Investigating the Generalizability of Deep Learning-based Clone Detectors

Eunjong Choi*, Norihiro Fuke†, Yuji Fujiwara†, Norihiro Yoshida‡ and Katsuro Inoue§
*Kyoto Institute of Technology, Japan, echoi@kit.ac.jp
†Osaka University, Japan, {nfuke-cs, y-fujiwr}@ist.osaka-u.ac.jp
‡Ritsumeikan University, Japan, norihiro@fc.ritsumei.ac.jp
§Nanzan University, Japan, inoue599@nanzan-u.ac.jp

***Abstract*—The generalizability of Deep Learning (DL) models is a significant challenge, as poor generalizability indicates that the model has overfitted to the training data and is not able to generalize to new data. Despite numerous DL-based clone detectors emerging in recent years, their generalizability has not been thoroughly assessed. This study investigates the generalizability of three DL-based clone detectors (CCLearner, ASTNN, and CodeBERT) by comparing their detection accuracy on different training and testing clone benchmarks. The results show that all three clone detectors do not generalize well to new data and there is a strong relationship between clone types and generalizability for CCLearner and ASTNN.**

***Index Terms*—code clone, deep learning, generalizability**

I. Introduction

Code clones (in short, clones) are similar or identical code fragments in the source code. Generally, they can be categorized into the following four types based on different levels of similarity [1]. **Type-1 (T1)** clones are identical code fragments except for variations in whitespace, layout, and comments. **Type-2 (T2)** clones are syntactically identical fragments except for variations in identifiers, literals, types, whitespace, layout, and comments. **Type-3 (T3)** clones are copied fragments with further modifications such as changed, added, or removed statements. Finally, **Type-4 (T4)** clones are two or more code fragments that perform the same computation but are implemented by different syntactic variants. The detection of clones plays a significant role in reducing the cost of software maintenance. To achieve this, various clone detectors based on their definitions of similarities have been proposed [1]. Traditional clone detectors, such as CCFinder [2], identify T1 and T2 clones with high accuracy. Meanwhile, they struggle to accurately identify T3 and T4 clones.

In recent years, Deep Learning (DL) has become the predominant approach for various software engineering tasks, including clone detection. DL-based clone detectors use DL models to learn the inherent similarities of clones in clone benchmarks [3]. These detectors are capable of accurately detecting all clone types, including T3 and T4 clones. However, a significant challenge in these detectors is their generalizability [4].

Poor generalizability means that the model has overfitted the training data and is not able to generalize to new, unseen data. If the generalizability of DL-based clone detectors is low, these detectors have a high possibility of inaccurately identifying clones in a software project that differ from the ones in the training dataset. This can negatively impact software maintenance, as incorrect information about clones may lead to wrong decisions about code refactoring or removal. The primary objective of DL-based clone detectors is to accurately detect previously unseen clones during training. Liu et al. studied the generalizability of current DL-based clone detectors and found that these detectors cannot generalize well [5]. However, their findings may have been impacted by the use of a single dataset for both training and testing, as this could have resulted in the presence of similar functionalities in both the training and testing benchmarks, thereby affecting the validity of their results.

To mitigate these problems, in this paper, we investigated the generalizability of DL-based clone detectors to answer the following two research questions:

RQ1. How accurate are DL-based clone detectors when different benchmarks are used for training and testing?

RQ2. Is there any relationship between clone types and detection accuracy when different benchmarks are used for training and testing?

In this paper, we compared the detection accuracy of DL-based clone detectors namely, CCLearner [6], ASTNN [7], and CodeBERT [8] using different clone benchmarks for training and testing. The results indicate that (1) the generalizability of CCLearner, ASTNN, and CodeBERT are not high, and (2) there is a strong relationship between the clone types and the generalizability for CCLearner and ASTNN.

II. Clone Detection

A. Clone Benchmarks

Several clone benchmarks have been generated and used to evaluate the accuracy of clone detectors [3]. **BigCloneBench (BCB)** [9], which has several versions[1], is a widely used clone benchmark that was built by mining clones of frequently implemented functionalities from Open Source Software (OSS) projects. Furthermore, it categorizes T3 and T4 clones according to their syntactic similarity as follows: **Very-strongly type-3 (VST3)** clones that have syntactic similarity [90% 100%), **Strongly Type-3 (ST3)** clones when similarity is in [70%-

[1]https://github.com/clonebench/BigCloneBench

90%), **Moderately Type-3 (MT3)** clones when similarity is in [50% - 70%), and finally, **Weakly Type-3/Type-4 (WT3/T4)** clones when similarity is [0%-50%).

Other clone benchmarks, such as **Google Code Jam (GCJ)** and **CodeNet (CN)** [10], are generated by collecting the same solutions of the programming competitions as clones. *GCJ* is a benchmark collected from an online programming competition held annually by Google. It consists of 1,669 Java files, 275,570 true cloned pairs, and 1,116,376 non-cloned pairs from 12 different competition problems. *CN* is a benchmark collected from competitive programming sites such as *AIZU Online Judge*[2] and *AtCoder*[3]. It contains approximately 14 million total solutions in 55 programming languages.

B. Clone Detectors

To accurately detect T3 and T4 clones, over the past decade, several DL-based clone detectors have been proposed. DL-based clone detectors find out clones by capturing similarities of source code using the clone benchmarks. Li *et al*. proposed **CCLearner** [6], a token-based clone detector using a deep neural network (DNN). CCLearner tokenizes the input source code and then classifies tokens into eight categories. Given a code pair, it computes the similarity score based on their token frequencies. The similarity score is then fed to the DNN model to classify the cloned and non-cloned pair and detects clones using the DNN model. Zhang *et al*. proposed **ASTNN** [7], a code representation learning approach for clone identification tasks. ASTNN splits the abstract syntax tree into a sequence of small statement trees and encodes them into lexical and syntactical vectors. Finally, it generates the vector representation of source code, using a bidirectional gated recurrent, based on the sequence of statement vectors. Feng *et al*. proposed **CodeBERT**, a bimodal pre-trained mode model based on RoBERTa [11]. CodeBERT utilizes both bimodal instances of programming language and natural language pairs and a large number of programming language pairs by using masked language modeling and replaced token detection. After pre-training, fine-tuning is applied on the labeled data.

C. Generalizability of existing DL-based clone detectors

The goal of the DL-based models is to generalize well from the training data to any data from the problem domain. However, the accuracy of the DL-based models is low when the testing data is different from the training data. In many literatures, DL-based clone detectors are trained and tested by splitting the single benchmark into training and testing benchmarks. The accuracy of DL-based clone detectors is also decreased when they detect clones in unseen data [4], [5]. Detection of clones can help to reduce the cost of software maintenance. Therefore, using clone detectors with high accuracy is relevant for developers to maintain software effectively. However, developers sometimes need to detect clones from a new system, which is different from the training dataset. In this case, the DL-based clone detector might provide inaccurate clone information. Various factors, such as coding conventions and token similarity, can affect the generalizability of DL-based clone detectors. However, it is still unclear what factors affect the generalizability of the DL-based clone detectors.

III. Investigations

To mitigate the problems described in Section II-C, we investigated the generalizability of DL-based clone detectors by answering the following two Research Questions (RQ)s :

RQ1. How accurate are DL-based clone detectors when different benchmarks are used for training and testing? To answer RQ1, we first investigated the generalizability of DL-based clone detectors by comparing the accuracy of detectors that were trained and tested on the same benchmark to those that were trained and tested on different benchmarks. If the accuracy of DL-based clone detectors trained and tested on different benchmarks is lower than that of detectors trained and tested on the same benchmark, it indicates that the generalizability of the DL-based clone detector is poor. Therefore, developers need to pay attention to inaccurate clone information provided by DL-based clone detectors when using them for a new software system.

RQ2. Is there any relationship between clone types and detection accuracy when different benchmarks are used for training and testing? By answering RQ2, we can understand the relationship between the generalizability of DL-based clone detectors and different clone types. If the generalizability of DL-based clone detectors is related to the specific clone type, it emphasizes the importance of improving the accuracy of DL-based clone detectors for each specific clone type.

To answer RQ1 and RQ2, using three famous clone benchmarks (Section III-A), we investigated the generalizability of the DL-based clones detectors (Section III-B).

A. Clone benchmarks

In this study, we used three representative clone benchmarks, namely *BCB*, *GCJ*, and *CN*. To accurately compare the detection accuracy, in RQ1, we utilized the *BCB* versions that were used in each paper, *BCB v1* for CCLearner and *BCB ERA v1* for ASTNN. For CodeBERT, we adopted a version of *BCB*, *CodeXGlue* [12]. In RQ2, we used the *BCB ERA v1*, which was used in [7]. Same as in the paper, we computed the accuracy of clone detection by weighted sum result according to the percentage of various clone types. From *GCJ*, we chose 275,570 true cloned pairs and randomly selected the same number of non-cloned pairs. To consolidate the detection of method-level clones, callees are inlined into the main method. Additionally, the source code that handles input/output is removed because it frequently appears in programming contest solutions. If a method is called recursively, it is inlined only once. Moreover, only user-defined methods are inlined. In *CN*, at first, we chose the *Java250* dataset from the *CN* and selected the same number of cloned and non-cloned pairs as *GCJ*. To this end, we sorted the questions based on the ascending order of the problem ID and removed files with a single line and code that required significant changes for the inline. Then,

[2]https://judge.u-aizu.ac.jp/onlinejudge/

[3]https://atcoder.jp

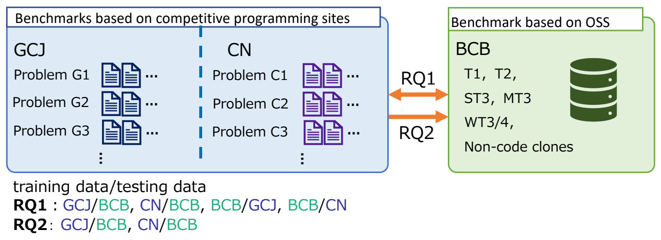

Fig. 1. Overall of the investigations of the generalizability of the DL-based clone detectors

275,570 cloned pairs from the top 12 sets of questions, and the same number of non-cloned pairs were selected. We also inlined callees into the main method by adopting the same approach as *GCJ*.

B. Investigation Methodology

Investigations: In this study, we assumed that *GCJ* and *CN* are similar to each other as they were both derived from data collected from the programming competitions. In contrast, *BCB* was created by mining clones from OSS, making it dissimilar to the *GCJ* and *CN*. Based on these assumptions, we investigated the generalizability of DL-based code detectors using *GCJ*, *CN* and *BCB*. The overall investigations to answer RQ1 and RQ2 are shown in Figure 1. As can be seen in this figure, to answer **RQ1**, we trained CCLearner, ASTNN and CodeBERT on *BCB* and tested them on *GCJ* or *CN*, and vice versa. We then compared these detection results with ones obtained from training and testing on the same benchmark. To answer **RQ2**, we calculated the clone detection accuracy of each clone type by training clone detectors on *GCJ* or *CN* and testing on each clone type in *BCB*. Furthermore, we examined the correlation between the median of Jaccard index for each clone type and the detection accuracy.

Parameters: CCLearner and ASTNN detect clones based on threshold values. In this study, we set threshold values as 0.98 and 0.5 for CCLearner and ASTNN, respectively, the same values as those used in their papers when trained on *BCB*. However, when we used the same threshold values for training on *GCJ* and *CN*, the clone detection accuracy of CCLearner and ASTNN were very low. Thus, we set the threshold through a grid search to identify the best-performing model. We set respectively 0.17 and 0.01 for training CCLearner on *GCJ* and *CN* and 1.1×10^{-4} and 2.8×10^{-4} for training ASTNN on *GCJ* and *CN*.

Evaluation Measurements: We used F_1 and *Matthews Correlation Coefficient* (MCC) to evaluate the detection accuracy of the clone detectors. F_1 is the harmonic mean of precision and recall. MCC is one of the famous measurements to evaluate performance on classically unbalanced data [13]. The MCC is defined as below:

$$MCC = \frac{(tp \times tn) - (fp \times fn)}{\sqrt{(tp + tn)(tp + fn)(fp + tn)(fp + fn)}} \qquad (1)$$

where tp is true positive, tn is true negative, fp is false positive, and fn is false negative. If the MCC value is 1, the detection is accurate. If it is -1, the detection accuracy is exactly opposite, while 0 means that the detectors returns only a forecast no better than a random one.

IV. Investigation Results

A. Results of RQ1

Table I shows the F_1 and MCC values for CCLearner, ASTNN, and CodeBERT using different clone benchmarks and the same benchmark. In this table, the values in the 'SAME' column were computed using the same benchmark for both training and testing, while the values in the 'DIFFERENT' column were computed using different benchmarks for training and testing. Higher values between these columns are highlighted in bold. Furthermore, the F_1 values achieved by training and testing using *BCB* computed by the authors [6]. As we can see in Table I, for CCLearner, when trained and tested on *BCB*, its F_1 value is 0.93. However, when trained on *BCB* and tested on *GCJ* and *CN*, F_1 value drops to 0.109 and 0.357, respectively. When CCLearner is trained and tested on *GCJ* or *CN*, F_1 is approximately 0.55 and MCC ranges from 0.3 to 0.35. Meanwhile,when it trained on *GCJ* or *CN* and tested on *BCB*, the F_1 and MCC values significantly decreases to 0.1 and from 0.05 to 0.1, respectively. For **ASTNN**, when trained and tested on *BCB*, is F_1 value is 0.939 and MCC value is 0.891. Meanwhile, when trained on *BCB* and tested on *GCJ* or *CN*, F_1 is approximately 0.66 and the MCC is approximately ranging from 0.01 to 0.03. When trained and tested using *GCJ* or *CN*, F_1 and MCC is approximately 0.5 and 0.45, respectively. Meanwhile, when trained on *GCJ* and tested on *BCB*, F_1 and MCC is 0.372 and 0.001, respectively. Moreover,when trained on *CN* and tested on *BCB*, F_1 and MCC is 0.09 and -0.15, respectively. Finally, for **CodeBERT**, when trained and tested on *BCB* F_1 and MCC is 0.896 and 0.881, respectively. Meanwhile, when trained on *BCB* and tested on *GCJ* or *CN*, both F_1 and MCC significantly decrease (F_1 is approximately 0.65 and MCC is approximately 0.1). When trained and tested on *GCJ* or *CN*, both of F_1 and MCC are approximately 1. However, F_1 drops significantly to 0.6 when trained on *GCJ* and tested on *BCB*, and to 0.65 when trained on *CN* and tested on *BCB*. In addition, MCC also drops to approximately 0.004.

From our investigations, we can state that the clone detection accuracies of CCLearner, ASTNN, and CodeBERT decrease when they are trained and test on different clone benchmarks, compared to when they are trained and tested on the same clone benchmark.

B. Results of RQ2

Table II shows MCC values for different clone types, which are achieved by training on *GCJ* or *CN* and testing on *BCB*. In the case of **CCLearner**, MCC values for the *T1* clones are the highest compared to other types, while MCC of *WT3/T4*

TABLE I
CLONE DETECTION ACCURACY USING THE SAME AND DIFFERENT CLONE BENCHMARKS

		CCLearner		ASTNN		CodeBERT	
	training benchmark	SAME	DIFFERENT	SAME	DIFFERENT	SAME	DIFFERENT
F_1	BCB	**0.93**	0.109 (GCJ) 0.357 (CN)	**0.939**	0.667 (GCJ) 0.663 (CN)	**0.896**	0.630 (GCJ) 0.651 (CN)
	GCJ	**0.566**	0.116 (BCB)	**0.522**	0.372 (BCB)	**0.998**	0.608 (BCB)
	CN	**0.572**	0.0874 (BCB)	**0.481**	0.0900 (BCB)	**0.999**	0.651 (BCB)
MCC	BCB	–	0.159 (GCJ) 0.296 (CN)	**0.891**	0.0275 (GCJ) 0.0129 (CN)	**0.881**	0.0988 (GCJ) 0.0612 (CN)
	GCJ	**0.340**	0.0847 (BCB)	**0.463**	0.00101 (BCB)	**0.997**	0.00429 (BCB)
	CN	**0.318**	0.0500 (BCB)	**0.434**	-0.150 (BCB)	**0.998**	0.00432 (BCB)

TABLE II
MCC ACHIEVED BY TRAINING AND TESTING USING DIFFERENT CLONE BENCHMARKS FOR EACH CLONE TYPE

	CCLearner		ASTNN		CodeBERT	
Type	GCJ	CN	GCJ	CN	GCJ	CN
T1	0.961	0.967	0.713	0.851	0.000266	-0.00114
T2	0.853	0.911	0.517	0.687	0.00156	0.0166
ST3	0.626	0.699	0.518	0.456	0.00390	0.0000778
MT3	0.462	0.281	0.174	0.0129	-0.00144	0.000612
WT3/T4	0.0745	0.0408	-0.00596	-0.159	0.00438	0.00438

clones are the lowest. As you can see in the table, MCC values decrease as the dissimilarity of clones increases. The correlation coefficient between the median of the Jaccard index and MCC for each clone type is 0.982 when trained on *GCJ* and 0.997 when trained on *CN*. Similarly, MCC values of **ASTNN** show the similar tendencies to CCLearner, MCC values decreases with the increasing dissimilarities of clones, regardless of whether they are trained on *GCJ* or *CN*. The correlation coefficient between the median of the Jaccard index and MCC for each clone type is 0.970 when trained on *GCJ* and 0.995 when trained on *CN*. Finally, **CodeBERT** exhibited low MCC values (approximately 0) when trained on either *GCJ* or *CN*. The correlation coefficient between the median of the Jaccard index and the MCC for each clone type is -0.308 when trained on *GCJ* and 0.217 when trained on *CN*.

From our investigations, we can conclude that CCLearner and ASTNN show a strong relationship between clone types and generalizability, while CodeBERT shows a weak relationship between clone types and generalizability.

V. DISCUSSIONS

From the results of RQ1 and RQ2, we observed that the generalizability of CCLearner, ASTNN, and CodeBERT is not high and clone detection accuracy of CCLearner and ASTNN have a strong relationship with clone types. Meanwhile, the clone detection accuracy of CodeBERT shows a weak relationship with clone types. Based on these results, we suggest the following when developers detect clones from a software project that is distributed differently from the training benchmark by using CCLearner, ASTNN, and CodeBERT: (1) Developers must check detected clones, as the generalizability of these clone detectors is low. (2) If developers trained CCLearner or ASTNN on a clone benchmark containing lower similar clones, such as *T3* and *T4* clones, the detector accurately detects *T1* and *T2* clones. However, since the generalizability of *T3* and *T4* clones is low, developers need to check the accuracy of the detected *T3* and *T4* clones.

Threats to Validity: The results of the investigation are highly dependent on the clone benchmarks and DL-based clone detectors used in this study. In the investigation, we selected *BCB*, *GCJ*, and *CN* because they are widely used clone benchmarks. Regarding *BCB*, we used the version of *BCB* used in the papers because we believe that they were trained on *BCB* for the best accuracy. Similarly, F_1 has been commonly used in clone research for evaluating clone detection accuracy, and MCC is a commonly used evaluation measurement in several fields such as machine learning, thus we believe that these evaluation measurements do not affect the results of these investigations.

VI. CONCLUSION

In this study, we investigated the generalizability of CCLearner, ASTNN, and CodeBERT. The results show that the generalizability of CCLearner, ASTNN, and CodeBERT are not high. There is a strong relationship between clone types and generalizability for CCLearner and ASTNN, and a weak relationship for CodeBERT.

ACKNOWLEDGMENT

This work was supported by JST, PRESTO Grant Number JPMJPR21PA, Japan. Also, this work was supported by JSPS KAKENHI Grant Numbers JP20K11745, JP19K20240, JP18H04094, and JP21H03416.

References

[1] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Sci. Comput. Program.*, vol. 74, no. 7, pp. 470–495, 2009.

[2] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: a multilinguistic token-based code clone detection system for large scale source code," *EEE Trans. Softw. Eng.*, vol. 28, no. 7, pp. 654–670, 2002.

[3] M. Lei, H. Li, J. Li, N. Aundhkar, and D.-K. Kim, "Deep learning application on code clone detection: A review of current knowledge," *J. Syst. Softw.*, vol. 184, no. C, feb 2022.

[4] H. Yu, W. Lam, L. Chen, G. Li, T. Xie, and Q. Wang, "Neural detection of semantic code clones via tree-based convolution," in *Proc. of ICPC*, 2019, p. 70—80.

[5] C. Liu, Z. Lin, J.-G. Lou, L. Wen, and D. Zhang, "Can neural clone detection generalize to unseen functionalities?" in *Proc. of ASE*, 2021, pp. 617–629.

[6] L. Li, H. Feng, W. Zhuang, N. Meng, and B. Ryder, "CCLearner: A deep learning-based clone detection approach," in *Proc. of ICSME*, 2017, pp. 249–260.

[7] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, "A novel neural source code representation based on abstract syntax tree," in *Proc. of ICSE*, 2019, pp. 783–794.

[8] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, Nov. 2020, pp. 1536–1547.

[9] J. Svajlenko, J. F. Islam, I. Keivanloo, C. K. Roy, and M. M. Mia, "Towards a big data curated benchmark of inter-project code clones," in *Proc. of ICSME*, 2014, pp. 476–480.

[10] R. Puri, D. S. Kung, G. Janssen, W. Zhang, G. Domeniconi, V. Zolotov, J. Dolby, J. Chen, M. Choudhury, L. Decker *et al.*, "Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks," *arXiv preprint arXiv:2105.12655*, 2021.

[11] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[12] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. B. Clement, D. Drain, D. Jiang, D. Tang, G. Li, L. Zhou, L. Shou, L. Zhou, M. Tufano, M. Gong, M. Zhou, N. Duan, N. Sundaresan, S. K. Deng, S. Fu, and S. Liu, "Codexglue: A machine learning benchmark dataset for code understanding and generation," *CoRR*, vol. abs/2102.04664, 2021.

[13] G. Jurman, S. Riccadonna, and C. Furlanello, "A comparison of MCC and CEN error measures in multi-class prediction," *PLoS ONE*, vol. 7, no. 8, 2012.