Deep Transfer Bug Localization

ABSTRACT

ACM Reference Format:
. 2017. Deep Transfer Bug Localization. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 5 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

An active software project often receives numerous bug reports daily [1]. To resolve each report, developers often need to spend much time and effort [11]. One main task that developers need to do during debugging is to identify code that needs to be fixed to resolve the bug. This task, often referred to as *bug localization*, is a non-trivial one as relevant files need to be identified out of a collection of hundreds or even thousands of files.

To help developers in locating bugs, various automated solutions have been proposed [2, 3, 5, 8–10]. Many of them analyze the description of a bug report to identify source code files relevant to it [2, 5, 8–10]. These text-based solutions can be further divided into two families: unsupervised and supervised. Unsupervised solutions, which were historically proposed first, typically employ information retrieval techniques to identify files that contain many of the words that appear in the bug report [5, 8–10]. More recently, supervised approaches are introduced [2]. These approaches use a collection of bug reports and their relevant source code files as training data. This data is then used to learn a good model that can map new bug reports to their respective relevant source code files. Supervised approaches have been shown to be superior than unsupervised ones.

One limitation of a supervised approach is the need for sufficient good quality training data. Insufficient or low quality data can be detrimental to its effectiveness. This problem is particularly important when a bug localization approach needs to be applied to new projects with limited bug fixing history. Unfortunately, this issue, often referred to as the *cold-start problem*, has not been explored much by past supervised bug localization studies.

To address the above mentioned limitation, in this work we propose *Deep Transfer Bug Localization* (DTBL). DBTL deals with cold-start problem affecting a target software project by adapting data from other projects. DTBL, the first cross-project bug localization solution, combines deep and transfer learning to address the cold-start problem. David says: Please add brief description of the approach.

There have been a number of transfer learning solutions designed to help with cold-start problem in software engineering context. For example, ... Our approach is unique from previous solutions in the following ways. First, ... David says: Please compare the approach with existing work and highlight its novelty.

We have evaluated our proposed solution on ... The experiment results first highlight the need for DBTL as existing solutions cannot effectively make use of data from other projects to create a model that can accurately locate bug in a target project (given a bug report). ... dlPlease add brief description of the results.

Our contributions are as follows:

(1) We present a new direction of research on cross-project bug localization. We highlight that existing supervised bug localization techniques are not able to perform well when they are are trained using data from other projects.
(2) We propose a deep transfer learning solution that .... David says: Please add very brief description of the approach highlighting its novelty.
(3) We have evaluated our proposed approach on ... The results show that ... David says: Please add very brief description of the results.

The structure of the remainder of this paper is as follows. In Section 2 we summarize the state-of-the-art work on supervised bug localization that our approach builds upon. We elaborate the details of our approach in Section 3. The results of the evaluation of the approach are presented in Section 4. We discuss pertinent points and threats to validity in Section 6. We describe related work in Section 7, before concluding and mentioning future work in Section 8.

2 BACKGROUND

3 PROPOSED APPROACH

The goal of cross-project bug localization is using data from source project and a few data from target project to locate the potentially buggy source code in the target project that produce the program behaviors specified in a given bug report. Let $C_s = \{c_{s_1}, c_{s_2}, \cdots, c_{s_{nc_1}}\}$ and $C_t = \{c_{t_1}, c_{t_2}, \cdots, c_{t_{nc_2}}\}$ denote the set of source code from source project and target project respectively, $C = C_s \bigcup C_t$. $\mathcal{R}_s = \{r_{s_1}, r_{s_2}, \cdots, r_{s_{nr_1}}\}$ and $\mathcal{R}_t = \{r_{t_1}, r_{t_2}, \cdots, r_{t_{nr_1}}\}$ denotes the bug reports, respectively, $\mathcal{R} = \mathcal{R}_s \bigcup \mathcal{R}_t$, where nc_1, nc_2, nr_1, nr_2 denote the number of source files and bug reports from source project and target project, respectively. We formulate cross-project bug localization as a learning task which aims to learn a prediction function $f : \mathcal{R} \times C \mapsto \mathcal{Y}$. $y_{ij} \in \mathcal{Y} = \{+1, -1\}$ indicates whether a source code $c_j \in C$ is relevant to a bug report $r_i \in \mathcal{R}$.

In this paper, we propose a novel deep transfer neural network named TRANP-CNN (TRAnsfer Natural and Programm Language Convolutional Neural Network) to instantiate the cross-project bug localization problem, which is an extension of NP-CNN proposed by Huo et al. TRANP-CNN takes the raw data of bug reports and source code as inputs and learns a unified feature mapping

$\phi(\cdot, \cdot)$ for a given r_i and c_j, based on which the prediction can be made with a subsequent output layer. We will introduce the general framework of TRANP-CNN and explain the way to employ deep transfer technique for cross-project bug localization in the following subsections.

3.1 General Structure

The general framework of TRANP-CNN is shown in Figure 1. Specifically, TRANP-CNN consists of four parts: input layer, within-project feature extraction layers, cross-project feature fusion layers and output layers. During the training process, pairs of source code and bug reports and their relevant labels from source project are fed into the deep neural network, as well as very few data from target project, and the model is trained iteratively to optimize the training loss. For testing process, a new bug report from target project containing bugs and its candidate source code are fed into the model, which outputs their relevant scores indicating which code have high relevance to the given bug report and are located as buggy.

3.2 Within-project Feature Extraction Layers

Before processing in Within-project Feature Extraction layers, the source code and bug reports should be firstly encoded as feature vectors. Traditional techniques usually employs TFIDF to represent text content, which may lose the structural and sequential relationships between words and statements. In our model, to maintain the semantics with structural information of text, we apply word2vec technique to encode bug reports and source code.

To process the bug reports in natural language, we follow the standard approach [4] to extract semantic features $\mathbf{h}^r$ from bug reports, which has been widely studied. As mentioned in Huo et al. [2], bug reports and source code should be processed in different ways, because two languages have different structural semantic property.

Source code in programming language, although in textural format, differs from natural language mainly in two aspects. First, the basic language component carrying meaningful semantics in natural language is word or term, and the semantics of the natural language can be inferred from a bag of words. By contrast, in programming language the basic language component carrying meaningful semantics is statement, and the semantics of the programming language can be inferred from the semantics on multiple statements plus the way how these statements interact with each other along the execution path. Thus, to extract features from programming language, the convolution operations should explicitly respect to the atomicity of statements in semantics. Second, natural language organizes words in a "flat" way while programming language organizes its statements in a "structured" way to produce richer semantics. For example, a branching structure "if-then-else" defines two parallel groups of statements. Each group interacts with the statements before and after the branching block while there is no interaction between the two groups. Thus, to extract features from programming language, the convolution operations should obey the program structure defined by the programming languages.

3.3 Cross-project Feature Fusion Layers

After processing from Within-Project Feature Extraction layers, the high-level semantic features from bug reports and source code are extracted, which would be then fed into a fully-connected neural network for feature fusion. However, in cross-project bug localization, the data distribution of source project and target project is different, which means directly employing the same fully-connected network to fuse features from both source and target projects will have bias, leading to a poor bug localization performance.

One question arises here: can we design a particular network to extract features within project, and fuse feature in separate structure? To address this problem, we design two particular fully-connected networks to combine the middle-level features in Cross-project feature fusion layers: One fully-connected network fc_s is used for source project feature fusion and the other fully-connected work fc_t is used to fuse features in the target project. The structure suggests that the feature extraction of different projects are similar and can be processed in the same Convolutional Neural Network, and the feature fusion and projection process is different so that two separate fully-connected neural network are designed to solve this problem. The objective function in the cross-project feature fusion layers can be rewritten in Eq. (1):

$$\arg \min_{\mathbf{W}} \sum_{s_i, s_j} \mathcal{L}(\mathbf{h}_{s_i}, \mathbf{h}_{s_j}, y_{s_{ij}}; W_{fc_s}, W_{conv})) \\ + \sum_{t_i, t_j} \mathcal{L}(\mathbf{h}_{t_i}, \mathbf{h}_{t_j}, y_{t_{ij}}; W_{fc_t}, W_{conv}) + \lambda ||\mathbf{W}||^2 \quad (1)$$

where, $\mathcal{L}$ is the square loss, λ is the trade-off parameter and the weight vectors W contains the weight vectors in convolutional neural networks W_{conv}, in fully-connected network of source domain W_{fc_s} and in fully-connected network of target domain W_{fc_t}. All the weights is learned by minimizing the objective function based on SGD (stochastic gradient descent) in the same time.

4 EXPERIMENTS

To evaluate the effectiveness of TRANP-CNN, we conduct experiments on open source software projects and compare it with several state-of-the-art bug localization methods.

4.1 Research Questions

Our experiments are designed to address the following research questions:

RQ1: *Is there a need for cross-project bug localization?*

RQ2: *Does the cross-project feature fusion layer improve the bug localization performance?*

RQ3: *Can TRANP-CNN outperform other bug localization methods?*

4.2 Data Sets

The data sets are presented here.

4.3 Evaluation Metrics

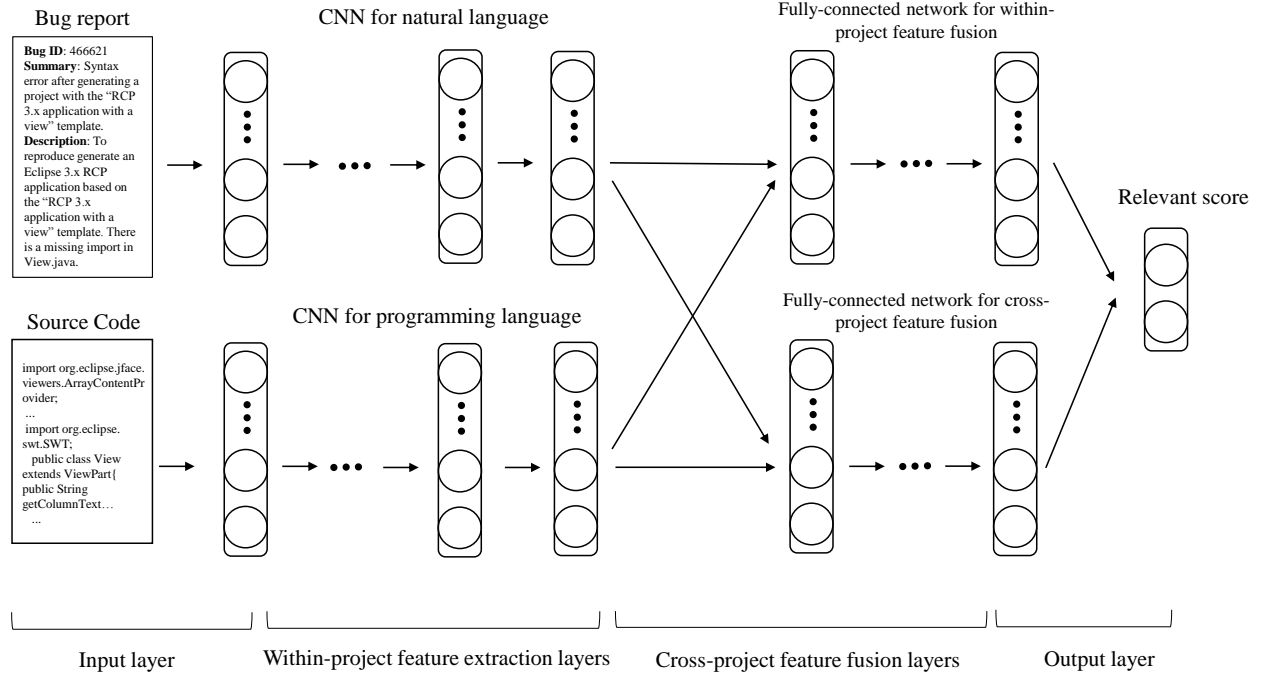The evaluation metrics are presented here.

Figure 1: The overall structure of Transfer Natural and Programming language CNN.

4.4 Compared Methods

We compare our proposed model TRANP-CNN with following baseline methods:

- **VSM (Vector Space Model) [7]**: a baseline method that firstly uses Vector Space Model to represent the text bug reports and source code, then employs Logistic Regression to predict the related buggy source code.
- **Burak (Burak Filter) [6]**: a state-of-the-art method for cross-project and cross-company defect prediction problem, which filters training sets using Burak filter that employs k-nearest neighbour to selects instances in the source project similar to the test project.
- **TCA-R (Transfer Component Analysis with Logistic Regression)**: a state-of-the-art transfer learning method in software engineering, which firstly employ TCA to map source and target project into a same feature space and then apply Logistic Regression for bug localization (same settings suggested in their paper).
- **TCA-P (Transfer Component Analysis with Multi-layer Perceptron)**: a state-of-the-art transfer learning method in software engineering, which firstly employ TCA to map source and target project into a same feature space and then apply MLPs for bug localization (same settings with fully-connected layers in TRANP-CNN).
- **TCA-D (Transfer Component Analysis with Deep features)**: a state-of-the-art transfer learning method in software engineering, which firstly employ TCA to map source and target

project into a same feature space and then apply Logistic Regression for bug localization (using deep features extracted from CNN instead of TFIDF features).
- **NP-CNN (Natural and Programming language Convolutional Neural Network) [2]**: a state-of-the-art deep model for bug localization, which use source project data for training and localizing the buggy source code for target project data.
- **SimpleTrans (Simple Transfer)**: a variant of the TRANP-CNN model, which trains the prediction model on the source project data, and use fine tune method for weight adjustment based on the target project.

4.5 Experimental Settings

For parameter settings of baseline methods, we use the same parameter settings suggested in their work [7]. For the TRANP-CNN model, we employ the most commonly used ReLU $\sigma(x) = \max(0, x)$ as active function and the filter windows size d is set as 3, 4, 5, with 100 feature maps each in Within-Project feature extraction layers. The number of neurons in fully-connect network in set the same number of CNN. In addition, the drop-out method is also applied which is used to prevent co-adaption of hidden units by randomly dropping out values in fully-connected layers, and the drop-out probability p is set 0.25 in our experiments.

For data partition, we use data from source projects and 20% target projects as training sets, and locates the 80% buggy code in target projects. This process repeats for 5 times to reduce the influence of randomness, and we report the average results in the next section.

5 EXPERIMENTAL RESULTS

5.1 Experimental Results for Research Questions

Table 1: Performance Comparisons between within-project and cross-project bug localization.

Tasks	Methods	Top 1	Top 5	Top 10	MAP	MRR
J→H	NPCNN	0.317	0.362	0.508	0.276	0.352
	NPCNN^{partial}	0.204	0.258	0.313	0.202	0.292
	NPCNN^{full}	0.533	0.617	0.650	0.472	0.580
L→H	NPCNN	0.142	0.192	0.345	0.161	0.218
	NPCNN^{partial}	0.204	0.258	0.313	0.202	0.292
	NPCNN^{full}	0.533	0.617	0.650	0.472	0.580
H→J	NPCNN	0.167	0.287	0.349	0.247	0.277
	NPCNN^{partial}	0.035	0.211	0.302	0.155	0.189
	NPCNN^{full}	0.508	0.587	0.679	0.462	0.557
L→J	NPCNN	0.152	0.182	0.318	0.176	0.221
	NPCNN^{partial}	0.035	0.211	0.302	0.155	0.189
	NPCNN^{full}	0.508	0.587	0.679	0.462	0.557
H→L	NPCNN	0.173	0.246	0.390	0.196	0.329
	NPCNN^{partial}	0.097	0.219	0.335	0.095	0.109
	NPCNN^{full}	0.289	0.484	0.611	0.287	0.387
J→L	NPCNN	0.110	0.255	0.323	0.141	0.176
	NPCNN^{partial}	0.097	0.219	0.335	0.095	0.109
	NPCNN^{full}	0.289	0.484	0.611	0.287	0.387
Avg.	NPCNN	0.177	0.254	0.372	0.200	0.262
	NPCNN^{partial}	0.112	0.229	0.317	0.151	0.197
	NPCNN^{full}	0.443	0.563	0.647	0.407	0.508

RQ1: *Is there a need for cross-project bug localization?*

To answer this research question, we compare the results of using NP-CNN for bug localization in different settings.

- NPCNN: Employ NPCNN directly for cross-project bug localization, which means directly training the model on the data from source projects and locating the bugs in the target project.
- NPCNN^{partial}: Employ NPCNN using partial data of target projects, which means training based on a few data (20%) in the target projects, and localizes target buggy files without using data from source project.
- NPCNN^{full}: Employ NPCNN using full data of target projects. In this setting, we conduct 5-folds cross-validation for comparison.

The results are detailed in Tab. 1. There are six tasks in the table, in which **H** represents project *httpclient*, **J** represents project *jackrabbit* and **L** represents *lucene-solr*. Meanwhile, the task **H → J** represents using *httpclient* as source project and predicts the location of buggy files in target project *jackrabbit*. The results show that the performance of bug localization using full data of target projects is the best, which has a large gap against the performance using partial data. For cross-project bug localization, the performance of NPCNN that directly uses source projects is better than NPCNN^{within}, showing that cross-project data is beneficial to improve the bug localization performance, but directly using within-project bug localization technique will not as well as NPCNN^{full}.

The results suggest that there is a need for cross-project bug localization, and directly using within-project bug localization method does not show good performance.

RQ2: *Does the cross-project feature fusion layer improve the bug localization performance?*

To answer this research question, we compare the results of TRANP-CNN with NPCNN and SimpleTrans (Simple Transfer). The difference of the structure between TRANP-CNN and NPCNN is that TRANP-CNN employs two fully-connected networks to combine deep features from source projects and target projects in the cross-project feature fusion layers, respectively, which will counter the influences that cross-project data may have different distribution leading to a bias performance. The results are detailed in Tab. 2.

According to the results, we find that SimpleTrans improves a little performance against NPCNN, showing that transfer technique is effective in improving cross-project bug localization performance. Meanwhile, it can be obviously found that TRANP-CNN performs better than NPCNN and SimpleTrans in terms of all evaluation metrics, which shows that the structure of cross-project feature fusion layer is able to improve cross-project bug localization performance.

Table 2: Performance Comparisons with previous deep models.

Tasks	Methods	Top 1	Top 5	Top 10	MAP	MRR
J→H	NPCNN	0.317	0.362	0.508	0.276	0.352
	SimpleTrans	0.354	0.396	0.563	0.298	0.395
	TRANP-CNN	0.500	0.583	0.625	0.376	0.543
L→H	NPCNN	0.142	0.192	0.345	0.161	0.218
	SimpleTrans	0.163	0.146	0.354	0.141	0.246
	TRANP-CNN	0.275	0.35	0.488	0.242	0.332
H→J	NPCNN	0.167	0.287	0.349	0.247	0.277
	SimpleTrans	0.133	0.324	0.365	0.273	0.301
	TRANP-CNN	0.396	0.443	0.514	0.371	0.434
L→J	NPCNN	0.152	0.182	0.318	0.176	0.221
	SimpleTrans	0.144	0.204	0.382	0.247	0.249
	TRANP-CNN	0.460	0.462	0.488	0.404	0.478
H→L	NPCNN	0.173	0.246	0.390	0.196	0.329
	SimpleTrans	0.197	0.323	0.426	0.152	0.313
	TRANP-CNN	0.361	0.445	0.535	0.279	0.414
J→L	NPCNN	0.110	0.255	0.323	0.141	0.176
	SimpleTrans	0.140	0.282	0.342	0.163	0.224
	TRANP-CNN	0.301	0.410	0.517	0.247	0.368
Avg.	NPCNN	0.177	0.254	0.372	0.200	0.262
	SimpleTrans	0.189	0.279	0.405	0.212	0.288
	TRANP-CNN	0.382	0.449	0.528	0.320	0.428

The results show that TRANP-CNN performs better than NP-CNN and SimpleTrans, which suggests that the cross-projects feature fusion layers can improve the performance of cross-projects bug localization.

RQ3: *Can TRANP-CNN outperform other bug localization methods?*

To answer this research question, we compare the results of TRANP-CNN with state-of-the-art methods: VSM (Vector Space Model), Burak (Burak Filter), TCA-R (Transfer Component Analysis

with Logistic Regression), TCA-P (Transfer Component Analysis with Multi-layer Perceptron). Vector Space Model is a baseline technique used in the within-project bug localization and we employ it on cross-project bug localization for comparison. Burak and TCA have been shown good performance on cross-project and cross-company defect prediction, and also we apply it on cross-project bug localization. The parameters are set suggested in their work, i.e., $k = 2$ in Burak method, and for TCA, we implement the algorithm with TCA+. For fair comparison, the classifier in their original paper (Logistic Regression) and multi-layer perception (same as TRANP-CNN) is compared in our experiments. The results are detailed in Tab. 3.

According to the results, we have several findings: 1. Burak and TCA techniques perform better than the baseline VSM model, indicating that using transfer algorithms is able to improve the performance in cross-project bug localization; 2. TRANP-CNN outperforms TCA-P, which shows that the high-level features extracted from CNN are more semantic and informative, leading to a better representation and bug localization performance; 3. TCA-D uses deep features extracted from CNN and the performance is not as well as TRANP-CNN, which further proves that the cross-project feature fusion layers improve bug localization performance; 4. TRANP-CNN obtains the best average values in terms of all evaluation metrics, suggesting that TRANP-CNN outperforms other traditional bug localization methods and transfer techniques on software engineering.

6 DISCUSSION

6.1 Why do the cross-project feature fusion layers work?

6.2 Why does TRANP-CNN improve the bug localization performance?

6.3 Threats to Validity

7 RELATED WORK

8 CONCLUSION AND FUTURE WORK

REFERENCES

[1] John Anvik, Lyndon Hiew, and Gail C. Murphy. 2005. Coping with an open bug repository. In *Proceedings of the 2005 OOPSLA workshop on Eclipse Technology eXchange, ETX 2005, San Diego, California, USA, October 16-17, 2005.* 35–39.
[2] Xuan Huo, Ming Li, and Zhi-Hua Zhou. 2016. Learning Unified Features from Natural and Programming Languages for Locating Buggy Source Code. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence.* New York, NY, USA, 1606–1612.
[3] James A. Jones and Mary Jean Harrold. 2005. Empirical evaluation of the tarantula automatic fault-localization technique. In *20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), November 7-11, 2005, Long Beach, CA, USA.* 273–282.
[4] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of Conference on Empirical Methods in Natural Language Processing.* Doha, Qatar, 1746–1751.
[5] Stacy K Lukins, Nicholas A Kraft, and Letha H Etzkorn. 2008. Source code retrieval for bug localization using latent dirichlet allocation. In *2008 15th Working Conference on Reverse Engineering.* IEEE, 155–164.
[6] Fayola Peters, Tim Menzies, and Andrian Marcus. 2013. Better cross company defect prediction. In *Proceedings of the 10th IEEE Working Conference on Mining Software Repositories.* IEEE, 409–418.
[7] Shivani Rao and Avinash Kak. 2011. Retrieval from software libraries for bug localization: a comparative study of generic and composite text models. In *Proceedings of the 8th Working Conference on Mining Software Repositories.* ACM, 43–52.

Table 3: Performance comparisons with traditional bug localization performance.

Tasks	Methods	Top 1	Top 5	Top 10	MAP	MRR
J→H	VSM	0.098	0.157	0.177	0.087	0.143
	Burak	0.110	0.126	0.138	0.116	0.121
	TCA-R	0.120	0.212	0.144	0.157	0.162
	TCA-P	0.114	0.133	0.154	0.123	0.176
	TCA-D	0.122	0.225	0.271	0.168	0.248
	TRANP-CNN	0.500	0.583	0.625	0.376	0.543
L→H	VSM	0.059	0.098	0.237	0.099	0.112
	Burak	0.113	0.203	0.242	0.143	0.143
	TCA-R	0.120	0.188	0.244	0.151	0.158
	TCA-P	0.128	0.200	0.252	0.161	0.167
	TCA-D	0.102	0.237	0.367	0.161	0.202
	TRANP-CNN	0.275	0.350	0.488	0.242	0.332
H→J	VSM	0.035	0.211	0.232	0.165	0.129
	Burak	0.130	0.150	0.206	0.225	0.195
	TCA-R	0.115	0.162	0.209	0.239	0.244
	TCA-P	0.114	0.154	0.203	0.237	0.241
	TCA-D	0.111	0.135	0.157	0.168	0.185
	TRANP-CNN	0.396	0.443	0.514	0.371	0.434
L→J	VSM	0.197	0.212	0.293	0.167	0.216
	Burak	0.161	0.132	0.368	0.170	0.187
	TCA-R	0.136	0.183	0.370	0.170	0.179
	TCA-P	0.114	0.116	0.397	0.138	0.191
	TCA-D	0.178	0.236	0.469	0.227	0.256
	TRANP-CNN	0.460	0.462	0.488	0.404	0.478
H→L	VSM	0.083	0.278	0.393	0.154	0.136
	Burak	0.105	0.226	0.272	0.123	0.222
	TCA-R	0.136	0.208	0.383	0.170	0.279
	TCA-P	0.143	0.226	0.394	0.171	0.288
	TCA-D	0.162	0.207	0.345	0.229	0.292
	TRANP-CNN	0.361	0.445	0.535	0.279	0.414
J→L	VSM	0.038	0.077	0.154	0.124	0.204
	Burak	0.138	0.161	0.176	0.168	0.226
	TCA-R	0.135	0.111	0.172	0.169	0.222
	TCA-P	0.136	0.132	0.192	0.173	0.237
	TCA-D	0.142	0.297	0.308	0.238	0.293
	TRANP-CNN	0.301	0.410	0.517	0.247	0.368
Avg.	VSM	0.085	0.172	0.248	0.133	0.157
	Burak	0.126	0.166	0.234	0.157	0.182
	TCA-R	0.127	0.178	0.254	0.176	0.207
	TCA-P	0.125	0.160	0.265	0.167	0.217
	TCA-D	0.136	0.223	0.319	0.199	0.246
	TRANP-CNN	0.382	0.449	0.528	0.320	0.428

[8] Shivani Rao and Avinash C. Kak. 2011. Retrieval from software libraries for bug localization: a comparative study of generic and composite text models. In *Proceedings of the 8th International Working Conference on Mining Software Repositories, MSR 2011 (Co-located with ICSE), Waikiki, Honolulu, HI, USA, May 21-28, 2011, Proceedings.* 43–52.
[9] Shivani Rao, Henry Medeiros, and Avinash C Kak. 2013. An incremental update framework for efficient retrieval from software libraries for bug localization.. In *WCRE.* 62–71.
[10] Ripon K. Saha, Julia Lawall, Sarfraz Khurshid, and Dewayne E. Perry. 2014. On the Effectiveness of Information Retrieval Based Bug Localization for C Programs. In *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014.* 161–170.
[11] G. Tassey. 2002. The Economic Impacts of Inadequate Infrastructure for Software Testing. In *National Institute of Standards and Technology, RTI Project, vol. 7007, no. 011, 2002.*