Classifying Commits as Bug-Inducing

SENG 474 Formal Proposal (Undergraduate)

Behl, Vibhu
vibhubehl@uvic.ca

Kaberry, James
jameslkaberry@uvic.ca

Mayall, Lucas
lucasmmayall@uvic.ca

Newcombe, Dave
davenewc@uvic.ca

Wilkes, Brennan
bwilkes@uvic.ca

February 13, 2023

1 The Problem

Śliwerski et al. [1] wrote the seminal paper on automatically identifying *fix-inducing changes*. This term would later evolve into the term *bug-inducing commits* [2]. A bug-inducing commit is a commit which introduces a bug into a software system, such that a fix, patch or revision is induced in the future. Naturally there will be an inherit similarity in the *diff* of a bug-inducing commit and it's corresponding *bug-fixing commit* [3]. We propose that this similarity can be recognized by a machine learning model, and as such, bug-inducing and bug-fixing commit pairs can be classified by said model, which we will describe in more detail later in this proposal.

Bug-inducing commits have tremendous teaching value, both in formal research, as well as in industry [4] but are notoriously difficult to automatically identify. Current industry-standard approaches involve mapping easy to generate [5] lists of bug-fixing commits to corresponding lists of bug-inducing commits using the annotation and blame features of *git* [6], or other version control software systems. These approaches are referred to as variants of the *SZZ algorithm* [7], named after Śliwerski et al. Despite widespread usage in industry, and numerous citations in research, even the most optimal of the SZZ variants have been shown to be faulty [8], mislabelling up to 50% of the commits studied. We propose that the SZZ algorithm represents an excellent method for creating lists of *candidate* bug-inducing commits, but that each one of these candidates can be critically evaluated by a machine learning model, and classified *in relation* to a bug-fixing commit as either bug-inducing or not bug-inducing

In order to test implementations of the SZZ algorithm, numerous datasets of bug-inducing and bug-fixing commit pairs have been created and studied [8–17]. We propose that these datasets will make up the basis for our training and testing data, and will represent the *positive* training examples. (It is trivial to generate *negative* training examples, as any two random commits can be paired together.) These datasets will need to be processed and transformed into a format which our model can train upon, but we propose that this can be done trivially using simple automated scripts.

There has been some limited work training machine learning models on bug-inducing commits, notably for the purpose of *JIT compilation*. Nadim et al. [18] proposed a method to *[detect] buggy and non-buggy software commits using five different machine learning classification models*, while Borg et al. [6] used *SZZ Unleased* to train a random-forest to classify bug-inducing commits. Levin et al. [19] used the R caret package in order to classify bug-inducing commits, while Nadim et al. [20] used numerous machine learning techniques to classify bug-inducing commits.

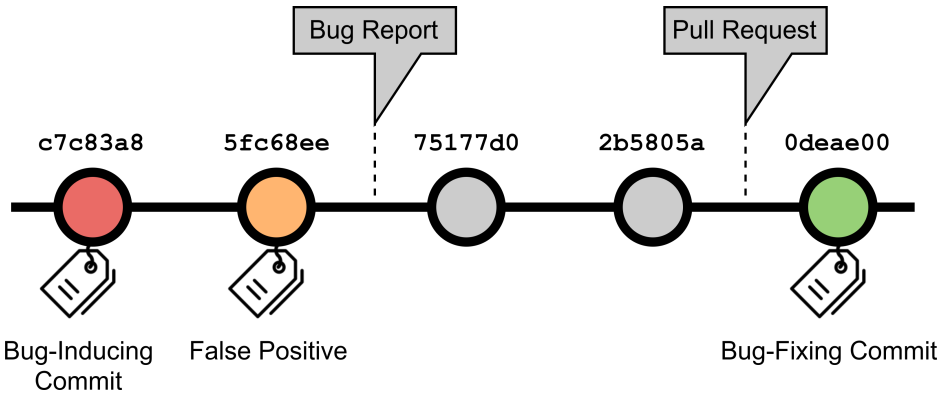

Figure 1: In this example, the SZZ algorithm falsely labels commit `5fc68ee` as the bug-inducing commit. With the addition of a classification model, it may be able to identify `5fc68ee` as a false positive (perhaps the model classifies it as bug inducing with a probability of only 12%), and to continue recursing to find the true bug-inducing commit `c7c83a8` (which the model classifies as bug inducing with sufficient confidence, perhaps a 78% probability).

2 Study Goals

The overarching goal of this study is to contribute to the industry standard methods of identifying bug-inducing commits by using machine learning techniques to improve upon the results of the SZZ algorithm. We propose that a classification model with a testing accuracy of *higher than 50%* would be a useful tool if integrated into an SZZ implementation. It could be used to evaluate the credibility of each candidate bug-inducing commit produced by SZZ (see Figure 1) in order to reduce the false-positive rate of the algorithm [8]. As such, we hope that our model can achieve higher than this minimal threshold of performance. Additionally, incremental improvements in data aggregation, data transformation, and feature vector design are likely to be seen, regardless of the final performance of the model. We do however propose that the integration of our model into an SZZ implementation will be left as future work, and as such suggest limiting the scope of this study to simply designing, training, and testing the model.

3 Study Plan

Principally, we intend to implement a neural network with the goal of, either completely or assisting with, classifying if a bug-inducing and bug-fixing commit pair is valid. Fundamentally this model will take as input the unknown *commit A* and the known bug-fixing *commit B* and will classify *commit A* as being either the corresponding bug-inducing commit which pairs with *commit B*, or not being the corresponding bug-inducing commit. Below we discuss ideas for the specific type of model we hope to use, as well as how we will transform the raw commit information into corresponding feature vectors. However, we wish to emphasise that these ideas are flexible, as we plan to further study works related to *transforming source code tokens into feature vectors* [21–24], *training machine learning models on code changes* [25], and *training machine learning models to identify bugs within software* [26, 27]. In the subsequent sections, we outline the relevant tasks which will be completed by the end of the project.

3.1 Acquisition of Data

One of the first, and most important, tasks in our study will be to acquire and aggregate a suitable dataset. First, *positive* training examples will be acquired from the replication packages of aforementioned studies. Careful consideration will be needed when selecting which of these datasets to pull from. Keshavarz et al. provide a total of 28,239 bug-inducing commits [12], but generated them using the SZZ algorithm itself, so the data's accuracy is questionable. However Rosa et al. provide a much smaller (1,930 bug-inducing commits), but significantly higher quality, dataset which has been manually reviewed [10].

In addition to *positive* training examples, a list of *negative* training examples will need to be generated. In order to improve the accuracy of the model, it is important that these examples cover a variety of possible pairs, including commits modifying completely separate parts of a codebase, as well as commits in close chronological proximity to the correct bug-inducing commit.

3.2 Transformation of Data into Feature Vectors

Multiple studies have recently emerged on the process of training machine learning models on both code blocks and code commits. Alon et al. presented a *neural model for representing snippets of code as continuous distributed vectors* [28] which drew inspiration from Mikolov et al. [29]. Lozoya et al. then continued this line of inquiry in their presentation of *commit2vec* [25] which represented *source code changes* as feature vectors. The *commit2vec* approach (as an extension of the *code2vec* approach) can be summarized as the following:

1. Create a *pre-commit* and *post-commit snapshot* of each method which has changed in a given commit.
2. Build an *abstract syntax tree* for each snapshot.
3. Compute the set of all unique *paths* (also referred to as *contexts*) between each *terminal* of each tree.
4. Encode the set of contexts as a set of vectors using *one-hot encoding*.
5. Compute a single vector of the *weighted-average* of each context using a *path-attention model* [28].

Lozoya et al. saw success utilizing and extending the pre-trained model provided by Alon et al. and we expect to see similar success with a similar approach to the data-encoding step of our study.

Another consideration that will have to be made is whether or not to include the *line number annotations* from the commit diffs. This consideration is novel, due to our inclusion of a second commit in the same input. It is possible that the relationship between the line numbers (and file paths) of each commit diff will be critical for successful classification, and as such these line numbers may be needed as features. Possible solutions may involve weighting the syntax tree nodes with the aforementioned annotation, or potentially including them as an additional vector. Similarly to how Lozoya et al. encode the syntax tree difference, we could consider encoding the line number differences as a separate vector. Doing this would require processing each line number so as to remove the *relative shift* created by intermediate commits. Mapping line numbers from one commit to another has been discussed in a number of studies, and common approaches involve an *annotation graph* [7, 30] or *line number mapping* [31, 32] leveraging Levenshtein-distance. While we don't plan on including this in the feature vector during the initial testing phase, it will potentially be included at a later point.

Finally, we will have to consider how to encode the commit *metadata*, such as the commit author, and commit time. Given our specific goal of comparing two commits, it may be productive to encode the relationship between each commit's metadata as a feature vector, instead of encoding each set of metadata itself. (For example, encoding the two commit authors as a single boolean value representing their equality, instead of the two strings separately.) Going further, the total number of commits made by the author in total, on this particular file, and in other repositories are also potential data points to explore. We intend to

attempt to balance this metadata in such a way that it can provide us with useful insight without causing unnecessary overfitting.

3.3 Model Design

The key novel ideas implemented in our proposed model (when compared to previous works such as *commit2vec* [25] and *code2seq* [24]) will be as follows:

- Classifying based upon the relationship between *two commits*, as opposed to a single commit.
- Classifying based upon commit metadata such as *commit author*, and *commit datetime*.
- Classifying based upon *diff* and, to an extent, *file* relationships within the repository.
- Leveraging and extending a commit-understanding model to classify bug-inducing commits.
- Exploring additional, seemingly inconsequential data points to find hidden relevance.

Some consideration pertaining to which exact model to use is still to be determined, however solutions involving long short-term memory networks [24, 25], transformers [21], and recurrent neural networks [33] will be considered. We will continue to investigate and develop our answer to this question as we make more progress on the project and have a better understanding of what is required of our model to maximize performance.

4 Task Breakdown

In Figure 2 we show a list of reasonable deliverables. The included deadlines not set by the instructor are tentative and subject to change. While we expect that every group member will contribute in some way to every portion of the project, we assign each deliverable a list of *primary* contributors.

Task	**Deadline**	**Primary Contributor(s)**
Data Aggregation	February 20th	James, Vibhu, Lucas
Feature Vector Design	March 1st	Brennan, James
General Model Design	March 1st	Brennan, Dave
Data Transformation Scripts	March 13th	Lucas, Dave, Vibhu
Model Implementation	March 13th	Brennan, Vibhu
Progress Report	**March 13th**	All Group Members
Tuned Hyper-parameters	March 28th	James, Lucas
Presentation	**March 31st**	All Group Members
Final Trained Model	April 7th	Brennan, Dave
Final Report	**April 10th**	All Group Members

Figure 2: Tentative schedule of deliverables. Bold dates represent deadlines set by the instructor.

References

[1] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *ACM sigsoft software engineering notes*, vol. 30, no. 4, pp. 1–5, 2005.

[2] G. Bavota, B. De Carluccio, A. De Lucia, M. Di Penta, R. Oliveto, and O. Strollo, "When does a refactoring induce bugs? an empirical study," in *2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation*. IEEE, 2012, pp. 104–113.

[3] M. Wen, R. Wu, Y. Liu, Y. Tian, X. Xie, S.-C. Cheung, and Z. Su, "Exploring and exploiting the correlations between bug-inducing and bug-fixing commits," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 326–337.

[4] G. An, J. Hong, N. Kim, and S. Yoo, "Fonte: Finding bug inducing commits from failures," *arXiv preprint arXiv:2212.06376*, 2022.

[5] C. Liu, J. Yang, L. Tan, and M. Hafiz, "R2fix: Automatically generating bug fixes from bug reports," in *2013 IEEE Sixth international conference on software testing, verification and validation*. IEEE, 2013, pp. 282–291.

[6] M. Borg, O. Svensson, K. Berg, and D. Hansson, "Szz unleashed: an open implementation of the szz algorithm-featuring example usage in a study of just-in-time bug prediction for the jenkins project," in *Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation*, 2019, pp. 7–12.

[7] S. Kim, T. Zimmermann, K. Pan, E. James Jr *et al.*, "Automatic identification of bug-introducing changes," in *21st IEEE/ACM international conference on automated software engineering (ASE'06)*. IEEE, 2006, pp. 81–90.

[8] S. Herbold, A. Trautsch, F. Trautsch, and B. Ledel, "Problems with szz and features: An empirical study of the state of practice of defect prediction data collection," *Empirical Software Engineering*, vol. 27, no. 2, p. 42, 2022.

[9] E. C. Neto, D. A. d. Costa, and U. Kulesza, "Revisiting and improving szz implementations," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2019, pp. 1–12.

[10] G. Rosa, L. Pascarella, S. Scalabrino, R. Tufano, G. Bavota, M. Lanza, and R. Oliveto, "Evaluating szz implementations through a developer-informed oracle," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 436–447.

[11] L. Bao, X. Xia, A. E. Hassan, and X. Yang, "V-szz: automatic identification of version ranges affected by cve vulnerabilities," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 2352–2364.

[12] H. Keshavarz and M. Nagappan, "Apachejit: a large dataset for just-in-time defect prediction," in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, pp. 191–195.

[13] Y. Fan, X. Xia, D. A. da Costa, D. Lo, A. E. Hassan, and S. Li, "The impact of mislabeled changes by szz on just-in-time defect prediction," *IEEE Transactions on Software Engineering*, vol. 47, no. 8, pp. 1559–1586, 2021.

[14] D. A. da Costa, S. McIntosh, W. Shang, U. Kulesza, R. Coelho, and A. E. Hassan, "A framework for evaluating the results of the szz approach for identifying bug-introducing changes," *IEEE Transactions on Software Engineering*, vol. 43, no. 7, pp. 641–657, 2017.

[15] H. Altinger, S. Siegl, Y. Dajsuren, and F. Wotawa, "A novel industry grade dataset for fault prediction based on model-driven developed automotive embedded software," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, ser. MSR '15. IEEE Press, 2015, p. 494–497.

[16] R. Ferenc, P. Gyimesi, G. Gyimesi, Z. Tóth, and T. Gyimóthy, "An automatically created novel bug dataset and its validation in bug prediction," *Journal of Systems and Software*, vol. 169, p. 110691, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121220301436

[17] R. Ferenc, Z. Tóth, G. Ladányi, I. Siket, and T. Gyimóthy, "A public unified bug dataset for java and its assessment regarding metrics and bug prediction," *Software Quality Journal*, vol. 28, pp. 1447–1506, 2020.

[18] M. Nadim and B. Roy, "Utilizing source code syntax patterns to detect bug inducing commits using machine learning models," *Software Quality Journal*, pp. 1–33, 2022.

[19] S. Levin and A. Yehudai, "Boosting automatic commit classification into maintenance activities by utilizing source code changes," in *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2017, pp. 97–106.

[20] M. Nadim, D. Mondal, and C. K. Roy, "Leveraging structural properties of source code graphs for just-in-time bug prediction," *Automated Software Engineering*, vol. 29, no. 1, p. 27, 2022.

[21] W. U. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "A transformer-based approach for source code summarization," *arXiv preprint arXiv:2005.00653*, 2020.

[22] Y. Liang and K. Zhu, "Automatic generation of text descriptive comments for code blocks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[23] Z. Chen and M. Monperrus, "A literature study of embeddings on source code," *arXiv preprint arXiv:1904.03061*, 2019.

[24] U. Alon, S. Brody, O. Levy, and E. Yahav, "code2seq: Generating sequences from structured representations of code," *arXiv preprint arXiv:1808.01400*, 2018.

[25] R. Cabrera Lozoya, A. Baumann, A. Sabetta, and M. Bezzi, "Commit2vec: Learning distributed representations of code changes," *SN Computer Science*, vol. 2, no. 3, p. 150, Mar 2021. [Online]. Available: https://doi.org/10.1007/s42979-021-00566-z

[26] N. Z. Oishie, "A human-centric approach for adopting bug inducing commit detection using machine learning models," Ph.D. dissertation, University of Saskatchewan, 2022.

[27] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, "Deep learning for just-in-time defect prediction," in *2015 IEEE International Conference on Software Quality, Reliability and Security*. IEEE, 2015, pp. 17–26.

[28] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning distributed representations of code," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–29, 2019.

[29] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[30] M. W. Godfrey and L. Zou, "Using origin analysis to detect merging and splitting of source code entities," *IEEE Transactions on Software Engineering*, vol. 31, no. 2, pp. 166–181, 2005.

[31] C. Williams and J. Spacco, "Szz revisited: verifying when changes induce fixes," in *Proceedings of the 2008 workshop on Defects in large software systems*, 2008, pp. 32–36.

[32] G. Canfora, L. Cerulo, and M. Di Penta, "Identifying changed source code lines from version repositories," in *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*. IEEE, 2007, pp. 14–14.

[33] Y. Xiao and J. Keung, "Improving bug localization with character-level convolutional neural network and recurrent neural network," in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, 2018, pp. 703–704.