

# Interpretability of Code Understanding using SHapley Additive exPlanations

XXXXX

Department of Computer Science, Virginia Tech

xxxxx@vt.edu

## Abstract

Large deep learning models have achieved benchmark results on numerous tasks such as image classification, sentiment analysis, speech understanding or strategic game playing. It is equally important to understand and interpret the predictions of these complex models. In recent years, many interpretation tools have been proposed to analyze model predictions across various domains. One such recently booming area is code understanding, however not much work is done for model explainability in this domain. So, in this paper, I investigate the interpretability for clone detection and defect detection – two popular tasks under CodeXGLUE benchmark dataset using SHAPley Additive exPlanations or (SHAP) values. SHAP is one of the widely used interpretation tool to explain the prediction of any model by computing the contribution of each feature for the prediction. To the best of my knowledge, I'm the first one to propose an effective study on the interpretability of deep learning models for code understanding domain.

**Keywords:** Interpretable AI, explainable AI, code understanding networks, deep neural networks

## ACM Reference Format:

Aditya Shah. 2022. Interpretability of Code Understanding using SHapley Additive exPlanations. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 3 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

## 1 Introduction and Related Work

With availability of large scale data and computing resources, more and more complex deep learning architectures have been developed for multiple domains from visual recognition, natural language processing, reinforcement learning, etc. However, due to their nature of overparameterization, it often becomes difficult to understand the predictions of

these models. This lack of interpretability can lead to several issues related to trust and security of these models. Thus understanding model predictions engenders required user trust and provides insight into how a model can be improved further. Various interpretation tools and algorithms have been proposed to explain or reveal the ways in which these deep models make decisions, such as the combination of features used for model decisions, or the importance of every training sample as the contribution for inference.

SHAPley Additive exPlanations proposed by Lundberg and Lee [9] is one of the popular interpretation tool which assigns each feature an importance value for a particular prediction. It's a method derived from coalitional game theory to provide a way to distribute the "payout" across the features fairly. SHAP values provide both global and local explainability and help in understanding how the features of the input data are related to the outputs. This method is used to interpret models for various downstream tasks like image classification, visual question answering, image captioning, sentiment analysis, natural language inference etc. One such recently booming area is code understanding and generation. Lu et al. [8] release CodeXGLUE, a benchmark dataset for code understanding. It consist of a collection of 10 tasks across 14 datasets and a platform for model evaluation and comparison. The authors used BERT-style [3], GPT-style [1], and Encoder-Decoder models [2] as their baseline systems.

In this paper, I investigate the interpretability of deep learning models for two such tasks under CodeXGLUE dataset - clone detection and defect detection using SHAP values and analyze it for 3 popular models - CodeBERT [4], Roberta [7], and GraphCodeBERT [5].

## 2 Methodology

### 2.1 Dataset

The model predictions are analyzed for two tasks:

- **Clone detection:** . The task is to predict whether two given code snippet have same semantics. I use BigCloneBench dataset proposed by Svajlenko et al. [10] for this task. It is a large code clone benchmark that contains over 6,000,000 true clone pairs and 260,000 false clone pairs from 10 different functionalities.
- **Defect detection:** . The task is to detect whether the given source code contains any defect that can potentially harm the system software. For this task, I use the Devign dataset given by Zhou et al. [11] that includes

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

Conference'17, July 2017, Washington, DC, USA

© 2022 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM. . \$15.00

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

27,318 manually-labeled functions collected from two large C programming language open-source projects popular among developers and diversified in functionality, i.e., QEMU and FFmpeg.

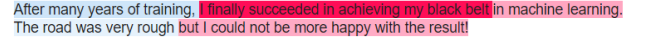
## 2.2 Model

I use 3 popular models - CodeBERT [4], Roberta [7], and GraphCodeBERT [5] and interpret their prediction values on the given datasets. CodeBERT is a transformer encoder model with 12 layers, 768 dimensional hidden states, and 12 attention heads for programming language (PL) and natural language (NL). It is trained using masked language modeling (MLM) and replaced token detection objective on the CodeSearchNet dataset [6], which includes 2.4M functions with document pairs for six programming languages. Roberta model uses a similar BERT based architecture and pretraining approach, but is trained on more dataset for longer time. GraphCodeBERT is another transformer based architecture which uses two structure-aware pre-training tasks in addition to MLM. One task is to predict code structure edges, and the other is to align representations between source code and code structure.

I investigate the model explainability for each of these models on both the tasks - clone and defect detection using SHAP. SHAP values interpret the impact of having a certain value for a given feature in comparison to the prediction we'd make if that feature took some baseline value. To compute this SHAP values, the model is retrained on all feature subsets  $S \subseteq F$  where  $F$  is the set of all features. So, model  $f_{S \cup i}$  is trained with that feature present, and another model  $f_S$  is trained with the feature withheld. The predictions obtained from both the models are compared on the respective inputs:  $f_{S \cup i}(X_{S \cup i}) - f_S(X_S)$  where  $X_S$  represents the input features for the set  $S$ . This comparison helps to find the influence of a respective feature in the subset of all the features. Such differences are computed for all feature subsets  $S \subseteq F$  and the final Shapley values are calculated as a weighted average of all possible differences which is given by:

$$\phi_i = \sum_{S \subseteq F} \frac{|S|!(|F| - |S| - 1)!}{|F|!} [f_{S \cup i}(X_{S \cup i}) - f_S(X_S)]$$

Consider a simple task of sentiment analysis, where given any input text such as 'After many years of training I finally succeeded in achieving my black belt in machine learning. The road was very rough but I could not be more happy with the result!'. we want to identify it's sentiment and analyze which words the model focused on while predicting the outcome. A BERT model classifies this as positive sentiment with 99% confidence. Using SHAP values, we can then determine, which words did the model focus on while predicting the outcome. This is illustrated in Figure 1



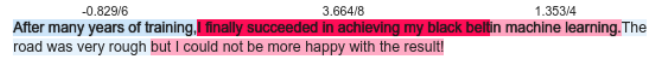
**Figure 1.** Visualizing SHAP values for the given text using heatmap

Here, the text highlighted in pink indicate the words which the model focused on the most while predicting the outcome. Darker shades of pink indicating higher weightage for the respective words. We can infer that the text 'I finally succeeded in achieving my black belt'. indicates a positive sentiment. Likewise the text highlighted in shades of blue indicate words which correspond to negative sentiment.

Through such visualizations, we can effectively analyze the model predictions for clone detection and defect detection tasks. This can help understanding what part of code the model focuses on for the outcome which can be useful to improve security of software systems especially in defect detection task.

## 2.3 Evaluation

SHAP values assign confidence scores to the words of the input text for given output label. In Figure 2, we see that SHAP assigns a high score to the text 'I finally succeeded in achieving my black belt', which contributes the most to the "positive" sentiment.



**Figure 2.** SHAP values for the text corresponding to "Positive" sentiment

For clone and defect detection, I use such confidence scores generated by SHAP and evaluate it's significance through human verification. By identifying such important code snippets in the input code and performing adversarial attacks on them, we can further study the robustness of the respective models.

## 3 Experiments and Outcomes

I conduct two sets of experiments in this paper:

- **Visualization of important code snippets:** I propose to visualize and analyze such predictions for the given 3 models and 2 tasks. Through human verification, we can understand what part of code the model focused on while detecting it as a clone or identifying any potential defect in the input code.
- **Black box adversarial attacks:** Once we find the important words or code snippets that the model focuses on, then we can perform several adversarial attacks to analyze the robustness of the model. Specifically, I perform black box attacks using following methods:

- Substitute such important words with some other similar words and analyze the model prediction.
- Mask out random important words from the code snippet and analyze the model outcomes.
- Add a random gaussian noise matrix  $e, e \sim \mathcal{N}(I, \sigma^2 I)$  in the input representations and interpret the model predictions for this noisy input

It would be interesting to see how the model performs for each of these settings and whether the model is robust enough to such adversarial attacks. Through the discussed evaluation method, we can effectively interpret these models for clone and defect detection.

## 4 Conclusion

In this paper, I propose SHAP values as an effective tool to study the interpretability of these large scale transformer models for code understanding tasks. Understanding the model predictions for a task like defect detection can be helpful to increase the trustworthiness on these so called black-box models and benefit in improving the security of these software systems through reliable AI models.

## References

- [1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. <https://doi.org/10.48550/ARXIV.2005.14165>
- [2] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. <https://doi.org/10.48550/ARXIV.1409.1259>
- [3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. <https://doi.org/10.48550/ARXIV.1810.04805>
- [4] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. <https://doi.org/10.48550/ARXIV.2002.08155>
- [5] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2020. GraphCodeBERT: Pre-training Code Representations with Data Flow. <https://doi.org/10.48550/ARXIV.2009.08366>
- [6] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search. <https://doi.org/10.48550/ARXIV.1909.09436>
- [7] Yinhan Liu, Mylène Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. <https://doi.org/10.48550/ARXIV.1907.11692>
- [8] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. <https://doi.org/10.48550/ARXIV.2102.04664>
- [9] Scott Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. <https://doi.org/10.48550/ARXIV.1705.07874>
- [10] Jeffrey Svajlenko, Judith F. Islam, Iman Keivanloo, Chanchal K. Roy, and Mohammad Mamun Mia. 2014. Towards a Big Data Curated Benchmark of Inter-project Code Clones. In *2014 IEEE International Conference on Software Maintenance and Evolution*. 476–480. <https://doi.org/10.1109/ICSME.2014.77>
- [11] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks. <https://doi.org/10.48550/ARXIV.1909.03496>