Analysis of Evolution of Recent Trends in Code Clone Detection Techniques from Antiquated Techniques

Nidhi Sehrawat, Kamna Solanki, Sandeep Dalal, Amita Dhankhar
Maharashi Dayanand University, Rohtak, India
Corresponding author: Nidhi Sehrawat, Email: sehrawatnidhi1994@gmail.com

The process used to detect and reveal bugs from software is known as software testing. This is a process as well as practice. This approach is different from software development. It can be recognized as an integral element of software development. Clone testing is a popular type of software testing. It is used during software development to check for duplication in software. This approach not only saves the time but the efforts of the software programmers. The copy and paste program generate the code over and over again. Therefore, a bug found in one unit recurs in each copy. This makes the removal of bugs and protection of the software complex. Code cloning complicates the protection of software. A variety of recent methods of code clone detection and their evolution from antiquated techniques has been studied in this paper.

Keywords: Code Clone, Machine Learning, Deep Learning, Tokenization, Type-II

2023. In Satyasai Jagannath Nanda & Rajendra Prasad Yadav (eds.), *Data Science and Intelligent Computing Techniques*, 911–922. Computing & Intelligent Systems, SCRS, India. https://doi.org/10.56155/978-81-955020-2-8-77

1 Introduction

Software engineers during software development choose for copying and pasting a piece of source code directly from other source code part, albeit with minor changes, with the goal that they are comparable or even. that seem indistinguishable. This is known as software/code cloning, sometimes called duplication of code. There are many reasons for cloning code. One of the most fundamental reasons is that code clone allows the developers to accomplish the tasks in less time. This kind of behavior causes several problems related to programming and maintenance. To illustrate, let a bug occurs in a cloned code piece of a software framework [1], the developer needs to distinguish it all over and tackle it, making the process to maintain the software more challenging. In addition, in the context of software system security, the cloning of a vulnerable code segment can prompt the proliferation of clone vulnerability. Despite the fact that software engineers prefer to write source code that is more secure and limit the sensitivity in the source code during software development, code clone behavior certainly finds throughout the software programming cycle and makes the system more vulnerable. As it is difficult to dispense with every one of the current clones, the recognition and the board of code clones are critical. Since not all existing clones can be eliminated, the task of detecting and managing the code clones is more significant.

1.1 Need of Code Cloning

There is the need of comprehensive and mature programming to develop a high- quality SE project for which systematic and mature programming is required in the development. Nevertheless, sometimes, programmers like to recycle a code segment for accomplishing their tasks, even though it is not advisable. Following are the main reasons due to which the code clone is occurred.

- Cost and Time Constraints: The programmers generally do code cloning to complete software development more competently [2]as it reduces cost as well as time, especially when tasks are meeting deadlines.
- Limitations of Programmers' Skills: Some programming tasks can be more difficult for some inexperienced, and even experienced programmers. For example, programmers may not be more proficient in a programming language or may find it difficult to understand functions that facilitate code reuse.
- Use of Templates: Gradually, code templates deliver a systematic and finest code, and structures for programmers to assist them complete software development in a more excellent manner. Nevertheless, programs using the similar template may contain matching or substantially identical code segments, which results in code clones.
- Fear to Bring in New Ideas: Occasionally, fresh plans or novel code leads to a longer SDLC, or more errors in existent software. Therefore, developers are afraid to introduce original notions or new code into their existent projects.
- Accidental Cloning: Sometimes, some part of code written by chance by a programmer matches existent code, causing a sort of accidental code cloning [3].

1.2 Issues in Code Cloning

While developing the software, not "re-inventing the wheel" is like two-fold approach. Clone cloning or reutilization of a code from other software projects is less time- consuming and does not require a large workforce. Whether intentionally or unintentionally, code clones have caused certain problems in software development and maintenance, as discussed below:

- Redundancy: Using code clones excessively for developing software does not follow the rules of the encapsulation. Redundant clones initiate uncontrolled copy-paste, which makes it difficult for later developers to move the project forward.

- Licensing and plagiarism: Most of the programmers focus on reusing the code from other sources via publicly available platforms without having the knowledge of the license terms and conditions of its use. The clear result is the severe breach of copyright rules.
- Reliability and security: To clone the code from unreliable third-party archives or bug-infected sources also presents great issues in analyzing and testing the software, as this type of cloned code needs more efforts to fix the bug, re- factor, and incorporate [4].
- Bugs propagation: When the code, having a bug, is cloned, the bug is transmitted to several settings in the software system. This supports the maintenance effort for recognizing this bug in all cloned code pieces.
- Vulnerability propagation: In case of copying of a fragment of code having susceptibility against certain assaults, the spread of vulnerabilities is occurred throughout the software system.

1.3 Code Clone Detection

It is a common belief that the presence of code clones increases the complexity of software maintenance. A code clone refers to a code piece which contains a similar code piece in the source code. There are many reasons for introducing code clones such as code reuse through 'copy and paste'. Modifying a code clone with several identical code pieces makes it essential to consider whether or not it is required to amend each of them. Especially, this process is very complex and costly for wide-ranging software. We sometimes ignore some code pieces that should be modified together [5]. To automatically detect code clones, various detection techniques are present. Researchers have identified CODE duplication as a potentially grave issue adversely affecting the functional reliability, comprehension and development of software frameworks. In recent years, the research group of software clone has devised a number of techniques for detecting and analyzing copied code. The researchers have also paid attention to clone management operations, for example the detection of clones in a project's history, analyzing the uniformity of changes to clones, sequentially updating clone groups as the project develops, and refactoring of clones based on priority. Detection of code clones is essential in multiple software engineering operations, for example refactoring, code discovery [6], reuse, and bug finding. Upon finding a defect in a code snippet, the inspection of all cloned code snippets is essential for the similar defect. Consequently, code clone errors can propagate, causing maintenance overhead to a high level. Therefore, code clone detection is of great importance in software engineering, and has been broadly studied.

1.4 Code Clone Detection

It is a common belief that the presence of code clones increases the complexity of software maintenance. A code clone refers to a code piece which contains a similar code piece in the source code. There are many reasons for introducing code clones such as code reuse through 'copy and paste'. Modifying a code clone with several identical code pieces makes it essential to consider whether or not it is required to amend each of them. Especially, this process is very complex and costly for wide-ranging software. We sometimes ignore some code pieces that should be modified together [5]. To automatically detect code clones, various detection techniques are present. Researchers have identified CODE duplication as a potentially grave issue adversely affecting the functional reliability, comprehension and development of software frameworks. In recent years, the research group of software clone has devised a number of techniques for detecting and analyzing copied code. The researchers have also paid attention to clone management operations, for example the detection of clones in a project's history, analyzing the uniformity of changes to clones, sequentially updating clone groups as the project develops, and refactoring of clones based on priority. Detection of code clones is essential in multiple software engineering operations, for example refactoring, code discovery [6], reuse, and bug finding. Upon finding a defect in a code snippet, the inspection of all cloned code snippets is essential for the similar defect. Consequently, code clone errors can propagate, causing

maintenance overhead to a high level. Therefore, code clone detection is of great importance in software engineering, and has been broadly studied.

1.5 Types of Code Clones

To generate more understanding about the study type related to code cloning, and analyzing the destination source code more fruitfully, the code clone issues are categorized into 2 categories called text-level and semantic-level clones. Both of these code clone types are discussed as follow:

1.5.1 Text-Level Clone

- Type-1: Exact clone: It represents a code segment which represents a correct duplicate of the real code segment, excluding whitespaces, blanks, and comments that are considered to be an accurate clone. Unlike the real code segment, such code segment only makes modification in the outline of the comments and removes a blank line, so that it looks like the original code fragment.
- Type-2: Renamed clone: It refers to a code segment having similarity to the real code segment, apart from the names of variables, categories, functions, and literals, are treated like a renamed clone [8].
- Type-3: Near miss clone: It is a code segment which is nearly identical to the real piece of code, excepting a few changes, for example included or eliminated statements, and a dissimilar usage of literals, variables, layouts, and comments is considered to be a near miss clone. This kind of segment is closely related to the miss clone form due to the changes which respectively provides 'a' and 'b' an alternate of 'string' and 'elem' variables.

1.5.2 Semantic -Level Clone

The code clones in this category depend on the semantic level, and is known as the Type-4 clone:

- Type-4: Semantic clone: It is a code segment that is identical to the real code segment depending on their functions and not syntax. Such a segment amends the code with the help of a 'for loop' for deploying the similar result obtained through the function 'count', that is considered to be a semantic clone [9].

1.6 Code Clone Detection Phases

Figure 1 provides a schematic representation of the complete process of detecting the code cloning. This process starts from preprocessing and ends with cloned clones reporting. The main element of frameworks of detecting the code clone is the code clone detector. Its main objective is to obtain copy-paste or duplicate source code and process the crucial steps of detection the cloned code.

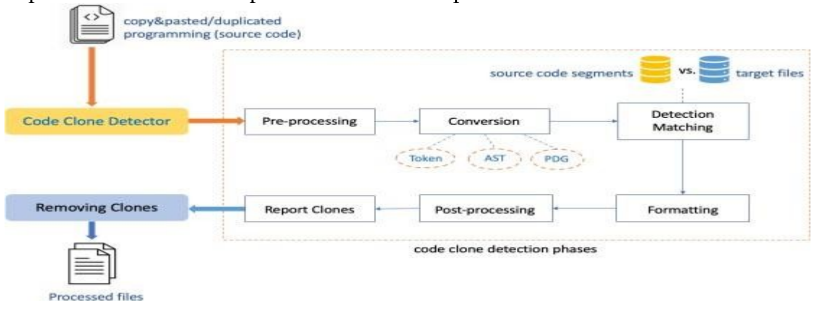

Figure 1. Code Clone Detection Process [5].

The steps involved in the life time of detecting the code clone are explained as below:

- Pre-Processing: Pre-processing code is the first step in clone detection removing all redundant or inappropriate fragments of the source code. This process aims at reducing irrelevant comparisons and computations, identifying the remaining the source code as source units to check the presence of direct clones' relationships [10] among one another once the unrelated portions are removed, and dividing the source parts into minor pieces based on the comparison algorithmic approach.
- Conversion: This step is also known as transformation. It converts the source code obtained from the initial stage into the respective intermediary depiction for additional comparison. Intermediate representation types are tokens, etc.
- Detection Matching: It identifies similar base code fragments by comparing the source code units with the destination files through a special comparison algorithmic approach. This step generates output in the form of a catalog of cloned classes [11].
- Formatting: It involves formatting the catalog of clone pairs achieved from the earlier stage depending on the comparison algorithmic approach into a fresh clone pair list corresponding to the real source code.
- Post-Processing: This stage, known as manual analysis, is an optional step in most of the code clone detection frameworks. This step filters out FPs or missed clones based on re-analysis which professional or automatic heuristics are conducted.
- Report Clones: This stage emphasizes on reporting the results examined and established through the earlier steps to the framework to take more actions, for example correcting or deleting the source code [12].

1.7 Code Clone Detection Phases

Clone detection methods are basically of four types which are explained as below:

- Textual approach: These methods need less normalization or alteration of code. This method typically performs line to line comparison. This comparison is based on two kinds of matching i.e., simple line matching and parameterized line matching. This approach is based on string principally.
- Lexical approach: In this technique, the source code is converted into tokens using lexical rules. Then, these tokens are compared [13].
- Syntactic approach: In this technique the theoretical tree is constructed. By means of parse code, the transformation of source code is done into a parse tree. Afterward, the theoretical tree is processed to discover clones through either tree matching or metrics.
- Semantic approach: This technique represents source code as a program dependency graph. The nodes depict statements and terminology. Edge is the representative of control and information dependency.

A clone identifier technique should attempt to find pieces of code of extreme resemblance in the source of a system or text. This technique does tell which code fragment will repeat again and this is the major issue of this approach. Hence, the detector or identifier should compare the promising code fragment with every second promising fragment in real manner. This kind of comparison is quite costly from calculation viewpoint. Therefore, a number of procedures are utilized to decrease the area of comparison before conducting real comparisons. Additional scrutiny and technique assistance may be needed for the identification of real clones even after the identification of potentially cloned fragments. In this chapter, the complete summary of fundamental stages in a clone discovery procedure is offered. This clone detection techniques can be compared and evaluated with respect to their fundamental systems for specific levels using this comprehensive approach.

2 Literature Review

2.1 Software Code Clone Detection using Deep Learning

K. Xu, et.al (2021) suggested an improved SCCD-GAN (semantic code clone detection based on Graph Attention Network) for computing the similarity of code pairs and achieving FPR (false positive rate) [14]. The control and data flow information were extended to original abstract syntax tree for developing the graph representation of the code. An attention mechanism was adopted in this model for extracting significant code portions and attributes so that higher precision was obtained. BigCloneBench2 and Google Code Jam datasets were applied to compute the suggested algorithm. The results depicted that the suggested algorithm outperformed the existing techniques, and led to increase the precision and mitigate the FPR.

D. Yuan, et.al (2022) devised a new GR (graph representation) technique on the basis of intermediate code for detecting the functional code clones [15]. The GE (graph embedding) methods were implemented for extracting the syntactic and semantic attributes. Thereafter, the Softmax classification algorithm was presented for detecting the functional code clone pairs. BigCloneBench dataset was considered in the experimentation while evaluating the devised technique. The experimental outcomes exhibited the supremacy of the devised technique over the traditional algorithms and enhanced the F1 score up to 33.4%.

Jie Zeng, et.al (2019) suggested a fast technique of detecting a code clone on the basis of weighted RAE (Recursive Auto Encoders) for evaluating the measure code similarity at the function level [16]. Initially, a weighted RAE was adopted for analyzing the program abstract syntax trees, to extract the program attributes and encode the functions to vectors. Subsequently, functions having similar vectors were reported as code clone pairs. However, this process consumed much time. This issue was resolved by converting the problem related to detect the clone into an ANNS in a high-dimensional vector set. The experimental results obtained on BCB (Big Clone Bench) depicted that the suggested technique was performed more effectively in comparison with other techniques. Moreover, this technique effectively detected the Type-3 or Type-4 clones, and offered FPR up to 0.05.

Yi Gao, et.al (2019) developed a tree embedding method in order to detect the clone [17]. First of all, a tree was embedded for attaining anode vector for every intermediate node in Abstract Syntax Tree that was able for capturing the structure information regarding ASTs. Afterward, a lightweight technique was implemented for generating a tree vector from its involving node vectors. In the end, the code clones were recognized after evaluating the Euclidean distances among tree vectors. The BCB (Big Clone Bench) dataset was applied to compute the developed method. The outcomes exhibited that the accuracy and recall acquired from the developed method were found superior over the other techniques.

Wenhan Wang, et.al (2020) constructed a graph representation of programs recognized as Flow-Augmented Abstract Syntax Tree algorithm [18]. The original ASTs were augmented with explicit control and data flow edges using FA-AST. Thereafter, this algorithm made the deployment of 2 GNNs (Graph Neural Networks) for quantifying the similarity of code pairs. The GNNs were adaptable in the domain of detecting the code clone. Google Code Jam and BCB (Big Clone Bench) datasets were utilized to implement the constructed algorithm. The results revealed the effectiveness of the constructed algorithm over the existing techniques.

Yuan Yuan, et.al (2019) presented a new technique for detecting the semantic level clone [19]. The traditional DTW (dynamic time warping) algorithm was incorporated with bi-directional RNN (Recurrent Neural Network) auto encoder and GCN (Graph Convolutional Network) for exploiting the CFG as an intermediate illustration of the coding technique with the purpose of detecting the semantic level clone from local to global. A dataset was utilized to conduct the experiments. The experimental outcomes demonstrated that the presented technique had generated the optimal results to detect the clone.

Guangjie Li, et.al (2020) intended a DL mechanism with the objective of detecting the code clones [20]. This mechanism was focused on mapping the syntactical features, whose extraction was done from code pairs, into predictions of clones. The results of simulation validated that the intended mechanism had potential to differentiate the clone pairs from non-clone at precision of 90%.

Table 1. Deep Learning based CCD

Author	Year	Technique Used	Findings	Limitations
K. Xu, et.al	2021	Improved SCCD-GAN (semantic code clone detection based on Graph Attention Network)	The results depicted that the suggested algorithm outperformed the existing techniques	This algorithm was unable to detect all kinds of clones.
D. Yuan, et.al	2022	GR (graph representation) technique	The experimental outcomes exhibited the supremacy of the devised technique over the traditional algorithms and enhanced the F1 score up to 33.4%.	This technique offered poor results in some scenarios.
Jie Zeng, et.al	2019	Weighted Recursive Auto encoders (RAE)	This technique effectively detected the Type-3 or Type-4 clones, and offered FPR up to 0.05.	The suggested technique was incapable of detecting all kinds of code clones.
Yi Gao, et.al	2019	Tree embedding technique	The outcomes exhibited that the accuracy and recall acquired from the developed method was found superior over the other techniques.	The precision was computed on randomly sampled set using this method which caused unintentional errors.
Wenhan Wang, et.al	2020	Flow- augmented abstract syntax tree (FA-AST)	The results revealed the effectiveness of the constructed algorithm over the existing techniques.	This algorithm had not captured syntactic and semantic features of source code with accuracy.
Yuan Yuan, et.al	2019	A novel semantic level technique	The experimental outcomes demonstrate d that the presented technique had generated optimal results to detect the clone.	This technique was inadequate to some programming language.
GuangjieLi, et.al	2020	Deep learning- based approach	The results of simulation validated that the intended mechanism had potential to differentiate the clone pairs from non-clone at precision of 90%.	The intended mechanism provided poor performance in case the data was collected from programming contest platforms

2.2 Software Code Clone Detection using Deep Learning

W. Zhu, et.al (2022) investigated a MSCCD (Multilingual Syntactic Code Clone Detector) to detect the code cloning in which a parser generator was exploited for creating a code block extractor for the target language [21]. This extractor was useful for extracting the semantic code blocks from a parse tree. In addition, the Type-3 clones were detected at several granularities using the investigated algorithm. There were twenty languages executed for quantifying the investigated algorithm on BigCloneEval with respect to execution time. The experimental outcomes revealed that the investigated algorithm performed well in comparison with other methods.

Abdullah M. Sheneamer, et.al (2021) projected a novel formal framework of similarity in which the similarity measures were integrated [22]. Distinct similarity measures and similarity scores were employed as attributes in ML (machine learning) for detecting the code clones. The similarity measure was computed for extracting the similarity score attributes which were illustrated as vectors. The results exhibited that the projected framework detected the clone more effectively as compared to other detectors. Furthermore, this framework offered the success rate of 99% to detect the cloned codes and accuracy up to 100% in the majority of cases.

Golam Mostaeen, et.al (2020) established an ML (Machine Learning) technique in order to automate the validation process [23]. At first, the code clones were taken from various clone detection tools to construct a training dataset for diverse subject systems. Thereafter, this technique was trained with the extraction of various attributes from those clones. The clones were validated using the trained algorithm without human inspection. The established technique was assisted in eliminating the FP (false positive) clones from the outcomes and computing the accuracy of any clone detectors for any given set of datasets in automatic way. The experimental results confirmed that the accuracy of the established technique was calculated 87.4% while classifying the clone. Abdullah M. Sheneamer, et.al (2021) projected a novel formal framework of similarity in which the similarity measures were integrated [24]. Distinct similarity measures and scores were employed as attributes in ML for detecting the code cloning. The similarity measure was computed for extracting the similarity score attributes which were illustrated as vectors. The results exhibited that the projected framework detected the clone more effectively as compared to other detectors. Furthermore, this framework offered the success rate of 99% to detect the cloned codes and accuracy up to 100% in the majority of cases.

Dongjin Yu, et.al (2019) introduced a method to detect the code clone [25]. This method focused on implementing the Smith–Waterman algorithm on the byte code alignment. The technique calls and instruction sequences regarding code blocks were employed for detecting the cloned code of two levels. The similarity of code fragments was computed with regard to the similarity of both the sequences which implied the introduced method was capable of detecting some semantic code clones. At last, this method was computed in the experimentation. The experimental outcomes depicted that the introduced method was more efficient in contrast to other techniques.

Hongfa Xue, et.al (2020) designed a new closed-loop known as Twin-Finder with the purpose of detecting the pointer-related code clone [26]. In this approach, the ML (Machine Learning) was put together with the symbolic execution methods for attaining precision. A clone verification system was presented for determining whether 2 clone samples were indeed clones. A feedback loop was utilized for enhancing the ML model and further mitigating the FPs. The outcomes of experiment indicated that the designed approach was adaptable for recognizing more code clones as compared to other detector and eliminating an average FP up to 91.69%.

Table 2. Machine Learning based CCD.

Author	Year	Technique Used	Findings	Limitations
W. Zhu, et.al	2022	MSCCD (Multilingual Syntactic Code Clone Detector)	The experimental outcomes revealed that the investigated algorithm performed well in comparison with other methods.	This algorithm was inapplicable on real time datasets.
Abdullah M. Sheneamer, et.al	2021	novel formal framework	The results exhibited that the projected framework detected the clone more effectively as Compared to other detectors. Furthermore, this framework offered the success rate of 99% to detect the cloned codes and accuracy up to 100% in the majority of cases.	The recall was mitigated and the execution time was maximized in case of presence of Type-2 clones.

Golam Mostaeen, et.al	2020	Machine learning approach	The experimental results confirmed that the accuracy of the established technique was calculated 87.4% while classifying the clone.	The error rate of this technique was broadcasted in the process of computing the attributes.
Abdullah M. Sheneamer, et.al	2021	A new formal model	Furthermore, this framework offered the success rate of 99% to detect the cloned codes and accuracy up to 100% in the majority of cases.	This framework was not detected theType-3 or Type-4 clones.
Dongjin Yu, et.al	2019	Smith–Waterman algorith m	The experimental outcomes depicted that the introduced method was more efficient in contrast to other techniques	The introduced method was not adaptable for the real-world software development scenario.
HongfaXue, et.al	2020	Twin- Finder	The outcomes of experiment indicated that the designed approach was adaptable for recognizing more code clones as compared to other detectors and eliminating an average FP up to 91.69%.	The designed approach was Ineffective of eliminating 100 % of FPs (false positives).

2.3 Clone Software Code Clone Detection using Different Mechanisms

Shogo Tokui, et.al (2020) suggested a system called Clone Notifier that was alerted on conceptions and changes of code clones to the programmers of software [27]. Initially, this system was adopted for recognizing the conceptions and modifications of code clones. Afterward, this system had potential for grouping them into 4 kinds and assigning the labels: consistent and inconsistent. The results revealed that the suggested system was effective for detecting these kinds of clones.

KyoheiUemura, et.al (2019) formulated an integrated approach in order to detect the code clones and track their histories with the help of Historage [28]. An augmented form of the Git repository was acquired from the Historage in which pre-analyzed syntactic information was comprised. The code clones of method-level were taken in account in this approach and a technique of Git was useful to detect and track the code clones. Ten publicly available projects were considered to track and analyze the method-level code clones. The results indicated that the formulated approach was effective for eliminating the code clones of method-level without considering their changes or the time-interval of their change and generating a collection of method-level code clones simultaneously to prolong the survival as compared to those created at an individual level.

Davide Pizzolotto, et.al (2020) investigated a tool named Blanker that was implemented to search and to unify the comparable statements of the language prior to deploy the source into the traditional code clone detector which was capable of detecting type-2 clones [29]. This stage was considered as a normalization step and led to generate the re-factorable results without any error occurred due to the potentially unrelated added, and with added flexibility in comparison with the checking for identical code portions. NiCad was employed for detecting the clones when the normalization stage was started and ended. The investigated tool had detected the10% more type-2 clones subsequent this stage. Moreover, no limitation, no FPs (false positives) and FNs (false negatives) were obtained intype-2 report.

Benjamin Bowman, et.al (2020) recommended a method known as VGRAPH in order to recognize the vulnerable code clones, so that the robustness was provided to modify the code [30]. Three graph-based components, having potential to detect the customized code clones having susceptibility, and differentiate them from the patched ones, were implemented to construct a matching algorithm. This process had attained the precision of 98% and recall around 97%. This method became effective of recognizing ten vulnerabilities after its implementation on numerous versions of software packages.

The results proved the efficacy of the recommended method for detecting more vulnerable code clones and obtained lower FPs (false positives).

Zhipeng Gao, et.al (2019) developed Smart Embed, which was a web service tool assisted the Solidity developers in discovering the repetitive contract code and clone related bugs in smart contracts [31]. This system was planned on the basis of code embeddings and similarity checking methods. The similarities were compared among the code embedding vectors so that the cloned codes and bugs were detected for any code given via users. Consequently, the confidence of user was enhanced in the reliability of their code. The developed system offered the clone ratio of solidity code nearer to 90.01% which was found superior to the conventional software and precision up to 96%.

Table 3. CCD based on Different Mechanisms.

Author	Year	Technique Used	Findings	Results
Golam Mostaeen, et.al	2020	Machine learning approach	The experimental results confirmed that the accuracy of the established technique was calculated 87.4% while classifying the clone.	The error rate of this technique was broadcasted in the process of computing the attributes.
Abdullah M. Sheneamer, et.al	2021	A new formal model	Furthermore, this framework offered the success rate of 99% to detect the cloned codes and accuracy up to 100% in the majority of cases.	This framework was not detected theType-3 or Type-4 clones.
Dongjin Yu, et.al	2019	Smith–Waterman algorithm	The experimental outcomes depicted that the introduced method was more efficient in contrast to other techniques.	The introduced method was not adaptable for the real- world software development scenario.
HongfaXue, et.al	2020	Twin- Finder	The outcomes of experiment indicated that the designed approach was adaptable for recognizing more code clones as compared to another detector and eliminating an average FP up to 91.69%.	The designed approach was Ineffective of eliminating 100% of FPs (false positives).

This section presents the state-of-the-art review layout, a step-by-step method for the literature discussed. This research focuses on categorizing the current literature on code clone detection assessing the current trends. This evaluation finds relevant research articles from reputable electronic databases and the top conferences in the field. After then, inclusion and exclusion criteria were used to reduce the number of papers that were considered. Following that, final research studies were chosen based on a variety of variables. The information given here is the product of a thorough investigation. For this review study, various electronic database sources were investigated; some of the popular electronic databases used in this search like google scholar, Elsevier, Science direct etc. Using the inclusion criterion, which mainly depends on the techniques, the relevant work of code clone detection algorithms is retrieved from the enormous collection of data given by search engines. The data shows that journals account for most of the work in this study (51%), with conferences accounting for 40% of the work and book chapters accounting for 9%. In addition, the data depicts a year-by-year study of work relevant to code clone detection.

3 Conclusion

After removing this defect, this approach can be used to serve many purposes. The type 4 code clone is integrated with PM to remove the complexity of the function. Entire functions are cloned in type 4 but in conjunction with the PM algorithm, just a few lines are cloned. This in turn saves time and speeds up the process as compared to the earlier task. The code is an indication of the clone type complexity and the intensity of the complexity involved in detecting and identifying clones. In conclusion, Main material is issued in popular journals and Type 4 code clone detection schemes will be developed in the time ahead.

References

[1] A. Sheneamer and J. Kalita, “A Survey of Software Clone Detection Techniques”, International Journal of Computer Applications, vol. 12, no. 8, pp. 692-698, 2016

[2] B. van Bladel and S. Demeyer, "A Novel Approach for Detecting Type-IV Clones in Test Code," 2019 IEEE 13th International Workshop on Software Clones (IWSC), 2019, pp. 8-12

[3] N. Gupta, V. Gandhi, C. Hariya and V. Shelke, "Detection of Code Clones," 2018 International Conference on Smart City and Emerging Technology (ICSCET), 2018, pp. 1

[4] Y. Yuki, Y. Higo and S. Kusumoto, "A technique to detect multi-grained code clones," 2017 IEEE 11th International Workshop on Software Clones (IWSC), 2017, pp. 1-7

[5] Roopam and G. Singh, "To enhance the code clone detection algorithm by using hybrid approach for detection of code clones," 2017 International Conference on Intelligent Computing and Control Systems (ICICCS), 2017, pp. 192-198,

[6] T. Zhang and M. Kim, "Automated Transplantation and Differential Testing for Clones," 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), 2017, pp. 665-676

[7] C. M. Kamalpriya and P. Singh, "Enhancing program dependency graph based clone detection using approximate subgraph matching," 2017 IEEE 11th International Workshop on Software Clones (IWSC), 2017, pp. 1-7

[8] Y. Sabi, Y. Higo and S. Kusumoto, "Rearranging the order of program statements for code clone detection," 2017 IEEE 11th International Workshop on Software Clones (IWSC), 2017, pp. 1-7

[9] C. Wijesiriwardana and P. Wimalaratne, “Component-based experimental testbed to facilitate code clone detection research”, 2017, 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), 2017, 165 – 168

[10] G. Li, Y. Tang, X. Zhang and B. Yi, "A Deep Learning Based Approach to Detect Code Clones," 2020 International Conference on Intelligent Computing and Human-Computer Interaction (ICHCI), 2020, pp. 337-340

[11] H. Yu, W. Lam, L. Chen, G. Li, T. Xie and Q. Wang, "Neural Detection of Semantic Code Clones Via Tree-Based Convolution," 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), 2019, pp. 70-80

[12] D. Yu, J. Yang, X. Chen and J. Chen, "Detecting Java Code Clones Based on Bytecode Sequence Alignment," in IEEE Access, vol. 7, pp. 22421-22433, 2019

[13] Y. -L. Hung and S. Takada, "CPPCD: A Token-Based Approach to Detecting Potential Clones," 2020 IEEE 14th International Workshop on Software Clones (IWSC), 2020, pp. 26-32

[14] K. Xu and Y. Liu, "SCCD-GAN: An Enhanced Semantic Code Clone Detection Model Using GAN," 2021 IEEE 4th International Conference on Electronics and Communication Engineering (ICECE), 2021, pp. 16-22

[15] D. Yuan, S. Fang, T. Zhang, Z. Xu and X. Luo, "Java Code Clone Detection by Exploiting Semantic and Syntax Information From Intermediate Code-Based Graph," in IEEE Transactions on Reliability, vol. 1, pp. 571-579, 2022

[16] J. Zeng, K. Ben, X. Li and X. Zhang, "Fast Code Clone Detection Based on Weighted Recursive Autoencoders," in IEEE Access, vol. 7, pp. 125062-125078, 2019

[17] Y. Gao, Z. Wang, S. Liu, L. Yang, W. Sang and Y. Cai, "TECCD: A Tree Embedding Approach for Code Clone Detection," 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2019, pp. 145-156

[18] W. Wang, G. Li, B. Ma, X. Xia and Z. Jin, "Detecting Code Clones with Graph Neural Network and Flow-Augmented Abstract Syntax Tree," 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2020, pp. 261-271

[19] Y. Yuan, W. Kong, G. Hou, Y. Hu, M. Watanabe and A. Fukuda, "From Local to Global Semantic Clone Detection," 2019 6th International Conference on Dependable Systems and Their Applications (DSA), 2020, pp. 13-24

[20] G. Li, Y. Tang, X. Zhang and B. Yi, "A Deep Learning Based Approach to Detect Code Clones," 2020 International Conference on Intelligent Computing and Human-Computer Interaction (ICHCI), 2020, pp. 337-340

[21] W. Zhu, N. Yoshida, T. Kamiya, E. Choi and H. Takada, "MSCCD: Grammar Pluggable Clone Detection Based on ANTLR Parser Generation," 2022 IEEE/ACM 30th International Conference on Program Comprehension (ICPC), 2022, pp. 460-470

[22] A. M. Sheneamer, "Multiple Similarity-based Features Blending for Detecting Code Clonesusing Consensus-Driven Classification", Expert Systems with Applications, vol. 1, no. 13, pp. 4547- 4551, 2021

[23] G. Mostaeen, B. Roy and J. Svajlenko, "A machine learning based framework for code clone validation", Journal of Systems and Software, vol. 45, pp. 699-705, 2020

[24] A. M. Sheneamer, "Multiple Similarity-based Features Blending for Detecting Code Clones using Consensus-Driven Classification", Expert Systems with Applications, vol. 1, no. 16, pp. 561- 569, 2021

[25] D. Yu, J. Yang, X. Chen and J. Chen, "Detecting Java Code Clones Based on Bytecode Sequence Alignment," in IEEE Access, vol. 7, pp. 22421-22433, 2019

[26] H. Xue, Y. Mei, K. Gogineni, G. Venkataramani and T. Lan, "Twin-Finder: Integrated Reasoning Engine for Pointer-Related Code Clone Detection," 2020 IEEE 14th International Workshop on Software Clones (IWSC), 2020, pp. 1-7

[27] S. Tokui, N. Yoshida, E. Choi and K. Inoue, "Clone Notifier: Developing and Improving the System to Notify Changes of Code Clones," 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2020, pp. 642-646

[28] K. Uemura, A. Mori, E. Choi and H. Iida, "Tracking Method-Level Clones and a Case Study," 2019 IEEE 13th International Workshop on Software Clones (IWSC), 2019, pp. 27-33

[29] D. Pizzolotto and K. Inoue, "Blanker: A Refactor-Oriented Cloned Source Code Normalizer," 2020 IEEE 14th International Workshop on Software Clones (IWSC), 2020, pp. 22-25

[30] B. Bowman and H. H. Huang, "VGRAPH: A Robust Vulnerable Code Clone Detection System Using Code Property Triplets," 2020 IEEE European Symposium on Security and Privacy (EuroS&P), 2020, pp. 53-69

[31] Z. Gao, V. Jayasundara, L. Jiang, X. Xia, D. Lo and J. Grundy, "SmartEmbed: A Tool for Clone and Bug Detection in Smart Contracts through Structural Code Embedding," 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2019, pp. 394-397