Is Knowledge Graph a Promising Direction in Modeling Code Semantics?

– A Quantitative Analysis of a Machine Interpretable Knowledge Graph for Code –

Zhimin Zhao[1]
[1] *School of Computing, Queen's University, Canada*

***Abstract*—In this quantitative analysis, we would assess the reproducibility and the performance of GraphGen4Code[1]. GraphGen4Code is the first knowledge graph that captures the semantics of code, and its associated artifacts: 1) Nodes in the graph are classes, functions and methods in popular Python modules; 2) Edges indicate function usage and documentation about functions. It is said to power diverse applications covering program search, code understanding, refactoring, bug detection, and code automation.**

I. Introduction

A vast majority of work in the literature[2] has used either tokens or abstract syntax trees as input representations of code. When these input code representations are used for a specific application, the target is usually a distributed representation of code, with a few that build various types of probabilistic graphical models from code. In comparison, GraphGen4Code targets interprocedural data and control flow to create a more comprehensive representation of code and it uses this representation to drive the construction of a knowledge graph of code connected to its textual artifacts.

As shown in Figure 1, Abdelaziz et al. describe a set of generic extraction techniques applying to 1.3M Python files, i.e. 2.3K Python modules, and 47M forum posts drawn from GitHub to generate a knowledge graph with over 2 billion triples.

Abdelaziz et al. show us some initial use cases of GraphGen4Code in code assistance, enforcing best practices, debugging and type inference. The accuracy of the knowledge graph is typically evaluated by computing an averaged accuracy metric on a held-out test set. Rim et al. describe a behavioural testing framework that addresses the shortcoming of standard knowledge graph evaluation metrics. Since there is no strict analysis of the threat to validity in the original paper, we intend a qualitative analysis to assess the performance of GraphGen4Code.

II. Research Question

We propose three research questions to better understand GraphGen4Code.

A. RQ1: How reproducible is the generation of GraphGen4Code?

1) Main Objective: We intend to build a similar knowledge graph by following the step-by-step pipeline specified in Github[1]. Then we compare the generated graph with the original one[2] for future discussion.

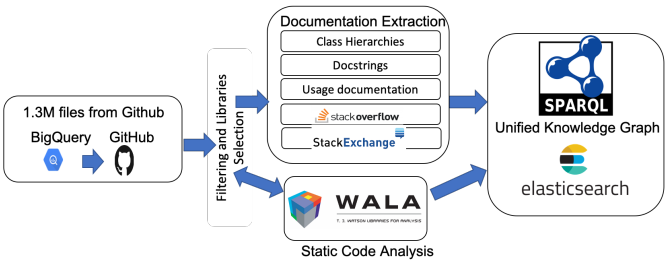

Figure 1: Pipeline to generate large-scale code knowledge graphs.

2) Experiment Setup: First, we would read the specification very closely in case we miss something important.

Secondly, we follow the readme file step by step to create a knowledge graph with the required toolchain, environment, and dataset.

(1) Configure the runtime environment[3] and ensure no version conflict.
(2) Download the dataset[4][5] and read through the specification.
(3) Build graph either on cloud or on my local machine. This step is tricky due to the large data volume. If we build the graph via cloud computing, the cost might be considerable. If we build the graph via my PC, the time might be considerable. I think a tradeoff is necessary for both time and cost, e.g. a build on a subset of the original dataset.
(4) Solve any runtime error if any.

B. RQ2: How is the performance of GraphGen4Code?

1) Main Objective: We would query the built graph for different use scenarios such as code assistance, enforcing best practices, debugging and type inference. Then we can assess the performance of the GraphGen4Code qualitatively.

2) Experiment Setup:

[1] https://github.com/wala/graph4code
[2] https://archive.org/download/graph4codev1
[3] https://github.com/wala/graph4code/blob/master/requirements.txt
[4] https://www.sri.inf.ethz.ch/py150
[5] https://archive.org/details/stackexchange

(1) We set up benchmarks for different programming tasks covering code assistance, enforcing best practices, debugging and type inference.
(2) We collect appropriate test cases for different tasks.
(3) We test the use cases and record the metrics separately.
(4) We compare the metrics with other models for a qualitative conclusion.

C. RQ3: Why is the discrepancy in reproducibility or performance, if any?

1) Main Objective: If there is any discrepancy in reproducibility or performance, we need to figure out why it happens: 1) reading literature on the generation techniques [threat to validity]; 2) checking the toolchain [weaknesses & bugs]; 3) researching the dataset [balance, quality, etc].

2) Experiment Setup:

(1) Peer discussion is necessary before/in/after the aforementioned survey.
(2) Collect and summarize until no feedback is received (or deadline is there).

III. Timeline

Overall, we would loosely follow the timeline as suggested in the CISC 836 course. First, we would familiarize ourselves with the necessary knowledge ready for building the graph by Nov. 21th. Then we would complete the rebuild of the knowledge graph by Nov. 21th. Afterwards, we test the benchmarks for different programming tasks with the original graph by Nov. 28st. Then we finish the research on any discrepancy in reproducibility or performance by Dec 3rd. Last, we would keep on polishing the report and give the presentation by Dec. 6th.

References

[1] I. Abdelaziz, J. Dolby, J. P. McCusker, and K. Srinivas, "A toolkit for generating code knowledge graphs," *The Eleventh International Conference on Knowledge Capture (K-CAP)*, 2021.

[2] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A survey of machine learning for big code and naturalness," *ACM Comput. Surv.*, vol. 51, no. 4, Jul. 2018. [Online]. Available: https://doi.org/10.1145/3212695