Learning to Represent Patches

Xunzhu Tang
xunzhu.tang@uni.lu
University of Luxembourg
Luxembourg

Haoye Tian*
haoye.tian@uni.lu
University of Luxembourg
Luxembourg

Zhenghan Chen
1979282882@pku.edu.cn
Peking University
China

Weiguo Pian
weiguo.pian@uni.lu
University of Luxembourg
Luxembourg

Saad Ezzini
s.ezzini@lancaster.ac.uk
Lancaster University
United Kingdom

Abdoul Kader Kaboré
abdoulkader.kabore@uni.lu
University of Luxembourg
Luxembourg

Andrew Habib
andrew.a.habib@gmail.com
University of Luxembourg
Luxembourg

Jacques Klein
jacques.klein@uni.lu
University of Luxembourg
Luxembourg

Tegawendé F. Bissyandé
tegawende.bissyande@uni.lu
University of Luxembourg
Luxembourg

Abstract

We propose Patcherizer, a novel patch representation methodology that combines context and structure intention features to capture the semantic changes in Abstract Syntax Trees (ASTs) and surrounding context of code changes. Utilizing graph convolutional neural networks and transformers, Patcherizer effectively captures the underlying intentions of patches, outperforming state-of-the-art representations with significant improvements in BLEU, ROUGE-L, and METEOR metrics for generating patch descriptions.

ACM Reference Format:
Xunzhu Tang, Haoye Tian, Zhenghan Chen, Weiguo Pian, Saad Ezzini, Abdoul Kader Kaboré, Andrew Habib, Jacques Klein, and Tegawendé F. Bissyandé. 2024. Learning to Represent Patches. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3639478.3643521

1 Introduction

Recent research has evolved from treating code and patches as mere token sequences [3, 4] to recognizing the importance of code structure, using tools like Abstract Syntax Trees (ASTs) for deeper structural understanding [1, 13]. Simple textual diffs, marked by + and −, fail to fully convey code semantics, leading to approaches like CC2Vec [5] that integrate ASTs for more structured representations. The latest efforts aim to merge token and structure information for enhanced patch representation, with applications such as FIRA's patch description generation [3].

On the one hand, token-based approaches for patch representation [5] lack structural intention information of source code. On the other hand, graph-based representation of patches [7] lacks the context intention which is better represented by the sequence of tokens [5, 10, 12] of the patch itself. Approaches that try to combine context and AST information to represent patches (e.g., FIRA [3]) do not use the intention features of either sequence or graph from the patch but rather rely on representing the code changes while adding some *ad-hoc* annotations to highlight the changes for the model. This paper introduces Patcherizer, a novel approach for patch representation. Patcherizer combines contextual information around code changes with two innovative components: a SeqIntention representation for sequential patches and a GraphIntention representation for structural aspects. These elements enable the utilization of advanced deep learning models, making Patcherizer adaptable and pre-trained for various downstream tasks. The paper presents a comprehensive evaluation of Patcherizer in generating natural language patch descriptions. The contributions of this work are manifold: a novel approach for learning patch representations that merges sequence and graph data, development of a large dataset of 90k parsable patches to facilitate accurate AST difference extraction, and experimental results demonstrate that Patcherizer outperform existing methods and baselines in patch description generation task.

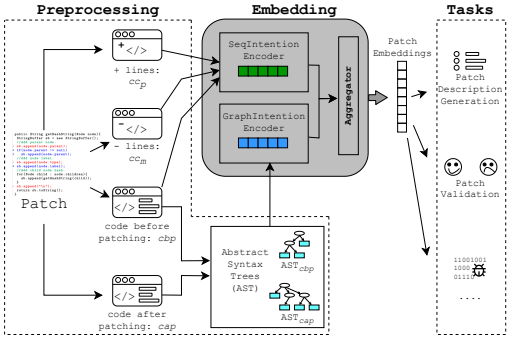


Figure 1: Overview of Patcherizer.

2 Patcherizer

Patcherizer is designed to process software patches by capturing multifaceted data elements. In its preprocessing stage, Patcherizer

*Corresponding author.

ICSE-Companion '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0502-1/24/04
https://doi.org/10.1145/3639478.3643521

meticulously extracts essential information such as the code context prior to the patch application, the specific lines added or removed (the plus and minus lines), and the variations in the Abstract Syntax Tree (AST) graphs due to the patch. This comprehensive approach ensures that both textual and structural changes in the patch are thoroughly analyzed. Patcherizer has two innovative encoders: the Sequence Intention Encoder and the Graph Intention Encoder. The Sequence Intention Encoder, leveraging a Transformer embedding layer ([2]), delves deep into the semantics of sequence changes in the patch. It surpasses conventional line-based analysis by incorporating contextual code data, offering a more nuanced understanding of the patch's impact. Concurrently, the Graph Intention Encoder employs a Graph Convolutional Network (GCN) ([6]) to decipher the structural shifts in the code at a graph level, analyzing the ASTs pre and post-patch application to comprehend the underlying intentions of code alterations. Following the encoding phase, Patcherizer amalgamates the diverse embeddings—encompassing sequence and graph-level data—into a singular, comprehensive representation of the patch. This is achieved through an $\mathcal{A}dd$ function, which effectively merges the SeqIntention embeddings (comprising components like O_{cc_p}, O_{cc_m}, and others) with the GraphIntention embedding $O_{GraphIntention}$. The resulting unified representation, $E_{Patcherizer}$, embodies a holistic view of the patch. Furthermore, Patcherizer is designed with adaptability for a spectrum of downstream software engineering tasks, such as patch description generation. Experimental results indicate that Patcherizer outperforms the state-of-the-art approaches.

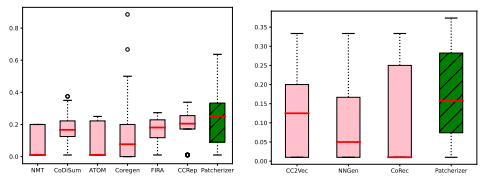
3 Experimental Results

The experiment evaluated Patcherizer's performance on patch description generation task. Using the FIRA dataset and metrics like BLEU, ROUGE-L, and METEOR, Patcherizer was compared against both generative and retrieval-based methods. The results demonstrated that Patcherizer outperformed competing techniques in nearly all metrics, highlighting its effectiveness in both generating new patch descriptions and retrieving existing ones.

Table 1: Performance Results of patch description generation.

Type	Approach	Rouge-L (%)	BLEU (%)	METEOR (%)
Generation	Codisum [12]	19.73	16.55	12.83
	FIRA [3]	21.58	17.67	14.93
	CoreGen [10] (Transformer)	18.22	14.15	12.90
	CCRep [8]	23.41	19.70	15.84
	Patcherizer	25.45	23.52	21.23
Retrieval	CC2Vec [5]	12.21	12.25	11.21
	NNGen [9]	9.16	9.53	16.56
	CoRec [11]	15.47	13.03	12.04
	Patcherizer	17.32	15.21	17.25

"**Generation**" for generation-based strategy.
"**Retrieval**" for retrieval-based approaches.



(a) Generation (b) Retrieval

Figure 2: Comparison of the distributions of BLEU scores for different approaches in patch description generation

4 Conclusion

Patcherizer introduces a novel approach in distributed patch representation learning by integrating contextual, structural, and sequential information from code changes, utilizing Sequence and Graph Intention Encoders for comprehensive patch analysis. Its evaluation on patch description generation shows that Patcherizer significantly outperforms existing baselines and state-of-the-art methods, demonstrating its effectiveness and robustness in patch representation.

Data Availability: https://anonymous.4open.science/r/Patcherizer-1E04

5 Acknowledgments

This work is supported by the NATURAL project, which has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant No. 949014).

References

[1] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. code2vec: learning distributed representations of code. *PACMPL* 3, POPL (2019), 40:1–40:29. https://doi.org/10.1145/3290353

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[3] Jinhao Dong, Yiling Lou, Qihao Zhu, Zeyu Sun, Zhilin Li, Wenjie Zhang, and Dan Hao. 2022. FIRA: Fine-Grained Graph-Based Code Change Representation for Automated Commit Message Generation. (2022).

[4] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[5] Thong Hoang, Hong Jin Kang, David Lo, and Julia Lawall. 2020. Cc2vec: Distributed representations of code changes. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 518–529.

[6] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[7] Bo Lin, Shangwen Wang, Ming Wen, and Xiaoguang Mao. 2022. Context-aware code change embedding for better patch correctness assessment. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 3 (2022), 1–29.

[8] Zhongxin Liu, Zhijie Tang, Xin Xia, and Xiaohu Yang. 2023. CCRep: Learning Code Change Representations via Pre-Trained Code Model and Query Back. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 17–29. https://doi.org/10.1109/ICSE48619.2023.00014

[9] Zhongxin Liu, Xin Xia, Ahmed E Hassan, David Lo, Zhenchang Xing, and Xinyu Wang. 2018. Neural-machine-translation-based commit message generation: how far are we?. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 373–384.

[10] Lun Yiu Nie, Cuiyun Gao, Zhicong Zhong, Wai Lam, Yang Liu, and Zenglin Xu. 2021. CoreGen: Contextualized Code Representation Learning for Commit Message Generation. *Neurocomputing* 459 (2021), 97–107.

[11] Haoye Wang, Xin Xia, David Lo, Qiang He, Xinyu Wang, and John Grundy. 2021. Context-aware retrieval-based deep commit message Generation. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 4 (2021), 1–30.

[12] Shengbin Xu, Yuan Yao, Feng Xu, Tianxiao Gu, Hanghang Tong, and Jian Lu. 2019. Commit message generation for source code changes. In *IJCAI*.

[13] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu. 2019. A novel neural source code representation based on abstract syntax tree. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 783–794.