CloReCo: Benchmarking Platform for Code Clone Detection

Franz Burock[1], Wolfram Amme[1], Thomas S. Heinze[2] and Elisabeth Ostryanin[1]
[1]*Friedrich Schiller University Jena, Jena, Germany*
[2]*Cooperative University Gera-Eisenach, Gera, Germany*

Keywords: Benchmark, Container, Reproducibility, Code Clone, Code Clone Detection.

Abstract: In this paper, we present the Clone Recognition Comparison (CloReCo) platform which supports an uniform performance analysis of code clone detectors. While there exists various benchmarks for code clone detection, these benchmarks on their own have limitations, so that the idea of using multiple benchmarks is promoted. Such a more comprehensive evaluation however requires a benchmarking platform, which integrates the different benchmarks and tools. The CloReCo platform addresses this challenge by implementing a container infrastructure, providing consistent environments for multiple benchmarks and clone detectors. Using CloReCo's web interface or command line interface then allows for conducting performance experiments, adding new clone detectors or benchmarks, managing or analyzing experimental results and thereby facilitates reproducibility of performance analysis by researchers and practitioners in the area of code clone detection.

1 INTRODUCTION

Copy & paste is a pervasive practice in software development and code fragments which are plainly duplicated or rather copied and modified are frequently found in software repositories (Inoue and Roy, 2021). In the evolution of code, such *code clones* can further syntactically deviate from their origin and identifying code clones becomes harder, though important, e.g., for remediating software bugs or vulnerabilities which have been propagated conjoined with the code clones. There exists a large variety of tools for finding code clones, so-called *clone detectors*. On the one hand, these tools usually find exact or near-miss clones, i.e., code fragments which either equal the original code or only exhibit minor modifications, with reasonable performance (Inoue and Roy, 2021; Roy et al., 2009). On the other hand, developing clone detectors for finding code clones which feature larger syntactical deviations is an open problem.

Development in code clone detection requires benchmarks for analyzing and comparing the performance of tools for clone detection. Besides the tools' abilities in finding real code clones and distinguishing them from spurious ones (i.e., the tools' recall and precision), such benchmarks need to focus on the tools' capabilities in finding code clones in large code bases comprising multiple million lines of code in a timely fashion (i.e., runtime and scalability). Note that defining such a benchmark is not trivial, reasoned by the rather fuzzy definition of what constitutes a code clone which is based upon a a given similarity function (Roy et al., 2009). This similarity function can include syntactical as well as functional similarity and the minimum similarity required for a code clone can be disputed, such that the concrete definition of code clones usually resorts to human judgement. *BigCloneBench/BigCloneEval* provides such a state-of-the-art benchmark for code clone detection for Java (Svajlenko and Roy, 2015; Svajlenko and Roy, 2016; Svajlenko and Roy, 2022). The benchmark consists of more than 8 million code clone samples with different levels of syntactical similarity, ranging from exact code clones to code clones with more than 50% syntactical deviation, assembled among over 250 million lines of Java code clones from real-world software projects.

Unfortunately, BigCloneBench has certain limitations, e.g., missing capability to estimate precision, flawed quality of ground truth, bias towards certain functionalities (Svajlenko and Roy, 2022; Krinke and Ragkhitwetsagul, 2022). Developers of clone detectors therefore often use their own benchmark or combine several benchmarks in the tools' evaluations. Other complementary benchmarks though stem from programming contests (e.g., *Google Code-Jam*, *Project CodeNet* (Puri et al., 2021)), which may not generalize to, e.g., industrial code. In conjunc-

Burock, F., Amme, W., Heinze, T. S. and Ostryanin, E.
CloReCo: Benchmarking Platform for Code Clone Detection.
DOI: 10.5220/0013644900003964
In *Proceedings of the 20th International Conference on Software Technologies (ICSOFT 2025)*, pages 394-399
ISBN: 978-989-758-757-3; ISSN: 2184-2833

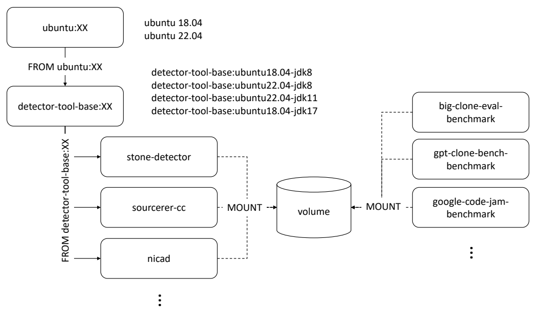

ubuntu:XX
ubuntu 18.04
ubuntu 22.04
FROM ubuntu:XX
detector-tool-base:XX
detector-tool-base:ubuntu18.04-jdk8
detector-tool-base:ubuntu22.04-jdk8
detector-tool-base:ubuntu22.04-jdk11
detector-tool-base:ubuntu18.04-jdk17
FROM detector-tool-base:XX
stone-detector
sourcerer-cc
nicad
MOUNT
volume
MOUNT
big-clone-eval-benchmark
gpt-clone-bench-benchmark
google-code-jam-benchmark

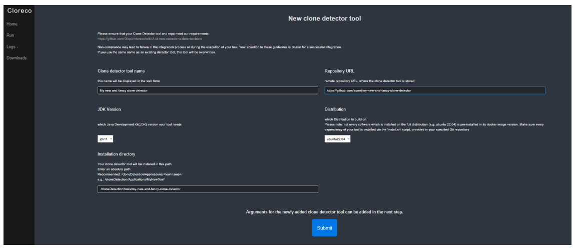
Figure 1: Components of the *CloReCo* platform.

tion with publicly unavailable implementations of clone detectors, inconsistent or incomplete information about tool and benchmark configurations, comparability and reproducibility of performance analysis for clone detection is cumbersome and often tends to be flawed. This becomes an even larger problem considering the current interest in Deep learning. Deep learning approaches have been used for code clone detection in recent years (Wei and Li, 2017; Wang et al., 2020; Guo et al., 2021). In order to assess their generalizability and performance atop the data used in their training, additional data reflecting a wide spectrum of code clones is needed. As is emphasized by other research (Schäfer et al., 2022; Krinke and Ragkhitwetsagul, 2022; Sonnekalb et al., 2022), using BigCloneBench or other benchmarks alone is insufficient and can lead to bias and impaired performance analysis of code clone detectors.

In this paper, we present our novel *Clone Recognition Comparison (CloReCo)* platform (Burock, 2024; Ostryanin, 2024)[1]. CloReCo allows for the uniform evaluation and comparison of the performance of code clone detectors on multiple benchmarks. To this end, CloReCo implements a Docker-based container infrastructure for supporting scriptable and consistent benchmarking environments. Based upon containers, researchers and practitioners can share and automate benchmarking environments in an uniform manner, implying more coherent and reproducible evaluation results. CloReCo supports the following use cases:

- Code clone detection performance analysis on preregistered benchmarks: *BigCloneBench* (Svajlenko and Roy, 2015), *Google CodeJam*, *GPT-CloneBench* (Alam et al., 2023), *Project CodeNet* (Puri et al., 2021) and clone detectors: *NiCad* (Roy and Cordy, 2008), *CCAligner* (Wang et al., 2018), *StoneDetector* (Amme et al., 2021; Schäfer et al., 2020) *Oreo* (Saini et al., 2018), *SourcererCC* (Saini et al., 2016), *NIL* (Nakagawa et al., 2021)

[1]Available online: https://github.com/Glopix/Cloreco



New clone detector tool

Please ensure that your Clone Detector tool and repo meet our requirements:
https://github.com/Glopix/cloreco/wiki/Add-new-codeclone-detector-tools

Non-compliance may lead to failure in the integration process or during the execution of your tool. Your attention to these guidelines is crucial for a successful integration.
If you use the same name as an existing detector tool, this tool will be overwritten.

Clone detector tool name
this name will be displayed in the web form
My new and fancy clone detector

Repository URL
remote repository URL, where the clone detector tool is stored
https://github.com/acme/my-new-and-fancy-clone-detector

JDK Version
which Java Development Kit(JDK) version your tool needs
jdk11

Distribution
which Distribution to build on
Please note: not every software which is installed on the full distribution (e.g. ubuntu 22.04) is pre-installed in its docker image version. Make sure every dependency of your tool is installed via the 'install.sh' script, provided in your specified Git repository
ubuntu22.04

Installation directory
Your clone detector tool will be installed in this path.
Enter an absolute path.
Recommended: /cloneDetection/Applications/<tool name>/
e.g.: /cloneDetection/Applications/MyNewTool/
/cloneDetection/tools/my-new-and-fancy-clone-detector

Arguments for the newly added clone detector tool can be added in the next step.

Submit

Figure 2: Screenshot of *CloReCo*'s web interface.

- Managing benchmarking/tool configurations
- Managing and analyzing benchmarking results
- Registration of additional clone detectors
- Registration of additional benchmarks

With our contribution, we hope to amplify the development and evaluation of clone detectors and to facilitate the reproducibility of experimental benchmarking and performance analysis by researchers and practitioners (cf. (Collberg et al., 2014)).

2 THE CloReCo PLATFORM

CloReCo is based upon the *Docker container infrastructure*[2]. *Containers* implement lightweight, standalone, and executable software packages that include everything required to execute a certain piece of software, e.g., its code, dependencies and configuration settings. Containers work by isolating the software from the host system, where it is executed, ensuring that the software runs consistently regardless of where it is deployed. This isolation is put into effect through virtualization at the operating system level, such that a container shares the host operating system though operates in its own isolated user space. Container infrastructures support the easy creation, provisioning, distribution, and management of containers, allowing for packaging software conjoined with its dependencies into a single image that can be deployed everywhere. This way, portability and consistency is achieved.

We here advocate for the usage of containers to enhance reproducibility in benchmarking experiments for code clone detection. Containers can be used to encapsulate the required benchmarking environment, including clone detectors, benchmarking datasets, dependencies and configuration settings, ensuring that benchmarking experiments can be replicated across different systems. This consistency is crucial for evaluating and assessing clone detectors' performance metrics like precision, recall, runtime, or scalability. Note that encapsulation is not only

[2]https://www.docker.com/

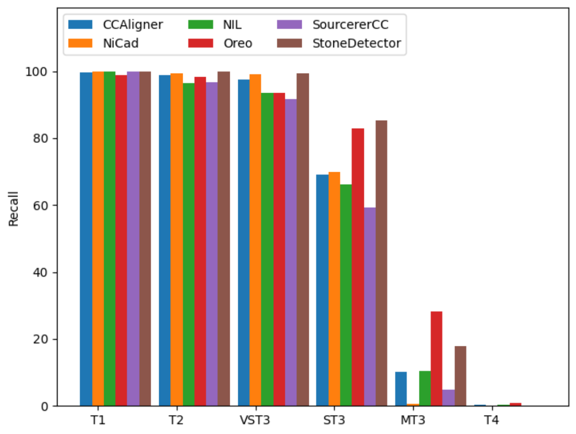


Figure 3: Sample performance results on benchmark *Big-CloneBench* (T1: Type 1, T2: Type 2, VST3: Very-Strongly Type 3, ST3: Strongly Type 3, MT3: Moderatly Type 3, T4: Type 4 code clones; cf. (Svajlenko and Roy, 2016)).

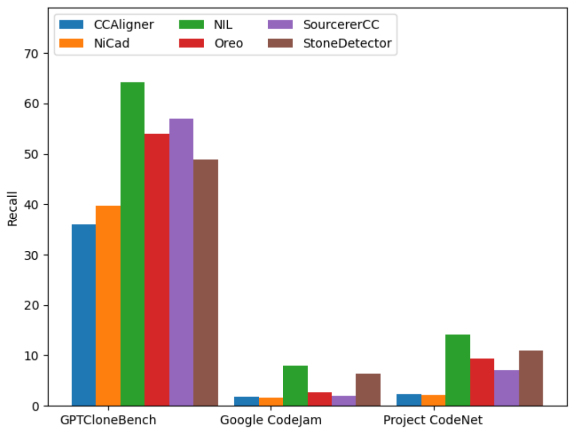


Figure 4: Sample performance results on other benchmarks.

important for the clone detectors, in order to harmonize their execution environments and configuration parameters, but also for the benchmarks, due to potentially required dependencies, evaluation scripts and settings. Based upon containers, researchers and practitioners can thus share their benchmarking environments, implying more coherent and reproducible results. Additionally, containers facilitate the automation of the benchmarking process, allowing for extensive analysis and validation with only limited or even without manual intervention. This approach not only streamlines the development workflow but also enhances the credibility and reliability of performance analysis in this area (Collberg et al., 2014).

For benchmarking experiments in clone detection, two components are basically needed: a clone detector and a benchmarking dataset. Both components are encapsulated by separate containers in the CloReCo infrastructure, as can also be seen in Fig. 2. As shown in the figure, a container for a certain clone detection tool is derived from a standard Ubuntu image and a base image (*detector-tool-base* in Fig.2). The latter provides the uniform environment for clone detection benchmarking, including Python and Java installations as well as several helper and configuration scripts for orchestrating and automating the benchmarking process. CloReCo already comes with predefined container images of some clone detectors (e.g., *sourcerer-cc* in Fig. 2), but also allows for the inclusion of other clone detectors or varying configurations of the same clone detector by means of additional containers. Besides the containers for clone detectors, the CloReCo infrastructure comprises benchmark containers, where each benchmark has its own container image (e.g., *gpt-clone-bench-benchmark* in Fig. 2). Again, besides the benchmarks predefined in CloReCo, further benchmarks may be included in terms of additional container images.

For conducting a benchmarking experiment, a number of tool and benchmark containers is selected, configured, launched, and integrated using a temporary volume (again compare with Fig. 2). Configuration is implemented based upon uniform configuration files for setting the tools' and benchmarks' parameters, e.g., applied similarity threshold or accepted minimum size of code fragments (Svajlenko and Roy, 2016). Due to the usually long-running nature of the benchmarking, the experiment's progress is logged and monitored in the CloReCo platform. After the experiment finishes, the experiment's results (logfiles, metrics like recall, precision, runtime, etc.) as well as its configuration settings become available for (visual) analysis and are also made persistent in the platform's database. The platform supports the interactive management of multiple experiments.

CloReCo offers both, a web application and a command line application as user interface. While the former assists researchers and practitioners in the easy usage of the platform, the latter enables the platform's headless operation. An example of CloReCo's web interface is shown in Fig. 2. The screenshot depicts the dialogue for registering a new clone detection tool to be added to the CloReCo platform. A container image will subsequently be generated and the tool becomes available for benchmarking experiments. As can be seen, adding a clone detector to the platform only requires a link to the tool's code repository, and an standardized runner script and configuration file.

We have conducted exhaustive performance analysis of clone detectors using CloReCo (Burock, 2024; Ostryanin, 2024). The results of two sample benchmarking experiments regarding the tools' recall are depicted in Fig. 3 and Fig. 4. In Fig. 3, the clone detectors' ability to find code clones is depicted for the six tools *CCAligner* (Wang et al., 2018), *NiCad* (Roy and Cordy, 2008), *NIL* (Nakagawa et al., 2021), *Oreo* (Saini et al., 2018), *SourcererCC* (Saini et al., 2016), and *StoneDetector* (Amme et al., 2021; Schäfer et al., 2020) on *BigCloneBench* (Svajlenko and Roy, 2015), thereby considering different levels of syntactical similarities, ranging from almost identical code duplicates where only the formatting differs (T1 code clones) to code fragments featuring more than 50% of syntactical deviations (T4 code clones). In Fig. 4, the six tools' recall on three other benchmarks, i.e., *Google CodeJam*, *Project CodeNet* (Puri et al., 2021), *GPTCloneBench* (Alam et al., 2023), are shown, complementing the picture.

3 RELATED WORK

Reproducibility of experiments on software has long been identified as an issue, reinforced by the seminal report (Collberg et al., 2014). There also has been substantial research on using container technology to support reproducibility in software research. Carl Boettiger, for instance, describes Docker as a tool for reproducible experiments, emphasizing its ability to address common reproducibility challenges through operating system virtualization and cross-platform portability (Boettiger, 2015). Several guidelines have been published offering best practices for using containers to ensure reproducibility (Cito and Gall, 2016). However, we are not aware of such comprehensive approach, combining multiple benchmarks and clone detectors for performance analysis in the area of code clone detection. Comparative evaluations in this area tend to either resort to a single benchmark, i.e., state-of-the-art *BigCloneBench* (Svajlenko and Roy, 2015) in case of Java, or provide more exhaustive evaluations which are however flawed by missing or insufficient replication data.

4 CONCLUSION

In this paper, we present the CloReCo benchmarking platform and advocate for the use of container technology for facilitating reproducibility of experimental benchmarking and performance analysis by researchers and practitioners in the area of code clone detection. Using CloReCo allows for managing and conducting benchmarking experiments combining multiple benchmarking datasets, clone detectors, and clone detector configurations. The CloReCo platform therefore offers an extensible integration approach based upon containers and provides a web as well as a command line interface. In future work, we hope to add more and more clone detectors as well as benchmarks to the CloReCo platform, besides the six clone detectors and four benchmarks mentioned in this paper, and to conduct evaluations on the resulting larger grounds. In particular, we are interested in the more comprehensive benchmarking of Deep learning approaches and tools for code clone detection. While Deep Learning clone detectors, e.g., Oreo, are already supported and can be integrated within the platform, we hope to provide features for more sophisticated performance analysis regarding aspects like consideration of bias in training/evaluation data, data imbalance, or reliability of ground truth (Schäfer et al., 2022; Sonnekalb et al., 2022).

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their many insightful comments and suggestions.

REFERENCES

Alam, A. I., Roy, P. R., Al-Omari, F., Roy, C. K., Roy, B., and Schneider, K. A. (2023). Gptclonebench: A comprehensive benchmark of semantic clones and cross-language clones using GPT-3 model and semanticclonebench. In *IEEE International Conference on Software Maintenance and Evolution, ICSME 2023, Bogotá, Colombia, October 1-6, 2023*, pages 1–13. IEEE.

Amme, W., Heinze, T. S., and Schäfer, A. (2021). You look so different: Finding structural clones and subclones in java source code. In *IEEE International Conference on Software Maintenance and Evolution, ICSME 2021, Luxembourg, September 27 - October 1, 2021*, pages 70–80. IEEE.

Boettiger, C. (2015). An introduction to docker for reproducible research. *ACM SIGOPS Oper. Syst. Rev.*, 49(1):71–79.

Burock, F. (2024). Plattform für klondetektoren-automatisierung. Master's thesis, Faculty of Mathematics and Computer Science, Friedrich Schiller University Jena.

Cito, J. and Gall, H. C. (2016). Using docker containers to improve reproducibility in software engineering research. In Dillon, L. K., Visser, W., and Williams, L. A., editors, *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume*, pages 906–907. ACM.

Collberg, C., Proebsting, T., Moraila, G., Shankaran, A., Shi, Z., and Warren, A. M. (2014). Measuring reproducibility in computer systems research. Technical report, Department of Computer Science, University of Arizona.

Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S., Tufano, M., Deng, S. K., Clement, C. B., Drain, D., Sundaresan, N., Yin, J., Jiang, D., and Zhou, M. (2021). Graphcodebert: Pre-training code representations with data flow. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Inoue, K. and Roy, C. K., editors (2021). *Code Clone Analysis*. Springer Singapore.

Krinke, J. and Ragkhitwetsagul, C. (2022). Bigclonebench considered harmful for machine learning. In *16th IEEE International Workshop on Software Clones, IWSC 2022, Limassol, Cyprus, October 2, 2022*, pages 1–7. IEEE.

Nakagawa, T., Higo, Y., and Kusumoto, S. (2021). NIL: large-scale detection of large-variance clones. In Spinellis, D., Gousios, G., Chechik, M., and Penta, M. D., editors, *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, pages 830–841. ACM.

Ostryanin, E. (2024). Evaluierung von klonerkennungsverfahren für java-quellcode unter verwendung von cloreco. Bachelor's thesis, Faculty of Mathematics and Computer Science, Friedrich Schiller University Jena.

Puri, R., Kung, D. S., Janssen, G., Zhang, W., Domeniconi, G., Zolotov, V., Dolby, J., Chen, J., Choudhury, M. R., Decker, L., Thost, V., Buratti, L., Pujar, S., Ramji, S., Finkler, U., Malaika, S., and Reiss, F. (2021). Codenet: A large-scale AI for code dataset for learning a diversity of coding tasks. In Vanschoren, J. and Yeung, S., editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Roy, C. K. and Cordy, J. R. (2008). NICAD: accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In Krikhaar, R. L., Lämmel, R., and Verhoef, C., editors, *The 16th IEEE International Conference on Program Comprehension, ICPC 2008, Amsterdam, The Netherlands, June 10-13, 2008*, pages 172–181. IEEE Computer Society.

Roy, C. K., Cordy, J. R., and Koschke, R. (2009). Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Sci. Comput. Program.*, 74(7):470–495.

Saini, V., Farmahinifarahani, F., Lu, Y., Baldi, P., and Lopes, C. V. (2018). Oreo: detection of clones in the twilight zone. In Leavens, G. T., Garcia, A., and Pasareanu, C. S., editors, *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, pages 354–365. ACM.

Saini, V., Sajnani, H., Kim, J., and Lopes, C. V. (2016). Sourcerercc and sourcerercc-i: tools to detect clones in batch mode and during software development. In Dillon, L. K., Visser, W., and Williams, L. A., editors, *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume*, pages 597–600. ACM.

Schäfer, A., Amme, W., and Heinze, T. S. (2020). Detection of similar functions through the use of dominator information. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS 2020, Companion Volume, Washington, DC, USA, August 17-21, 2020*, pages 206–211. IEEE.

Schäfer, A., Amme, W., and Heinze, T. S. (2022). Experiments on code clone detection and machine learning. In *16th IEEE International Workshop on Software Clones, IWSC 2022, Limassol, Cyprus, October 2, 2022*, pages 46–52. IEEE.

Sonnekalb, T., Gruner, B., Brust, C., and Mäder, P. (2022). Generalizability of code clone detection on codebert. In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*, pages 143:1–143:3. ACM.

Svajlenko, J. and Roy, C. K. (2015). Evaluating clone detection tools with bigclonebench. In Koschke, R., Krinke, J., and Robillard, M. P., editors, *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*, pages 131–140. IEEE Computer Society.

Svajlenko, J. and Roy, C. K. (2016). Bigcloneeval: A clone detection tool evaluation framework with bigclonebench. In *2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, Raleigh, NC, USA, October 2-7, 2016*, pages 596–600. IEEE Computer Society.

Svajlenko, J. and Roy, C. K. (2022). Bigclonebench: A retrospective and roadmap. In *16th IEEE International Workshop on Software Clones, IWSC 2022, Limassol, Cyprus, October 2, 2022*, pages 8–9. IEEE.

Wang, P., Svajlenko, J., Wu, Y., Xu, Y., and Roy, C. K. (2018). Ccaligner: a token based large-gap clone detector. In Chaudron, M., Crnkovic, I., Chechik, M., and Harman, M., editors, *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pages 1066–1077. ACM.

Wang, W., Li, G., Ma, B., Xia, X., and Jin, Z. (2020). Detecting code clones with graph neural network and flow-augmented abstract syntax tree. In Kontogiannis, K., Khomh, F., Chatzigeorgiou, A., Fokaefs, M., and Zhou, M., editors, *27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020, London, ON, Canada, February 18-21, 2020*, pages 261–271. IEEE.

Wei, H. and Li, M. (2017). Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code. In Sierra, C., editor, *Proceedings of the Twenty-Sixth*

International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017, pages 3034–3040. ijcai.org.