Analyzing Software using Deep Learning

– Project Description, Summer Semester 2021 –

Prof. Dr. Michael Pradel, Aryaz Eghbali, Moiz Rauf

June 18, 2021

1 Introduction

Programming languages have a grammar and a defined syntax for source codes in their respective language. For a source code to be interpreted or compiled correctly, the syntax should be valid, and thus compilers and interpreters show errors when the syntax is wrong. Although compilers try to present the developer the location of the error, it is not always easy to show the exact location of the syntax error. Moreover, fixing these errors can be tedious and sometimes hard.

2 Goal

The goal of this project is to design, implement, and evaluate a neural network-based program analysis that localizes and fixes syntactic errors in Python source code. Given a small function, for which the names of identifiers and the values of literals are abstracted, the analysis will find the location of the syntax error and predict a fix for it. To train this model, the approach relies on a given corpus of code, which contains Python functions in two versions: a syntactically correct version and a syntactically incorrect version. The project will focus on Python as the target language, i.e., the resulting tool will fix syntax errors in Python code. The implementation must be in Python (and work with Python version 3.8). The neural network part of the implementation should build on Pytorch (and work with Pytorch version 1.8.1).

3 Dataset

We provide a dataset of Python source code examples extracted from popular GitHub repositories. It consists of about 50k functions. For each function, we provide the following information:

- The original source code.
- An abstracted version of the original source code where all identifiers are replaced with `ID` and all literals are replaced with `LIT`.
- A syntactically incorrect variant of the abstracted source code. There are three kinds of syntax errors considered in this project: a single missing token, a single superfluous token, and a single incorrect token.

There are 100 JSON files in the provided dataset, with the following format. Each JSON file contains a list of 500 objects, each representing a Python function. Here is an example from the dataset for clarification:

```
{
    "code": "def get_pyzmq_frame_buffer(frame):\n    return frame.buffer[:]\n",
    "metadata": {
        "file": "py150_files/data/0rpc/zerorpc-python/zerorpc/events.py",
        "fix_location": 18,
        "fix_type": "delete",
```

```
        "id": 1
    },
    "correct_code": "def ID (ID ):\n    return ID .ID [:]\n",
    "wrong_code": "def ID (ID ):\n    with return ID .ID [:]\n"
}
```

The `correct_code` and `wrong_code` attributes are the abstracted correct code and abstracted wrong code, respectively. The attributes `fix_location`, `fix_type`, and `fix_token` under `metadata` represent the character location, the type of fix, and the fixing token of the syntax error.

4 Milestones

This project has been broken down into three milestones for better guidance throughout the semester. Each milestone has a progress meeting, in which each student meets with her/his mentor, presents the deliverable, and seeks help if needed. Some code templates have been prepared for each milestone, in which you should implement your solutions.

4.1 Milestone 1 (June 17, 2021)

On the progress meeting of this milestone, course participants are expected to have implemented a tokenizer that receives Python code and outputs a sequence of tokens. The tokenizer should split the code into tokens as defined by the Python language. For example, students may use the "tokenizer" module provided in the Python standard library.[1] To test your implementation, run it on several real-world code examples, including code that contains comments (each comment should be represented as a single token). As shown in the template code provided, your implementation in `tokenizer.py` file contains two methods, `to_token_list` and `write_tokens`. The method `to_token_list` takes a Python file as input and returns a list of tokens, and `write_tokens` takes a list of tokens and writes them to a JSON file with the format shown in `sample_tokens.json`.

4.2 Milestone 2 (July 1, 2021)

The expectation for this milestone is that each student has trained a neural network that predicts the location of a syntax error in the input code. For example, consider the following syntactically incorrect Python code:

```
def ID(ID:
  return ID + LIT
```

Your implementation should read the code, represent it as a sequence of tokens, and then predict with the help of a neural model at what location the syntax error is (the character index). In the example, the error is a missing `)` token at index 9.

Specifically, you should present a `Train.py` file, that reads a JSON file formatted as in the provided dataset, and outputs a serialized Pytorch model[2]. You should also present a `Predict.py` file, that takes your serialized model and a Python code example as the input and outputs the first character location of the syntax error of the input code in a JSON file with the template below. A sample output file has been provided alongside the template codes as well.

```
 {
        "metadata": {
                "file": "py150_files/data/0rpc/zerorpc-python/zerorpc/events.py",
                "id": 1
        },
        "predicted_location": 18
}
```

[1] `https://docs.python.org/3/library/tokenize.html`

[2] `https://pytorch.org/tutorials/beginner/saving_loading_models.html`

4.3 Milestone 3 (July 15, 2021)

It is expected that each course participant has trained a neural network that predicts how to fix a syntax error in code.

Similar to Milestone 2, you should present a `Train.py` file, that reads a JSON file formatted as the dataset provided, and outputs a serialized model. You should also present a `Predict.py` file, that takes your serialized model and a Python code example as the input and outputs the fixed Python code, and a JSON file with a list of objects (one for each input function) with the following format.

```
[{
    "metadata": {
      "file": "py150_files/data/0rpc/zerorpc-python/zerorpc/events.py",
      "id": 1
    },
    "predicted_location": 18,
    "predicted_type": "delete"
},
{
    "metadata": {
      "file": "py150_files/data/0rpc/zerorpc-python/zerorpc/events.py",
      "id": 2
    },
    "predicted_location": 8,
    "predicted_type": "modify",
    "predicted_token": "ID"
}]
```

This output means that for the first function (id=1), the token starting at character 18 (zero-based) should be deleted to fix the syntax error. For the second function (id=2), the token starting at character 8 should be modified to `ID`.

The attribute `predicted_location` must be an integer in the range between zero and the maximum character index of the given code. The attribute `predicted_type` must be one of `insert`, `delete`, `modify`. For `insert` and `modify` the `predicted_token` represents the token that should be inserted or that should replace the existing token, respectively.

Note that for the evaluation of this milestone your model might predict a token that fixes the syntax error but is not the same as what is presented as correct answer in the dataset. Therefore, any output that passes the `is_correct` function in `check.py` is considered correct. You can use this function as the evaluation metric for this milestone, as we will use the same function.

5 Mentoring and How to Submit

Each student has a mentor, either Aryaz Eghbali (aryaz.eghbali@iste.uni-stuttgart.de) or Moiz Rauf (moiz.rauf@iste.uni-stuttgart.de). Students must meet their mentor at least three times during the semester, on the dates indicated for the milestones. In addition, students may consult their mentor to resolve any questions, to ensure progress in the right direction, and to help you submit a successful project in time. We also recommend to ask general questions in the Ilias forum.

The deadline for submitting the project is July 23, 2021. This deadline is firm. The submission will be via Ilias and should include:

- A scientific report of at most four pages that summarizes the approach taken and the results obtained.
- A .zip file with your implementation. The implementation must be usable via the scripts we provide as a template. Please do not change the interface of these scripts.

Each person must present the project on the week of July 26–30, 2021, in a short talk, followed by a question and answer session.

The project is individual, i.e., students must work by themselves and not share their solution with anyone else. Any form of collaboration that results in similar or identical solutions being submitted will be punished according to the usual rules for plagiarism.

Grading will be based on the following criteria:

Criterion	Contribution
Progress meetings and final presentation (progress made, clarity, illustration, quality of answers)	25%
Implementation (completeness, documentation)	25%
Results (soundness, reproducibility)	25%
Report (clarity, illustration, discussion and interpretation of the results)	25%