TransClone: A Language Agnostic Code Clone Detector

Subroto Nag Pinku
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
subroto.npi@usask.ca

Debajyoti Mondal
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
d.mondal@usask.ca

Chanchal K. Roy
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
chanchal.roy@usask.ca

***Abstract*—We introduce TransClone, a language-agnostic clone detection tool that leverages a source-to-source translator and an extended version of the graph matching network to detect clones. This tool accepts a source directory or directories as input and identifies code clones within it. It comes with a command line interface and reports identified clones in a CSV file format. Currently, it supports code written in Java and Python. However, TransClone's modular structure provides a flexible way to augment clone-detection support for multiple languages, as better code translators become available. Our experiment on the benchmark dataset shows that it can detect clones with high precision and recall. The TransClone tool including the datasets and trained model can be found here[1].**

***Index Terms*—Code Clone Detection, Cross-Language Clones, Software Maintenance Tools**

I. Introduction

Code clones are code fragments written in the same or different languages with similar functionalities. It arises due to the natural tendency to reuse existing code by copying-pasting with little or no modification. Code clones are generally categorized into four types [1]. Among these four types of clones, the first three are related to the syntax of the program code whereas the fourth one is concerned about the semantics of program code and is known as semantic clones [1]. There are many clone detection tools, and techniques available for detecting clones in single-language software systems which is specific to one programming language. However, cross-language clones are prevalent in modern multi-language software development environments [2]. In such systems, features and functionalities are often ported from one to another programming language and are also interdependent. Detecting and studying cross-language clones could help in the comprehension and maintenance of such systems including finding bug-introducing change patterns [3] across languages. Detecting clones across different programming languages is difficult due to the difference in representation and underlying structure of the programming language. As more such systems are becoming available, researchers are also looking into innovating clone detection techniques in cross-language settings.

With the recent advancements in deep learning, the field has witnessed a number of high-performing clone detection models that are predominantly built on deep learning architectures. Also, pre-trained deep learning models have demonstrated their effectiveness in representation learning. Among such models, a pre-trained deep learning model Transcoder [4] is a great example that has the ability to transform code snippets between several programming languages, a process known as transcompilation. Pinku et al. [5] utilized Transcoder to augment data to improve cross-language clone detection and combined it with an extended version of a high-performing single-language clone detection model to build a model for cross-language clones [5]. In this work, we built a tool, TransClone, which implements the idea Pinku et al. [5] to leverage Transcompiler in junction with a single-language clone detection model. This opens up possibilities for making use of the existing high-performing models for clone detection in cross-language settings.

[1]https://github.com/subrotonpi/transclone

II. Clone Detection Tools in the Literature

Many tools for single-language clone detection have been proposed in the literature. Compared to that only a few techniques are available for cross-language clone detection [6]. Most of the available tools employ traditional techniques rather than deep learning including single-language clone detectors (e.g., [7], [8]) and cross-language clone detectors (e.g., [9], [10]). These tools are capable of processing real-world software systems and detecting clones in them.

In contrast, deep learning based models are often not released as tools to be used for real-world software systems. For example, high-performing single-language models such as graph matching network [11], ASTNN [12] and cross-language models such as C4 [13] and CLCDSA [14] are not released as tools that can detect clones for a given software system. Using these models in a real-world scenario requires further modification and adaptation from the users. This demands users' expertise including but not limited to preprocessing the software system, creating a suitable representation of the code fragments, creating an appropriate input format for the models, and interpreting the models' output.

In some cases, researchers ensemble multiple models to achieve high performance in clone detection [15]. Although this demonstrates the strength and potential of ensemble learning approaches, it has not yet been deployed as a tool in the real-world clone detection setting. While building TransClone,

we carefully considered the above observations. TransClone reduces such burdens and includes the required steps following the standard practice.

III. The TransClone Clone Detector

We developed TransClone as a command line tool [5]. We also provided the Python script that can be invoked to run the tool. As a result, users of this tool can use it sequentially as a complete clone detection and analysis tool or they can use any particular component. Each component or tool is self-sufficient and able to map the input files to specific output files as defined in the package.

In TransClone, we leveraged the deep learning based pre-trained source-to-source translator, Transcoder [4] to process software systems or files which contain code fragments written in different languages. We call this collection of files a system. Once we have a single-language representation for the system we can process these files further to generate a list of potential clone pairs. Then these pairs are tested through the extended version of graph matching network [11] that gives us a list of predictions for them. These predictions are used for further analysis and to report the result to the user.

Figure-1 shows the overview of the TransClone approach. We notice that there is a step-by-step breakdown in the process. First, the process requires data from a directory or directories which contain the code bases. Second, the preprocessing step reads all code snippets, or more specifically the functions or methods from each file. Third, it creates all of the potential pair combinations. Fourth, if a pair contains code written in two different languages, it translates one of those code fragments to a uniform representation in the same language as the other code fragment in the pair. Finally, the tool calculates a similarity score for clone prediction for a pair, analyzes the predictions and generates a report. TransClone currently supports clone detection of software systems with code written in Java and Python.

We followed a sequential methodology to build TransClone. This was chosen to make each component user-friendly. It can be used sequentially to detect and analyze clones in a software system. Otherwise, a user can use any component of it through the *Python script* in combination with any other existing tool they might want to use. Apart from the initial input which is a directory, all other parts require a CSV (comma-separated value) file. The reason for choosing CSV file as it is a widely popular file format. It is also easily manageable across different programming languages and software. The output of each step is a CSV file and only the last step shows details results in a text file. Outputs from all steps including intermediary steps are logged into a log file for further usage.

IV. Usability and Extensibility of TransClone

TransClone is highly customizable since each of its parts accomplishes a single task and maps one input to one output. It can be used in different ways.

A. Python Script

TransClone can be used as a plugin or module in a wide variety of applications because of its adaptability. It is possible to import each of its distinct components separately, which enables the individualized implementation of the functionalities into programs. This adaptability allows users to make use of particular features as required, expanding the program's capabilities in general. With TransClone, programmers have the flexibility to modify and incorporate its features, ensuring a tailored and efficient implementation within their applications.

```
files = preprocess_system(system_directory)
pairs = preprocess_files(files)
predictions = detect_clones(pairs)
analytics = analyze_predictions(predictions)
```

Listing 1: Using the tool in Python

The user is only required to write four lines of Python code in order to make use of TransClone, as seen in Listing 1. The first two lines are used to carry out an analysis of the subject system or code base, and the third and fourth lines, respectively, are responsible for clone detection and report generation. Programmers will also have the possibility to extend the functionality of TransClone by making modifications to its source code, which will allow for enhanced integration with any other tool.

B. Command Line Interface

TransClone can be used as a command line tool. Users can simply run the *transclone.sh* script to use the tool. To do that, users should keep the subject system in the designated location that is under the *storage/subject_system/* directory and invoke the script. However, users can adjust different parameters through the command line such as the threshold, source and target programming language for Transcoder. The threshold for TransClone ranges between -1 to +1 where -1 denotes the lowest similarity and +1 denotes the highest for a given pair.

```
Usage: ./transclone.sh [--cuda] [--
    transcoder_path <path>] [--src_lang <
    language>] [--tgt_lang <language>] [--
    fragment_to_conv <fragment>] [--threshold <
    value>] [--data <directory>] [--
    subject_system <directory>] [--root <
    directory>] [--src_gmn_path <path>] [--
    pairs <file>]
```

Listing 2: Command line usage

Listing 2 shows the command line usage of TransClone. In particular, these arguments can be passed to TransClone to change any default settings. For example, users can also exploit the pipeline by providing a different version of the Transcoder by passing value to the *transcoder_path* argument. Modifying this can help to extend the tool's support for more

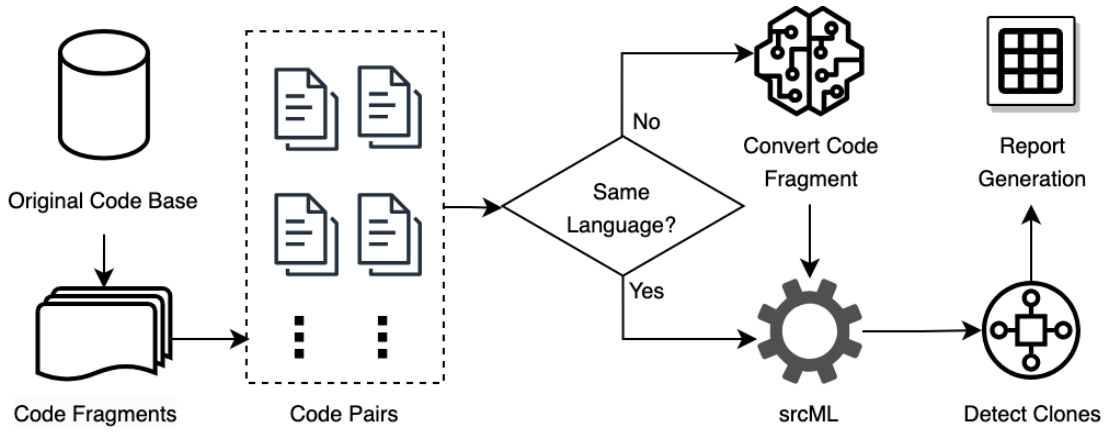

Fig. 1: Overview of TransClone approach

languages. Similarly, a user with expertise in the field can retrain the extended graph matching network model on a different representation or dataset and pass it through the command line argument (i.e., *src_gmn_path*).

V. Demonstration

To demonstrate the efficiency of TransClone, we trained and tested the clone detection model on a benchmark dataset [14]. TransClone is built on top of the proposed method by Pinku et al. [5].

A. Dataset

The CLCDSA [14] dataset provides six programming languages and data from two programming competition websites namely AtCoder and GoogleCodeJam. Our demonstration includes code fragments written in Java and Python. The number of total code fragments in our experiment was around 38 thousand. We split the dataset in an 8:1:1 ratio for train, test, and validation sets. Following existing literature for binary classification problems, we maintained a 1:1 ratio for clone and non-clone pairs [13].

TABLE I: Dataset Summary

	AtCoder		GoogleCodeJam	
Metric	Java	Python	Java	Python
#Problems	1095	1028	261	223
#Average Lines	55	19	73	57
#Code Fragments	15458	14714	4346	3592
#Parsable Fragments	14838	14703	4341	1121
#Unparsable Fragments	620	11	5	2471

The dataset is created by curating solutions to problem sets from online programming competitions. That is, for a given problem statement, the dataset contains solutions in multiple programming languages.

B. Experimental Settings

There are multiple versions of the pre-trained TransCoder available [4]. We chose the best-performing model for our tool. In particular, we chose the TransCoder model specifically trained for Java and Python. We trained the extended graph-matching network on the Java representation of the system. We considered all code fragments while training the extended version of the graph matching network since it can handle both parsable and non-parsable fragments. The model was trained on a machine with RTX-3080ti. For both TransCoder and graph matching networks we had the original hyperparameters [4], [11].

C. Performance

We have chosen the most widely used metrics to evaluate the tool's performance [5]. The precision, and recall of the deployed model of TransClone are 90% and 91% respectively on the test set. These metrics serve as a testament to the model's robustness and accuracy in accurately identifying and classifying code fragments. Our demonstration clearly highlights the efficiency and effectiveness of TransClone in code clone classification. The combination of advanced techniques, a powerful training environment, and extensive evaluation has resulted in this clone detection process.

VI. Discussion

We discuss the different aspects of the TransClone tool in this section.

A. Ease of use

TransClone is a user-friendly and effective clone detection tool. Here are a few important features that contribute to its user-friendliness:

- The application's command line interface lets users communicate with TransClone using simple, common commands and is easy for terminal users.
- Users can input one or more source directories to TransClone which allows users to find clones in their codebase without a cumbersome setup.
- TransClone reports clone detection results in CSV which allow easy interaction with data analysis and visualization tools, making findings interpretation and analysis easier.

- TransClone's GitHub repository provides access to its source code along with well-documented instructions. This guarantees simpler accessibility and adaptation in a variety of situations and needs.

B. Impact of the tool

TransClone has the potential to have a significant and far-reaching impact. The following are key areas where it can make a difference:

- TransClone is a language-agnostic approach and can support multiple programming languages, such as Java and Python and may help build flexible clone management in the cross-language context such as Microsoft .Net [16] with the availability of pre-trained deep learning based translators. .
- The performance of TransClone on the benchmark dataset, as well as its novel method of clone detection, could attract the interest of both academics and the industry. It can be used as a catalyst for further research in the field of software clones.
- TransClone could help study code reuse pattern in cross-language context making it possible to understand development practices, find both code reuse and bug-introducing patterns and build code searching tools over diverse varieties of systems written in different languages.
- Being open source, developers and researchers can enhance the tool, and its accuracy, speed, and language support, making it more valuable for the community.

VII. Limitations

Our approach heavily relies on pre-trained source-to-source translators and the single-language clone detection model being used. As a result, supporting more languages requires a better transcompiler and re-training of the clone detection model. However, this is now common and being improved and accessible day by day significantly. Since different transcompilers support different sets of programming languages, our proposed pipeline can be easily extended. The evaluation datasets consist of programming competition data which is not representative of real-world software data. However, these problems usually require a thoughtful process to solve and often involve critical solutions. As a result, the model faces a diverse range of data. Moreover, finding clones across systems written in different languages would require manual validation as there is a lack of real-world cross-language software data. We plan to evaluate TransClone as a benchmark dataset becomes available in the future.

VIII. Conclusion

In this work, we built and demonstrated TransClone, a cross-language clone detection tool that opens up new possibilities for clone detection in multi-language systems. Our tool's ability to efficiently detect clones in cross-language contexts represents a novel and valuable contribution to software maintenance. Furthermore, its modular design enables simple extensibility, reusability, and customization, making it a flexible platform for future research and experimentation. As we move forward, our vision for TransClone includes integrating a wider range of suitable source-to-source translators and single-language clone detection models to increase its capabilities even further.

Acknowledgment

This work was supported by NSERC Discovery, CFI-JELF, and NSERC CREATE graduate program on Software Analytics Research (SOAR) grants.

References

[1] M. F. Zibran and C. K. Roy, "The road to software clone management: A survey," *Technical Report 2012-03, Department of Computer Science*, p. 1–66, 2012.

[2] P. Mayer, M. Kirsch, and M. A. Le, "On multi-language software development, cross-language links and accompanying tools: a survey of professional software developers," *Journal of Software Engineering Research and Development*, vol. 5, pp. 1–33, 2017.

[3] M. Asaduzzaman, M. C. Bullock, C. K. Roy, and K. A. Schneider, "Bug introducing changes: A case study with android," in *9th Working Conference on Mining Software Repositories (MSR)*, pp. 116–119, 2012.

[4] B. Roziere, M.-A. Lachaux, L. Chanussot, and G. Lample, "Unsupervised translation of programming languages," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[5] S. Pinku, D. Mondal, and C. K. Roy, "Pathways to leverage transcompiler based data augmentation for cross-language clone detection," in *2023 IEEE/ACM 31st International Conference on Program Comprehension (ICPC)*, pp. 169–180, IEEE, 2023.

[6] M. Lei, H. Li, J. Li, N. Aundhkar, and D.-K. Kim, "Deep learning application on code clone detection: A review of current knowledge," *Journal of Systems and Software*, vol. 184, p. 111141, 2022.

[7] J. Svajlenko and C. K. Roy, "Cloneworks: A fast and flexible large-scale near-miss clone detection tool," in *39th International Conference on Software Engineering Companion (ICSE-C)*, pp. 177–179, 2017.

[8] M. S. Uddin, C. K. Roy, and K. A. Schneider, "SimCad: An extensible and faster clone detection tool for large scale software systems," in *2013 21st International Conference on Program Comprehension (ICPC)*, pp. 236–238, 2013.

[9] X. Cheng, Z. Peng, L. Jiang, H. Zhong, H. Yu, and J. Zhao, "Clcminer: detecting cross-language clones without intermediates," *IEICE TRANSACTIONS on Information and Systems*, vol. 100, no. 2, pp. 273–284, 2017.

[10] F. Al-Omari, I. Keivanloo, C. K. Roy, and J. Rilling, "Detecting clones across microsoft .net programming languages," in *2012 19th Working Conference on Reverse Engineering*, pp. 405–414, 2012.

[11] W. Wang, G. Li, B. Ma, X. Xia, and Z. Jin, "Detecting code clones with graph neural network and flow-augmented abstract syntax tree," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 261–271, IEEE, 2020.

[12] 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 *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 783–794, IEEE, 2019.

[13] C. Tao, Q. Zhan, X. Hu, and X. Xia, "C4: Contrastive cross-language code clone detection," in *2022 30th IEEE/ACM International Conference on Program Comprehension (ICPC)*, pp. 413–424, 2022.

[14] K. W. Nafi, T. S. Kar, B. Roy, C. K. Roy, and K. A. Schneider, "CLCDSA: cross language code clone detection using syntactical features and api documentation," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1026–1037, IEEE, 2019.

[15] S. M. Rabbani, N. Ahmad Gulzar, S. Arshad, S. Abid, and S. Shamail, "A comparative analysis of clone detection techniques on semantic-clonebench," in *2022 IEEE 16th International Workshop on Software Clones (IWSC)*, pp. 16–22, 2022.

[16] M. F. Zibran and C. K. Roy, "Towards flexible code clone detection, management, and refactoring in ide," in *Proceedings of the 5th International Workshop on Software Clones*, IWSC '11, p. 75–76, Association for Computing Machinery, 2011.