A Reproduction of Tailor: Learning Graph-based Code Representations for Source-level Functional Similarity Detection

Jiahao Liu[†*] Jun Zeng[†*] Xiang Wang[‡] Zhenkai Liang[†]
[†]National University of Singapore [‡]University of Science and Technology of China
{jiahao99, junzeng, liangzk}@comp.nus.edu.sg xiangwang@ustc.edu.cn

A. Abstract

This artifact provides the source code of Tailor and scripts to reproduce the experimental results from the ICSE 2023 paper — "Learning Graph-based Code Representations for Source-level Functional Similarity Detection" by Jiahao Liu, Jun Zeng, Xiang Wang, and Zhenkai Liang. All the benchmarks that are used in our evaluation are also included in the artifact. To facilitate the artifact evaluation, we further include the experimental logs when we performed the evaluation. We also provide a docker image that has set up the environment and installed all the dependencies, so the results can be reproduced easily. In this artifact evaluation, we obtain the "Artifact Available" and "Artifact Functional" badges.

B. Introduction

Tailor is a new code representation learning framework tailed to detect code functional similarity. Built upon a customized graph neural network, Tailor models graph-structured code features from code property graphs to provide better program functionality classification. The paper can be found at https://github.com/jun-zeng/Tailor/blob/main/paper.pdf.

C. Artifact check-list (meta-information)

- **Algorithm**: Tailor.
- **Data set**: Benchmarks listed in Table I have been included.
- **Run-time environment**: Ubuntu. We provide a docker image.
- **Hardware**: Two GPUs (each with 32 GB memory) and 64 GB physical memory.
- **Execution**: Run the Python and Bash scripts.
- **Metrics**: Precision, Recall, F1-score, and Accuracy.
- **Output**: Numerical results in text files.
- **How much disk space required (approximately)?**: 10 GB.
- **How much time is needed to prepare workflow (approximately)?**: 15 minutes.
- **How much time is needed to complete experiments (approximately)?**: 16 hours.
- **Publicly available?**: Yes.
- **Archived (provide DOI)?**: doi.org/10.5281/zenodo.7533280.
- **Code licenses (if publicly available)?**: GPL-3.0 license.

D. Description

1) How to access: Our source code, benchmarks, and the scripts to run experiments are publicly available on Zenodo: https://doi.org/10.5281/zenodo.7533280. We also provide a GitHub repository for potential future updates: https://github.com/jun-zeng/Tailor.

2) Hardware dependencies: To run all the experiments and reproduce the results, you need a machine with two GPUs (each with 32 GB memory) and 64 GB physical memory. Note that although it is possible to run our experiments on a CPU machine, you may not finish the training process within two weeks. For example, it takes us over two weeks to run the GGNN experiment using only CPUs. See Tailor/log/gnn_variants/classification_oj_ggnn_cpu.txt for details.

3) Software dependencies: Docker is required to install Tailor with our docker image. Python 3.6, Miniconda3, and CUDA 10.0 are required to install Tailor from scratch.

4) Data sets: : We have included the benchmarks (OJClone and BigCloneBench) in the released repository.

E. Installation

We provide two ways to set up the environment for Tailor.

- **Build on a docker image**: We provide a docker image that have installed all the dependencies.
- **Build on your machine from scratch**: You can install the dependencies manually.

We strongly recommend using our docker image to run the artifact. This is the image we used in our evaluation.

1) Build on a docker image: Please ensure your system has installed Docker. If not, you can install docker by following the instructions. After installing Docker, download our docker image (tailor_image.tar) from Zenodo and load it into your system with:

```
$ docker load < tailor_image.tar
```

Then, initialize a container using the image with:

```
$ docker run -it --gpus all tailor_image
bash
```

Note: If you encounter the problem of "*docker: Error response from daemon: could not select device driver with capabilities: [[gpu]].*", please install the *nvidia-container-toolkit* and *nvidia-docker2* packages. Further, if you meet the problem of "*Unable to locate package nvidia-container-toolkit*", please refer to this solution.

After that, go to Tailor and you are ready to start the experiments:

```
$ cd /home/Tailor
```

2) Build on your machine from scratch: Make sure you have installed Miniconda3 and CUDA 10.0 in your system. If not, please install them following their official installation tutorials:Miniconda and CUDA.

[*]Co-primary authors. Jun Zeng is the corresponding author of the artifact.

Then, create a new conda environment with our provided *environment.yml* file:

```
$ conda env create -f environment.yml
```

After that, activate the environment:

```
$ conda activate tailor
```

Note that you also need to compile Cython:

```
$ cd cpgnn
$ python setup.py build_ext --inplace
```

Now you are ready to start the experiments:

```
$ cd Tailor
```

F. Experiment workflow

1) CPG Construction: We construct code property graphs (CPGs) from the experimental datasets and generate their encodings for the tasks of code clone detection and source code classification.

Unzip Datasets

```
$ cd datasets
$ tar -zxvf ojclone.tar.gz
$ tar -zxvf bigclonebench.tar.gz
$ tar -zxvf ojclassification.tar.gz
```

CPG Construction and Encoding Generation (∼ 1.5 hours)

If you want to skip this procedure, you can download our encoding results from oj_clone_encoding, oj_classification_encoding, and bcb_clone_encoding.

```
$ cd cpg
$ python driver.py --lang c
--clone_classification clone --src_path
../datasets/ojclone --statistics --encoding
--encode_path ../cpgnn/data/oj_clone_encoding
```

```
$ python driver.py --lang c
--clone_classification classification
--src_path ../datasets/ojclassification
--statistics --encoding --encode_path
../cpgnn/data/oj_classification_encoding
```

```
$ python driver.py --lang java
--src_path ../datasets/bigclonebench
--statistics --encoding --encode_path
../cpgnn/data/bcb_clone_encoding
```

After running these commands, you will get the encodings of the experimental datasets in the *cpgnn/data* folder.

Visualize CPGs generated from C and Java programs

To demonstrate how CPGs look like, we provide two toy programs in C and Java in the *cpg/examples* folder. You can visualize their CPGs by running:

```
$ cd cpg/examples
$ python toy_visualization.py --lang c
--path ./isfactor.c --fig_name ./example_c


$ python toy_visualization.py --lang
java --path ./isfactor.java --fig_name
./example_java
```

Two figures will be generated in the *cpg/examples* folder. They are the CPGs constructed from the toy programs.

2) CPGNN Modeling: In this section, we first show how to train CPGNN for code clone detection and source code classification. Then, we present the ablation study of the CPGNN. To facilitate the artifact evaluation, we provide the logs when we run these experiments in the *cpgnn/logs* folder.

Make sure you have generated the following encodings in the *cpgnn/data* folder.

- oj_clone_encoding
- oj_classification_encoding
- bcb_clone_encoding

Train CPGNN for OJ Code Clone Detection (∼ 2 hours)

```
$ cd cpgnn
$ python main_oj.py --clone_test_supervised
--epoch 30 --classification_num 15
--clone_threshold 0.5 --dataset
oj_clone_encoding --type_dim 16 --layer_size
[32,32,32,32,32] --batch_size_clone 512
--gpu_id 0,1 --report clone_oj
```

See Tailor/log/gnn_layer/clone_oj_layer_5.txt to check our experimental log. We report this result in Table II in the paper.

Train CPGNN for OJ Source Code Classification (∼ 8 hours)

```
$ cd cpgnn
$ python main_oj.py --classification_test
--epoch 251 --classification_num 104
--dataset oj_classification_encoding
--type_dim 16 --layer_size [32,32,32,32,32]
--batch_size_classification 384 --gpu_id 0,1
--report classification_oj
```

See Tailor/log/gnn_layer/classification_oj_later_5.txt to check our experimental log. We report this result in Table V in the paper.

Train CPGNN for BCB Code Clone Detection (∼ 4 hours)

```
$ cd cpgnn
$ python main_bcb.py --clone_test_supervised
--epoch 30 --clone_threshold 0.5 --dataset
bcb_clone_encoding --type_dim 16 --layer_size
[32,32,32,32] --batch_size_clone 384 --gpu_id
0,1 --report clone_bcb
```

See Tailor/log/gnn_layer/clone_bcb_layer_4.txt to check our experimental log. We report this result in Table III and IV in the paper.

Ablation Study for Tailor

In this part, we present the workflow of Tailor's ablation study described in Table VI, VII, VIII, and IX in the paper. Here, we introduce how to reproduce the results using our scripts.

Step 1: Copy the artifact script to cpgnn folder, e.g.,

```
$ cp log/gnn_layer/artifact_gnn_layer.py
cpgnn/
```

Step 2: Run the ablation study script, e.g.,

```
$ cd cpgnn
$ python artifact_gnn_layer.py
```

Step 3: Check the experimental log.

```
$ cd cpgnn/log
```

In addition to different numbers of *gnn_layers*, we also investigate the effect of different *cpg_representation*, *embdding_initialization* and *gnn_variants*. You can follow the same workflow to conduct these ablation studies.

Note: We also provide our experimental logs for these ablation studies. You can find them in the *log/gnn_layer*, *log/cpg_representation*, *log/embedding_initialization* and *log/gnn_variants* folders.

G. Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-badging
- http://cTuning.org/ae/submission-20201122.html
- http://cTuning.org/ae/reviewing-20201122.html