30	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
31	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
32	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
33	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
34	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
35	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
36	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
37	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
38	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
39	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
40	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
41	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
42	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
43	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
44	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
45	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
46	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
47	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
48	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
49	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
50	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000

Note: the values in each cell are, in this order, precision recall f1-score

Epoch	Development	Education	Homework	Other
51	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
52	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
53	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
54	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
55	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
56	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
57	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
58	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
59	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
60	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
61	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
62	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
63	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
64	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
65	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
66	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
67	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
68	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
69	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
70	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
71	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
72	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
73	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
74	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
75	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
76	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
77	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
78	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
79	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
80	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
81	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
82	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
83	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
84	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
85	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
86	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
87	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
88	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
89	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
90	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
Note: the values in each cell are, in this order, precision recall f1-score				

Epoch	Development	Education	Homework	Other
91	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
92	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
93	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
94	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
95	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
96	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
97	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
98	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
99	0.375 1.000 0.545	0.375 1.000 0.545	0.250 1.000 0.400	0.000 0.000 0.000
Note: the values in each cell are, in this order, precision recall f1-score				