Received April 5, 2019, accepted May 3, 2019, date of publication May 22, 2019, date of current version July 16, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2918202

A Systematic Review on Code Clone Detection

QURAT UL AIN, WASI HAIDER BUTT, MUHAMMAD WASEEM ANWAR, FAROOQUE AZAM, AND BILAL MAQBOOL

Department of Computer and Software Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology (NUST), Islamabad 44000, Pakistan

Corresponding author: Muhammad Waseem Anwar (waseemanwar@ceme.nust.edu.pk)

ABSTRACT Code cloning refers to the duplication of source code. It is the most common way of reusing source code in software development. If a bug is identified in one segment of code, all the similar segments need to be checked for the same bug. Consequently, this cloning process may lead to bug propagation that significantly affects the maintenance cost. By considering this problem, code clone detection (CCD) appears as an active area of research. Consequently, there is a strong need to investigate the latest techniques, trends, and tools in the domain of CCD. Therefore, in this paper, we comprehensively inspect the latest tools and techniques utilized for the detection of code clones. Particularly, a systematic literature review (SLR) is performed to select and investigate 54 studies pertaining to CCD. Consequently, six categories are defined to incorporate the selected studies as per relevance, i.e., textual approaches (12), lexical approaches (8), tree-based approaches (3), metric-based approaches (7), semantic approaches (7), and hybrid approaches (17). We identified and analyzed 26 CCD tools, i.e., 13 existing and 13 proposed/developed. Moreover, 62 open-source subject systems whose source code is utilized for the CCD are presented. It is concluded that there exist several studies to detect type1, type2, type3, and type4 clones individually. However, there is a need to develop novel approaches with complete tool support in order to detect all four types of clones collectively. Furthermore, it is also required to introduce more approaches to simplify the development of a program dependency graph (PDG) while dealing with the detection of the type4 clones.

INDEX TERMS CCD, SLR, code clone detection, CCD tools, code clone types.

I. INTRODUCTION

Code duplication by copying and pasting without or minor modification into another section of code frequently occurs in software development. This copied code is called code clone and the process is called code cloning. If an error is identified in one part of the code, correction is required in all the replicated segments. Therefore, it is essential to identify all related segments throughout the source code. Various studies suggested that almost 20-50 percent of large software systems consist of cloned code [1], [2].

There is no appropriate definition of code clone. Different researchers used different terms for cloning. Krinke [3] utilized the term "similar code". Baxter *et al.* [4] suggested that a clone is "a code segment that is identical to another segment". Ducasse *et al.* [2] utilized the term "duplicated code". Komondoor and Horwitz [5] also used "duplicated code" and clone as an item of duplicated code. Basic types of clones are listed below [6]:

The associate editor coordinating the review of this manuscript and approving it for publication was Michael Lyu.

Exact clones (Type 1): Identical code segments except for changes in comments, layouts and whitespaces.

Renamed clones (Type 2): Code segments which are syntactically or structurally similar other than changes in comments, identifiers, types, literals, and layouts. These clones are also called parameterized clones.

Near Miss clones (Type 3): Copied pieces with further modification such as addition or removal of statements and changes in whitespaces, identifiers, layouts, comments, and types but outcomes are similar. These clones are also known as gapped clones.

Semantic clones (Type 4): More than one code segments that are functionally similar but implemented by different syntactic variants.

Although there are four types of clones, sometimes people use different terms when referring to the clone relation to their experiments. Common terms utilized by them are given below.

Structural clones: Simple clones that follow the syntactic structure of a particular language within the syntactic

2169-3536 © 2019 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

boundary. These boundaries can be statement boundary, structure boundary, class boundary etc. A structural clone can be any of the four types of clones depending on its similarity level.

Function clones: These clones are simple clones that are limited to the procedure or method/function level granularity. Similar to the structural clones these clones can also be any of the four types of clones based on their level of similarity.

Cloning is beneficial but it can also be harmful in many ways. For example, in many software engineering tasks such as aspect mining, program understanding, plagiarism detection, copyright infringement investigation, code compaction, software evolution analysis, code quality analysis, bug detection and virus detection may need the extraction of semantically or syntactically similar code blocks, making clone detection effective and useful part of software analysis [7]. They can also lead to the bug propagation that significantly increases the software maintenance cost. By considering these maintenance problems, software clone detection appeared as an active area of research. Several approaches and tools introduced so far, for the detection of code clones and there have been many comparisons and evaluations studies. Text-based approaches, Token-based approaches, Tree-based approach, Metric based, Semantic approaches and Hybrid approaches are mainly used [7], [8]. Similarly, several tools are available for CCD like NICAD [9], [10], CCFinderX [11], [12], Simian [13], CPMiner [14] etc. Furthermore, certain similarity measure algorithms such as Fingerprinting [15], [16], Neural Networks [17], Euclidian Distance [18] etc. are also utilized for the detection of code clones. However, this area of research is quite intensive. Therefore, there is a strong need for the intensive evaluation of modern CCD techniques and tools in the existing state-of-the-art studies. In this regard, this article performs a Systematic Literature Review (SLR) in the area of CCD. According to Brereton *et al.* [19], Budgen and Brereton [20] and Charters and Kitchenham [21], a Systematic Literature Review (SLR) is a process of finding, evaluating and interpreting all available research related to a particular topic, research question, area or phenomenon of interest.

It is important to note that there exist two survey articles [6], [22] , published in 2007, in the area of CCD. Furthermore, a SLR is also performed in 2013 in the domain of CCD [23] . Therefore, in this article, we conduct SLR from 2013 onward and comprehensively examine the latest tools and techniques for CCD. We define 5 research questions for the SLR as follows:

RQ1: What are the popular approaches utilized for CCD and which type of clones they can detect?

RQ2: What are the popular tools used/developed for CCD?

RQ3: What are the open source subject systems or benchmark datasets commonly used for CCD?

RQ4: What are the advantages and limitations of CCD techniques and tools?

*RQ5:*What are the key improvements required in the CCD tools / techniques in order to meet the modern technological advancements?

To answer the above-mentioned research questions, we performed SLR on 54 research articles which include both journal papers published during 2013-2018 and conference papers published during 2015-2018. The contributions of this study are summarized below:

- Firstly, this article selects 54 studies (i.e. 2013-2018) pertaining to CCD by utilizing the standard guidelines of SLR [21].
- Secondly, this article analyzes six main CCD techniques and 26 CCD tools. This analysis surely assists the researchers while selecting the right tool or technique for CCD.
- Lastly, this article highlights the important gaps where improvements are needed in CCD.

To perform this SLR, a review protocol is implemented in **(Sec. II)**. Firstly, six categories are defined **(Sec. II(A))** for simplification of data extraction and synthesis process. We selected four scientific databases (IEEE, ACM, Springer and Elsevier) for search process **(Sec. III(C))** as defined in selection and rejection rules **(Sec. II(B))**. Accordingly, we selected 54 research studies that have full conformity with our selection and rejection rules. After that, we ensure that these studies give reliable outputs in quality assessment **(Sec. II(D))**. For the extraction and analysis of selected studies, we define a template **(Sec. II(E))**. Six categories are defined to incorporate the selected studies **(Sec.III)**. Afterwards, these categories are comprehensibly examined namely Textual **(Sec. III (A))**, Lexical **(Sec. III(B))**, Tree-based **(Sec. III(C))**, Metric based **(Sec. III(D))**, Semantic **(Sec. III(E))** and Hybrid **(Sec. III(F))**. Various CCD tools are identified and examined in **(Sec. III(G))**. Furthermore, different open source software systems are investigated in **(Sec. III(H))**. The detailed analysis of these studies provide the answers to research questions **(Sec.IV)**. Discussion and limitations provided in **(Sec.V)**. Lastly, conclusion is provided in **(Sec. VI)**.

II. REVIEW PROTOCOL

Review protocol consists of seven elements. Two elements such as background and research questions are discussed earlier in the introduction. Therefore, in this section, the details of these elements are excluded. The detail of the remaining five elements is provided in the subsequent sections. The overview of SLR is shown in Figure 1.

A. CATEGORIES DEFINITION

For the simplification of data extraction and synthesis process, we define six categories. The description of these categories is given below.

1) TEXTUAL APPROACHES

Code clones can be detected by using different CCD approaches. One of them is textual approaches, they

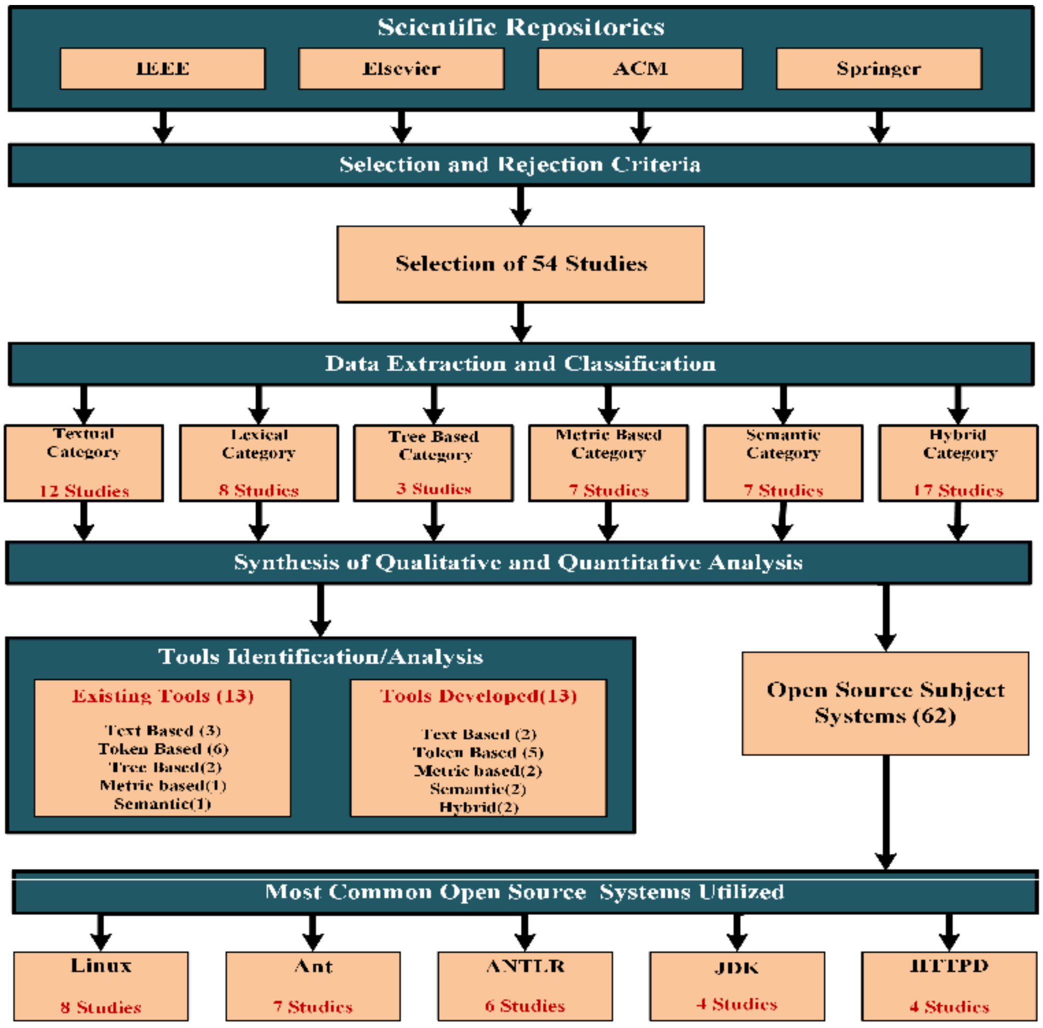
Scientific Repositories
IEEE
Elsevier
ACM
Springer
Selection and Rejection Criteria
Selection of 54 Studies
Data Extraction and Classification
Textual Category 12 Studies
Lexical Category 8 Studies
Tree Based Category 3 Studies
Metric Based Category 7 Studies
Semantic Category 7 Studies
Hybrid Category 17 Studies
Synthesis of Qualitative and Quantitative Analysis
Tools Identification/Analysis
Existing Tools (13)
Text Based (3)
Token Based (6)
Tree Based(2)
Metric based(1)
Semantic(1)
Tools Developed(13)
Text Based (2)
Token Based (5)
Metric based(2)
Semantic(2)
Hybrid(2)
Open Source Subject Systems (62)
Most Common Open Source Systems Utilized
Linux 8 Studies
Ant 7 Studies
ANTLR 6 Studies
JDK 4 Studies
HTTPD 4 Studies

FIGURE 1. Overview of systematic literature review.

detect type1 clones more effectively [64]. However, these approaches can also identify type2 and type3 clones [8]. Therefore, this category consists of research studies that particularly deals with CCD using textual approaches.

2) LEXICAL APPROACHES

The research studies that particularly deal with CCD based on lexical approaches are placed under this category. Lexical approaches are also known as token-based approaches and able to identify type 2 clones efficiently [64]. However, they can also uncover type1 and type3 clones [8].

3) TREE-BASED APPROACHES

This category consists of the studies that are dealing with CCD using tree-based approaches. They are most effective for the detection of type3 clones [64]. However, they have the ability to detect type1, type2, and type4 clones [8].

4) METRIC BASED APPROACHES

The research studies in which code clones are detected by utilizing metric-based approaches are placed under this category. They can detect type3 clones effectively [64]. However, they can also uncover type1, type2 and type4 clones.

5) SEMANTIC APPROACHES

The research studies that particularly deal with CCD based on semantic approaches are placed under this category. A semantic approach similar to tree-based and metric-based approaches have the ability to detect type1, type2, type3 and type4 clones. They are mainly used to uncover semantic or type 4 clones [64].

6) HYBRID APPROACHES

The research studies in which a combination of two or more aforementioned techniques e.g. (Textual, Lexical, Syntactic or Semantic) is utilized should be placed under this category.

B. SELECTION AND REJECTION CRITERIA

We define selection and rejection criteria to carry out this SLR for obtaining desired goals. For this purpose, some rules are defined as discussed below.

TABLE 1. Summary of search terms with results.

Sr.#	Search terms	Operator	IEEE	ACM	Elsevier	Springer
1	CCD	N/A	201	1385	25375	20134
2	CCD, Text-based Techniques	AND	5	88	161	113
		OR	2933	5274	15780	20245
3	CCD, Token-based techniques	AND	4	864	174	103
		OR	378	5155	3610	2377
4	CCD, Tree-based techniques	AND	8	877	2327	1748
		OR	3542	5228	13214	18331
5	CCD, Metric based techniques	AND	14	878	1018	665
		OR	7879	5245	3571	8987
6	CCD, PDG based techniques	AND	4	63	39	23
		OR	204	5154	9024	8759
7	CCD, Hybrid techniques	AND	8	38	19	79
		OR	1745	1847	1789	3358
8	CCD Tools	AND	111	325	413	389
		OR	5438	2646	9734	6559
9	CCD, Machine learning techniques	AND	10	539	818	564
		OR	4400	3734	6132	6377

- The research studies in which keyword "code clone detection" is included in the title or abstract are selected. We discard such research studies where code clone detection (CCD) is partially or not discussed.
- We consider the conference papers that are published from 2015 to 2018 and journals published from 2013 to 2018. All those research studies published before 2013 are rejected to assure the inclusion of the latest research studies.
- To perform this SLR, four well-known scientific databases (i.e. Springer, IEEE, Elsevier and ACM) are selected. Therefore, the research studies that are published in one of the above-mentioned databases are considered. Studies other than these repositories are not selected.
- Selected studies must be result oriented. Some solid evidence and experimentation must support the proposed methodologies and their ultimate outcomes.
- The research papers that have almost similar contents are discarded and only one of them is selected.

C. SEARCH PROCESS

The search process is started by utilizing four databases (IEEE, ACM, Springer and Elsevier) as described in selection and rejection rules. We have utilized many search terms or keywords while performing the search process. The overall summary of the search process isgiven in Table 1. To carry out the research process, two types of operators such as AND, OR are utilized. The outcomes collected from AND operator are not enough that is why OR operator is used. However, the results obtained by using the OR operator are very large, therefore, it is not feasible to scan all of these results. Therefore, advanced search options are utilized, provided by selected databases e.g. where keyword contain "time span" in order to get precise results. After the investigation of primary results, we select only those studies that are highly related to CCD. Finally, we get 54 research studies by following certain steps as shown in Figure 2**.**

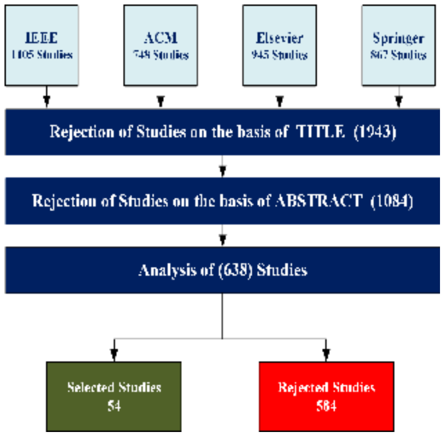
IEEE
1105 Studies
ACM
748 Studies
Elsevier
945 Studies
Springer
867 Studies
Rejection of Studies on the basis of TITLE (1943)
Rejection of Studies on the basis of ABSTRACT (1084)
Analysis of (638) Studies
Selected Studies
54
Rejected Studies
584

FIGURE 2. Summary of the search process.

- We overall consider 3665 research papers and reject 1943 by reading their title.
- Afterword we consider remaining 1722 papers and reject 1084 by examining their abstract.
- We investigate the remaining 638 research studies. Based on this investigation, we exclude 584 research studies and include 54 studies, which are completely meeting our defined selection and rejection criteria.

D. QUALITY ASSESSMENT

To assure the reliable results of this SLR, we have selected high impact studies e.g. researches from repositories that are authentic and accepted all over the world. Thirty-seven (37) research studies are selected from IEEE, seven (7) studies from Elsevier, five (5) studies from ACM and five (5) studies from Springer. The results presented in Table 2 indicate that

TABLE 2. Summary of selected studies according to scientific databases and publication type.

Database	Type	Reference	Total
IEEE	Conference	[24][26][27][28][29][30][33][34][35][40][41][42][43][44][46] [77][47][49][51][52][54][55][58][59][60][61][62][63] [64][65][66][71][72][74][75]	37
	Journal	[45][56]	
ACM	Conference	[31][39][48][50][68]	5
	Journal	Nil	
Elsevier	Conference	[53][73]	7
	Journal	[25][36][37][38][67]	
Springer	Conference	[57][76]	5
	Journal	[32][69][70]	

TABLE 3. Data extraction and synthesis template.

Sr.#	Description	Details
1	Bibliographic Information	The title, publication year and type of research paper (i.e. Conference or Journal) is observed.
2	Proposed methodology	The methodology followed by each study is analyzed.
3	Implementation details	Technologies used to implement the proposed methodology are analyzed.
4	Outcomes	Outcomes of each selected study are thoroughly analyzed.
5	Grouping	Selected studies are placed in to corresponding categories (Section II-A). The results are summarized in TABLE 4.
6	Investigation of categories	Analysis of each category to find the answers to the RQ's. The results are summarized below: Textual Approaches TABLE 5, Lexical Approaches TABLE 6, Tree-Based Approaches TABLE 7, Metric Based Approaches TABLE 8, Semantic Approaches TABLE 9 and Hybrid Approaches Table 10
7	Tools	Tools used and developed in the selected studies are observed in TABLE 11 and TABLE 12 respectively.
8	Open Source Subject Systems	Source code of various Open Source Subject utilized in selected studies are examined in TABLE *13*.

we try our best to choose the high impact and the latest research studies. The overall summary of the repositories w.r.t their publication type is given in Table 2. **Database** represents the names of the repositories. **Type** represents that whether the selected research study belongs to either journal or conference. **References** are given for selected studies. **Total** represents the number of total conference or journal papers of every scientific repository.

In Table 2, it can be observed that 35 conference papers and 2 journal papers selected from IEEE, 5 conference papers selected from ACM, 2 conference papers and 5 journal papers from Elsevier and 2 conference papers and 3 journal papers selected from Springer. To the best of our knowledge, only one paper published in 2013 that deals with the systematic investigation of CCD, therefore, we select papers from 2013 onward. We consider all journal papers published from 2013 to 2018. There is no journal paper related to CCD available in 2013, 2 papers published in 2014, 2 papers published in 2015, 1 study published in 2016, 2 studies in 2017 and 3 studies published in 2018. The journal papers represented by a brown bar in FIGURE 3.

Similarly, conference papers published from 2015 to 2018 are selected. 5 conference papers found published in 2015, 11 papers published in 2016, 20 studies published in 2017 and 8 studies published in 2018 as represented by a blue bar in FIGURE 3.

TABLE 4. Classification results of selected studies.

Sr.#	Category	References of Corresponding Studies	Total
1	Textual	[24][25][26][27][28][29][30][31][32][33][34][35]	12
2	Lexical	[36][37][38][39][40][41][42][43]	8
3	Tree Based	[44][45][46]	3
4	Metric Based	[47][48][49][50][51][52][53]	7
5	Semantic	[54][55][56][57][58][59][60]	7
6	Hybrid	[61][62][63][64][65][66][67][68][69][70][71] [72][73][74][75][76][77]	17

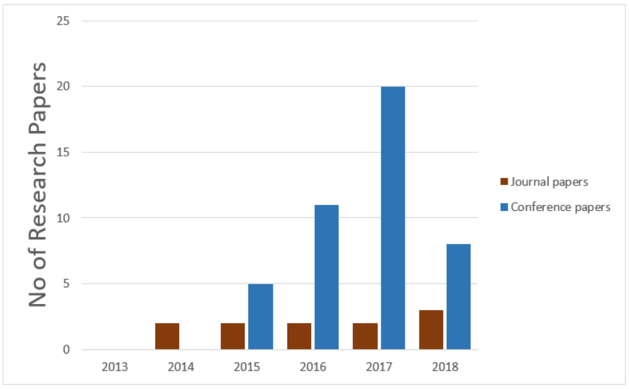

FIGURE 3. No. of selected studies w.r.t publication year.

E. DATA EXTRACTION AND SYNTHESIS

We have developed a template to extract data and perform synthesis as presented in TABLE 5. Firstly, the mining of bibliographic information of each selected study is performed. After that, core findings such as the proposed methodologies and implementation details of each selected study are extracted. In order to achieve the goals of literature, this provides the basis to carry out a detailed analysis. Moreover, tools that have been utilized or developed in the selected studies are identified. Furthermore, we find different open source software systems whose source code is used for CCD in the given studies. Finally, a comprehensive analysis is performed to get the answers of the RQ's as given above.

III. RESULTS AND ANALYSIS

The main objective of this article is to examine the given literature according to the research questions. Out of 54 research studies, 10 are published as journals and 44 are published in international conferences. Studies related to CCD are published in wide-range of conference and journal proceedings. It can be noticed that journals like Journal of Network and computer application, Expert system with applications, IEEE transaction on software engineering, Computer and Electrical engineering, the journal of system and software and Journal of computer science and technology are highly contributing to our research. Similarly, there is a wide variety of conferences such as conferences like International conference on software engineering, a conference on Software Analysis, Evolution, and Reengineering contribute largely in the selection process of studies. Almost 83% of our literature published in conferences and 17% published in journals. These research studies divided into six categories as presented in TABLE 4. For further analysis references of corresponding studies given against each category.

It can be seen that in TABLE 4, the Textual Approaches consist of twelve studies, Lexical Approaches comprises eight research studies, Tree-Based Approaches consist of three studies, Metric based Approaches comprises seven studies, Semantic Approaches comprises seven research studies, and Hybrid Approaches contain seventeen studies. The detail of these categories summarized in subsequent sections.

A. TEXTUAL APPROACHES

Several CCD techniques are based on text-based approach. These techniques consider the source code as a sequence of lines or strings. To find the sequences of the same lines, two code pieces are compared with each other. Whenever at least two code fragments found to be similar then by detection technique they are returned as clone class or clone pair. No or little transformation is done with source code because these are purely text-based techniques. In Table 5, CCD based on Textual approaches is analyzed with the parameters given below.1) **Language** describes the language of source code that is used for clone detection. 2) **Input Type/Intermediate state** shows that input taken by clone detection technique or intermediate format in which source code transform before clone detection. 3) **Algorithm/Classifier used** indicates that algorithms/Classifiers utilized for the identification of clones. 4) **Clone Type Detected** shows types of clones detected in these studies. The summary of these research studies is given below.

In Table 5, Ragkhitwetsagul and Krinke [24] utilize compilation/de-compilation to enhance clone detection. For this purpose, they use NICAD, a text-based code clone detector, java source code as input and uncover type1, type2 and

TABLE 5. **Summary of research studies using textual approaches.**

Sr.#	Reference No	Language	Input Type/ Intermediate State	Algorithm /Classifier Used	Clone Type Detected
1	[24]	Java	Source Code	N/A	1,2,3
2	[25]	C/C++	Fingerprints	N/A	1,2
3	[26]	C	Feature Set	SVM	3
4	[27]	Java	.txt files	N/A	1,2,3
5	[28]	C/C++	Fingerprints	MD5 Hash	1, 2
6	[29]	Multiple e.g. (HTML,Javascript)	Source Code	N/A	3
7	[30]	Layout XML Files	Layout Trees	CTPH Hash	1,2,3
8	[31]	Assembly	Feature Vector	AP Clustering	1,2,3
9	[32]	Java	N/A	N/A	1,2,3
10	[33]	C/C++,ST	Normalized Source code	N/A	1,2
11	[34]	C/C++	srcML	N/A	N/A
12	[35]	C/C++	Preprocessed Source code	MD5 Hash	N/A

type3 clones. Kim and Lee [25] introduce Vuddy, a scalable approach for vulnerable clone discovery by utilizing C/C++ programs as source code and generate fingerprints that are utilized as input. This approach has the ability to identify type1 and type2 clones. Jadon [26] proposes a technique for the detection of similar clones (type 3) and quantify their similarity. The proposed technique detects similar clones (type 3) by using C programs as source code, feature set as an intermediate format and Support Vector Machine (SVM) classifier is used for classification. Yu *et al.* [27] propose multigranuality CCD method based on Java bytecode by utilizing Java source code that transforms into the .txt format and uncovers type1, type2 and type3 clones. Kim *et al.* [28] present Vuddy, a scalable approach for vulnerable code clone discovery. The presented approach utilizes C/C++ programs as source code and generate fingerprints from this source code, which are utilized as input for CCD. MD5 Hash algorithm is used to produce hash values. It can detect type1 and type2 clones.

Nakamura *et al.* [29] introduce an approach to detect inter-language clones for a multilingual web application. Authors utilize source code of multiple programming languages. Pattern mining is used to identify frequently co-used programming languages. This approach is able to detect type3 clones. Lyu *et al.* [30] propose SuiDroid, an approach for android app clone detection. It is implemented by using python and shellcode. SuiDroid utilizes Layout XML files to identify the apps, layout trees as intermediate representation and CTPH Hash algorithm to measure the similarity. Results indicate that type1, type2 and type3 clones are identified. Xue *et al.* [31] describe a novel framework, clone hunter that integrates machine learning based binary CCD to speed up the elimination of redundant array bound checks in binary executables. They utilize assembly code as source code and Feature vectors as an intermediate format. AP Clustering algorithm is used for binary CCD. This framework uncovers type1, type2 and type3 clones. Chen *et al.* [32] apply NICAD, a text-based code clone detector for detecting android malware. For this purpose, Java source code is used as input and as we know that NICAD can detect type1, type2 and type3 clones so we can assume that these types of clones are identified. Thaller *et al.* [33] describe the results from the analyses of code clones in real-world PLC software. These results show that normalized C/C++, ST source code is utilized and type1 and type2 clones are detected. Newman *et al.* [34] develop a tool named as srSlice. It utilizes C/C++ source code, which is transformed into srcML as an intermediate format, and uncover code clones. Liu *et al.* [35] propose VEDFECT a vulnerable code clone system. For this purpose, C/C++ programs utilize as input, MD5 Hash algorithm that is applied on code blocks, (which are different from preprocessed code blocks) to construct fingerprints.

TABLE 6. Summary of research studies using lexical approaches.

Sr.#	Reference No	Dataset	Data Structure	Algorithm Used	Types of clones Detected
1	[36]	IJaDataset 2.0[i]	Delta inverted index	N/A	1,2,3
2	[37]	N/A	DAG	N/A	1,2
3	[38]	Zlib, DLL18, Malware297, DLL1GB	N/A	LSH	1, 2,3
4	[39]	C and Java files	N/A	N/A	1,2,3
5	[40]	Java files	N/A	Smith-Waterman	1,2,3
6	[41]	IJaDataset	Inverted Index	N/A	1,2,3
7	[42]	from Rosetta Code[ii]	N/A	N/A	1,2
8	[43]	IJaDataset[i]	N/A	Deep Neural Network	1,2,3

[i]https://www.dropbox.com/s/xdio16u396imz29/IJaDataset_BCEvalVersion.tar.gz?dl=0

[ii] http://www.rosettacode.org/wiki/Rosetta_Code

By matching, the preprocessed code blocks with fingerprints VEDEFECT uncovers the vulnerable code clones.

B. LEXICAL APPROACHES

Lexical approaches are also known as token-based approaches. These approaches consist of two steps, lexical analysis and clone detection. They transform targeted source code into a sequence of tokens with the help of laxer or parser. The sequence of tokens is scanned to find duplicate subsequences of tokens and finally, the original code fragment that represents the duplicate subsequences will be returned as clones.

In TABLE 6, CCD based on Lexical approaches is analyzed with the parameters given below. 1) **Dataset** describes the datasets available in these studies, which are converted into tokes. 2) **Data Structure** evaluates the data structure used for clone detection. 5) **Algorithm Used** indicates algorithms utilized to measure similarity. 5) **Clone Type Detected** evaluates types of clones detected in these studies. The summary of these research studies is provided in subsequent paragraphs.

In TABLE 6, Nishi and Damevski [36] introduce a clone detection approach by applying adaptive prefix filtering heuristic. It utilizes IJaDataset 2.0, a clone detection benchmark. As a data structure Delta inverted index is used for retrieving matching documents. This approach is able to find type1, type2 and type3 clones. Tekchandani *et al.* [37] present git code clone genealogy extraction model by utilizing the DAG data structure and can detect type1 and type2 clones. Farhadi *et al.* [38] present scalclone, a scalable assembly code clone search system by using Zlib, DLL18, Malware297 and DLL1GB datasets. LSH algorithm is applied to find inexact clones. It can detect type1, type2 and type3 clones. Wang *et al.* [39] develop CCAligner, a token based clone detector. It employs C, Java files as a dataset, find type1, type2, and type3 clones.

Yuki *et al.* [40] present a technique to detect multi-grained code clones. In the presented technique, Java files utilized as a dataset and Smith-Waterman algorithm utilized to identify the identical hash sequence. It uncovers type1, type2 and type3 clones. Sajnani *et al.* [41] propose SourcererCC, a token based clone detection tool. It employs IJaDataset. The inverted index data structure is applied to quickly query the proportional clones of a given code block. To measure recall, two benchmarks are used: 1) BigCloneBench, a benchmark of real clones, 2) Mutation/Injection based, the framework of thousands of artificial fine-grained clones. It can identify type1, type2 and type3 clones. Similarly, Semura *et al.* [42] develop another clone detection tool CCFinderSW. It takes dataset from Rosetta Code, a webpage that provides source code implemented in various programming languages, and uncovers type1 and type2 clones. Li *et al.* [43] present CCLEARNER, a deep learning based clone detection approach. This approach utilizes IJaDataset (java code) that transforms into tokens. Deep learning algorithm used by this approach is DNN and uncover type1, type2 and type3 clones.

C. TREE-BASED APPROACHES

In tree-based clone detection techniques the program is parsed to parse tree or abstract syntax tree with the help

TABLE 7. Summary of research studies using tree based approaches.

Sr.#	Reference No	Algorithm Used	Intermediate Representation	Machine Learning	Open Source software's	Clones Detected	Tool Support
1	[44]	Smith-Waterman	Variant of AST	N/A	JDK, Ant, Tomcat, ANTLR , dnsjava	Function	N/A
2	[45]	BFGS quasi-Newton method, MULTI-OBJECTIVE Genetic, NSGA-II	AST	Time Series Analysis, NN	ArgoUML	1, 3	CloneDr
3	[46]	Pattern Recognition	AST	Pattern Recognition	N/A	1,2	N/A

of laxer or parser. After that similar subtrees are searched by using a tree matching approach. When it matches, the corresponding source code of similar subtrees is returned as clone class or clone pair. In TABLE 7, CCD based on these approaches is analyzed with the parameters given below. 1) **Algorithm Used** evaluates, the algorithms utilized to measure similarity in these studies. 2) **Intermediate Representation** shows the state in which source code is converted before clone detection. 3) **Machine Learning** describes the machine learning techniques utilized for clone detection 4) **Open Source Software's** indicate that whose datasets or source code used for CCD. 5) **Clones Detected** describes types of clones detected in these studies. 6) **Tool Support** describes whether the tool used or develop support the Tree-based approaches. The summary of these research studies is given in subsequent paragraphs.

In TABLE 7, Yang *et al.* [44] propose a CCD technique based on automated functions. Firstly, it creates AST from functions, transforms it into a new tree structure and then utilizes the Smith-Waterman algorithm to obtain similarity score between functions. The experiment is conducted by using five open source projects (JDK, Ant, Tomcat, ANTLR, dnsjava) and function level (1, 2, 3 or 4) clone are detected. Pati *et al.* [45] discuss a method for appropriate checking and predicting evaluation of clone numbers across various versions of open source software applications by comparing three models BP-NN, ARIMA and MOGA-NN.

For this purpose, it utilizes AST as an intermediate format, Multi-Objective Genetic Algorithm (MOGA) for optimizing two objective functions as a cost function, BFGS quasi-Newton method for training neural network and Time series to evaluate clone components. ArgoUML is applied for the implementation of the experiment and CloneDr used as a support tool. This method uncovers type1 and type3 clones. Chodarev *et al.* [46] proposed an algorithm for clone detection in the program source code. For this purpose, AST is utilized as an intermediate state and Pattern recognition algorithm is used to identify potential clones. It has the ability to detect type1 and type2 clones.

D. METRIC BASED APPROACHES

In metric-based approaches, metrics are utilized to measure clones in software after the calculation from source code. For syntactic units such as function software or class, statement metrics are calculated and after that, comparison of these metrics values is performed. If two syntactic units have the same metric value, they can be considered as clone pair. For the calculation of the metric, this technique can also parse the source code to AST/PDG representation.

In TABLE 8, CCD based on metric-based approaches is analyzed with the following parameters.1) **Input Type/Intermediate State** shows that input taken by clone detection technique or intermediate format in which source code transform before clone detection.

2) **Similarity Measure** shows measuring of similarity for clone detection. 3) **Dataset** describes the datasets available in these studies on which clone detection is performed. 4) **Machine Learning** evaluates the machine learning techniques or algorithms utilized for clone detection.5) **Clones Detected** describe types of clones identified in these studies. The summary of these research studies is given in subsequent paragraphs.

In TABLE 8, Tsunoda *et al.* [47] assess the differences in clone detection methods utilized in fault-prone module prediction. For this purpose, the source code is used as input, the dataset is collected from Lucene 2.4.0, an open source software, and Logistic regression is used to build prediction models. Svajlenko and Roy [48] overviewed the concepts of CloneWorks, a near miss (type 3) clone detection tool by using IJaDataset, Jaccard similarity metric for clone detection. Sudhamani and Rangarajan [49] propose a method to detect duplicate clones. The experiment is conducted on dataset downloaded from fisourcecode. The source code is utilized as input, the similarity is measured by applying the self-defined formula and K-mean clustering is utilized for grouping the similar values where K= 2. It is able to find all types (type1, type2, type3 and type4) of clones. In another work, Svajlenko and Roy [50] provide further details of CloneWorks, a clone detector, by utilizing

TABLE 8. Summary of research studies using metric based approaches.

Sr.#	Reference No	Input Type/ Intermediate State	Similarity Measure	Dataset	Machine Learning	Clones Detected
1	[47]	Source Code	N/A	From Lucene[iii]	Logistic Regression	N/A
2	[48]	Source Code	Jaccard	IJaDataset[i]	N/A	1,2, 3
3	[49]	Source Code	Self-defined Formula	From fisourcecode	K-mean clustering	1,2,3,4
4	[50]	Source Code	Jaccard	IJaDataset[i]	N/A	1,2, 3
5	[51]	Source Code	N/A	N/A	N/A	1,2,3,4
6	[52]	PNG Image	Jaccard	Java source code Files	N/A	1, 2,3
7	[53]	Source Code	Self-defined Formula	C/C++ Java files	N/A	Structural

[i]https://www.dropbox.com/s/xdio16u396imz29/IJaDataset_BCEvalVersion.tar.gz?dl=0

[iii] https://lucene.apache.org/core/

IJaDataset source code as input and Jaccard similarity metric for clone detection. Results show that it can identify type1, type2 and type3 clones. Haque *et al.* [51] develop a generic technique to detect code clones from different input source codes by dividing the code into a number of functions or modules. This approach is a combination of more approaches and methods. It has the ability to uncover all types of clones. Ragkhitwetsagul *et al.* [52] present an image based clone detection approach and a tool named Vincent. It applies java source files as a dataset, transforms it into PNG image as an intermediate state and then utilizes Jaccard similarity for clone detection. Results indicate that type1, type2 and type3 clones are identified. Sudhamani and Rangarajan [53] address structure similarity detection using the structure of control statements. For this purpose, C/C++, java files are used as dataset and self-defined formula used for similarity computation. This method can efficiently uncover structurally similar (type1, type2, type3, or type4) clones.

E. SEMANTIC APPROACHES

In these techniques, the program is represented as a program dependency graph (PDG). Approaches that depend on program dependency graph goes one-step further to obtain high abstraction of source code representation than others because it considers semantic information of the source. Program dependency graph carries control flow and data flow information and hence contain semantic information. Once a set of PDGs is obtained, the isomorphic subgraph matching algorithm is applied for finding similar subgraphs which are returned as clones.

In TABLE 9, CCD based on semantic approaches is analyzed with the help of following parameters. 1) **Algorithm Used** shows that the algorithms utilized for identification of clones. 2) **Similarity Measure** indicates measuring of similarity for clone detection. 3) **Language** represents the language of the source code, which is transformed into PDG. 4) **PDG Constructor** describes a framework or anything that helps in the construction of PDG from source code. 5) **Clone Type Detected** evaluates types of clones detected in these studies. Summary of these studies is given in subsequent paragraphs.

In TABLE 9, Wang *et al.* [54] present CCSharp: An efficient three-phase clone detector using modified PDGs. It applies Frama-C2 to generate program dependency graphs of source code in C language.

It utilizes Vector filtering algorithm to exclude the PDG pairs, which are not likely to be cloned. Euclidean distance is used to measure the numerical similarity and Levenshtein Distance is utilized to measure string similarity. As this tool is based on PDG based approach, so we can suppose it has the ability to detect type 4 clones. Sabi *et al.* [55] examine how clone detection result changes by rearranging the program statements by using PDGs. For this purpose, the Java source code is used. Results show that type1 and type2 clones are identified. Crussell *et al.* [56] propose AnDarwin, a tool for finding applications with the similar code on large scale by utilizing WALA to generate program dependency graphs of source code in C language, LSH algorithm for finding an approximate nearest neighbor in a large number of vectors and Min-Hash algorithm to measure partial or full app similarity.

Sargsyan *et al.* [57] propose an algorithm for scalable and accurate clone detection. For this reason, PDG is constructed by a compilation of C program files, LLVM is used as compilation infrastructure and Isomorphism algorithm is used for similarity measure. The proposed algorithm can identify type 4 clones. Similarly, Hu *et al.* [58] present another algorithm by utilizing java source code to identify

TABLE 9. Summary of research studies using semantic approaches.

Sr.#	Reference No	PDG Constructor	Language	Algorithm Used	Similarity Measure	Clone Type Detected
1	[54]	Frama-C2	C	Vector filtering	Euclidean Distance, Levenshtein Distance	4
2	[55]	N/A	Java	N/A	N/A	1,2
3	[56]	WALA	Java	LSH	Min-Hash	N/A
4	[57]	LLVM	C	Fast Checking	Isomorphism	4
5	[58]	IDA Pro	Assembly	LCS	Min-Hash	4
6	[59]	N/A	Java	ASM	N/A	4
7	[60]	LLVM	C	N/A	N/A	1,2,3,4

new clone relations from the clone pair results of PDG base detection. For this purpose, the ASM algorithm is used and type 4 clones are identified. Kamalpriya and Singh [59] propose a semantic-based approach to find functions of binary clone and implement this approach in a prototype system named CACOMPARE. The experiment is conducted by using the binary code in assembly language, IDA Pro dissemble this code and extract CFGs, Min-Hash algorithm to quickly estimate the Jaccard index and LCS algorithm for similarity score computation. The result indicates that type 4 clones are detected. Avetisyan *et al.* [60] present a framework for CCD that is based on LLVM. It utilizes the source code written in C language and LLVM for the transformation of bytecode into PDGs. It uncovers type1, type2, type3 and type4 clones.

F. HYBRID APPROACHES

The combination of two or more CCD approaches (Textual, Lexical, Syntactic or Semantic) is called a hybrid approach. The hybrid approach holds better results than the normal one [61]. CCD based on hybrid approaches is analyzed in Table 10 with the help of following parameters.1) **Hybrid** shows the combination of clone detection approaches used as a hybrid. 2) **Dataset** indicates the dataset available in the form of source code which is converted into different states for clone detection.3) **Transformation** describes the source code undergoes in different forms for clone detection. 4) **Algorithm Used** shows the availability of algorithms for similarity measure or clone detection. 5) **Clone Type Detected** shows that types of clones detected in these studies. The summary is given below.

In Table 10, Singh [61] focuses on enhancements in the CCD algorithm by using a hybrid approach, that is a combination of metric based approach and PDG based approach. The dataset used by this approach is Java source code, which is transformed into AST and PDG. It can identify type1, type2 and type3 clones. Misu and Sakib [62] develop an interface driven CCD approach (IDCCD) by combining token based and metric-based approaches. For this purpose, IJaDataset is used that transforms into regularized tokens and ASTs. It has the ability to find type1, type2 and type3 clones. Sheneamer and Kalita [63] propose an efficient metric based approach for clone detection and it extracts features from ASTs and PDGs.

It utilizes IJaDataset 2.0 (Java code) that undergoes AST and PDG transformation, and Rotation Forest, Random Forest, Xgboost algorithms that can detect clones automatically. This approach can find type1, type2, and type3 and type4 clones. Vislavski *et al.* [64] describe LICCA, a tool for cross-language clone detection that is a combination of token based, AST based and metric-based approaches and uncovers type1, type2 and type3 clones. This tool utilizes Java, C, JavaScript, Scheme and Modula-2 code as a dataset, eCT representation for AST based detection and a variant of LCS algorithm for token based detection. Misu *et al.* [65] describe an exploratory study on interface similarity in code clones. For this purpose, token-based and text-based tools are used and Java source code undergoes AST transformation. Results indicate that type1, type2 and type3 clones are identified. Akram *et al.* [66] develop Droid CC a clone detection approach, by combining text-based and token based approaches for android applications. The dataset utilized by this approach is java code that transforms into regularized tokens. The MD5 Hashing algorithm is used to get hash values against each chunk. This approach can detect type1, type2 and type3 clones.

Sheneamer *et al.* [67] introduce a framework for obfuscated and semantic clones by using machine learning. It is a combination of semantic and tree-based techniques and

TABLE 10. Summary of research studies using hybrid approaches.

Sr.#	Reference No	Hybrid	Dataset	Transformation	Algorithm Used	Clone Type Detected
1	[61]	PDG Based + Metric Based	Java Code	AST + PDG	N/A	1,2,3
2	[62]	Token-Based + Metric Based	IJaDataset2.0[i] (Java code)	AST+ Tokens	N/A	1,2,3
3	[63]	AST Based + PDG Based	IJaDataset2.0[i] (Java code)	AST+PDG	Rotation Forest, Random Forest, Xgboost	1,2, 3,4
4	[64]	Token Based+ Tree-Based + Metric Based	Java ,JavaScript , C , Modula-2 , Scheme	eCT	LCS	1,2,3
5	[65]	Token-Based + Tree-Based	Java code	AST	N/A	1,2,3
6	[66]	Textual + Token-Based	Java code	Tokens	MD5 Hashing	1,2,3
7	[67]	Tree-Based + Semantic	Java code	BDG, AST, PDG	Naïve Bayes, SVM (IBK),Logit Boost, Random Committee, Ran- dom Subspace, Rotation Forest, Random Forest, J48	1,2,3,4
8	[68]	Token-Based + Tree-based	ML programs	Regularized tokens, AST	N/A	1,2,3
9	[69]	Textual + Metric	C, Java	N/A	Clustering	1,2,3,4
10	[70]	Token Based+ AST Based + Semantic	Java Code	Tokens, AST	N/A	4
11	[71]	Token-based + Metric Based	HDL code	C++ ,Tokens	N/A	1,2
12	[72]	Token-Based + PDG Based	C code	Tokens, PDG, AMS	N/A	4
13	[73]	Text Based + Metric Based	C , C# , Java ,Text files	N/A	N/A	Structural
14	[74]	Lexical + Metric Based	Murakami's(Java files)[iv]	Regularized Tokens	N/A	1,2,3
15	[75]	Lexical + Tree Based+ Metric Based	Java Code	AST	RtNN	3
16	[76]	Text Based + Token-Based + Tree-Based	Java Code files	Bytecode, Tokens, AST	N/A	N/A
17	[77]	Token-Based + Metric Based	N/A	Tokens	DNN	4

[i]https://www.dropbox.com/s/xdio16u396imz29/IJaDataset_BCEvalVersion.tar.gz?dl=0

[iv]http://sdl.ist.osaka-u.ac.jp/-h-murakm/20 14_ clone_references with_gaps

Java code is used as a dataset that transforms into BDG, AST and PDG. In order to train and test the model ensemble approach (majority voting) among ten classifiers (i.e. Naïve Bayes, IBK, SVM, Logit Boost, Ran-dom Subspace, Random Committee Rotation Forest Random Forest J48) is utilized. This framework can detect type1, type2, type3 and type4 clones. Matsushita and Sasano [68] present an algorithm that detects clones with gaps by using the combination of token based and AST based approaches. ML programs used as a dataset that transforms into regularized tokens and ASTs. The algorithm can find type1, type2 and type3 clones. For method level clone detection, Kodhai and Kanmani [69] propose an approach named LWH. This is a combination of metric-based and text-based techniques. It utilizes C and Java code as a dataset, clustering algorithm to measure the similarity and uncover type1, type2, type3 and type4 clones. Similarly, Tekchandani *et al.* [70] present another algorithm that is a hybrid of token based, AST based and semantic approaches for IoT applications by using Java source code that transforms into Tokens and ASTs. It can identify type 4 clones. Uemura *et al.* [71] propose a method CCD in Verilog HDL by combining token based and metric-based techniques. The dataset employed by this method consists of HDL code which is transformed into C++ code and then into tokens. It can detect type1 and type2 clones. Nasirloo and Azimzadeh [72] present a method for semantic (type 4) CCD using AMSs and PDGs. It applies C source files as a dataset that transforms into tokens PDGs and AMSs before clones are detected. Singh and Sharma [73] present a hybrid approach (text-based + metric Based) to detect file level clones for high-level cloning by applying dataset that consist of C, C#, Java and text files. It can detect structural (1, 2, 3 or 4) clones. Sheneamer and Kalita [74] present hybrid CCD technique using fine-grained and coarse-grained techniques. It utilizes the combination of lexical and metric-based approaches, Murakami's (java files) dataset that transforms into regularized tokens and uncovers type1, type2 and type3 clones.

White *et al.* [75] present a technique for CCD based on deep learning of code fragments by combining lexical, tree-based and metric-based techniques. This technique utilizes RtNN as deep learning algorithm and Java source code as a dataset, which transforms into tokens and ASTs. ASTs then transform into a full binary tree. It can detect type 3 clones. Ragkhitwetsagul *et al.* [76] compare the code similarity analyzers. For this purpose, they utilized java source code and transforms it into bytecode, which is utilized for similarity or clone detection. For clone detection bytecode further transforms into tokens or ASTs by using different clone detection tools (simian, NICAD, CCFinderX, iclones, Deckard) supported by clone detection techniques (text-based, token-based, tree-based). Results indicate that these clone detection tools and techniques perform better than general similarity measures. Ghofrani *et al.* [77] introduce a framework for clone detection using a deep neural network as a machine learning algorithm and hybrid (token based and metric-based) technique as a CCD technique. Regularized tokens utilized as an intermediate format and type 4 clones are identified.

G. CLONE DETECTION TOOLS

Overall 26 tools (13 existing and 13 developed) from 54 selected studies are identified as presented in TABLE 11 and TABLE 12. Detail of these tools provided in subsequent paragraphs.

***Existing Tools*:** There are 13 tools based on CCD techniques utilized in selected studies as summarized in TABLE 11 with parameters given below: 1) **Tool Name** indicates that the name of the tool. 2) **Technique** indicates the CCD techniques that support these tools. In this parameter, technique names are provided. 3) **Availability** indicates that the tool is openly available or not. 4) **License Type** shows that the type of license under which these tools released. **References of** these studies are given for further details. From TABLE 11, we overall analyzed 13 tools where three tools (NICAD, Simian and Duploc) based on text-based techniques, six tools (SourcererCC, CCFinderX, CCFinder, DUP, iClones and CPD) based on lexical techniques. Furthermore, two tools (Deckard, and CloneDR) based on tree-based techniques, one (CLAN) based on metric based technique and one (Duplix) based on PGD based technique.

NICAD is a text-based hybrid code clone detector. It can detect method level clones, and with the help of its pre-defined configuration files it categories these clones into exact (type1), parameterized (type2) and gapped (type3) clones [24], [65]. It is based on TXL [32] and embed it for pretty printing, and compares source code by using string similarity [76]. It is an open source tool and released under the license of BSD. ***Simian*** is a clone detection tool based on text-based technique [47]. It uncovers exact (type 1) clones in all source codes but for several programming languages, it incorporates additional normalization features [33]. It relays on the comparison of text line with an ability for checking basic modification of code [76]. ***Duploc*** is a text-based detector and identifies code clones by normalizing source code [69]. It is freely available and released under GPL license.

SourcererCC is a clone detection tool, which is based on the lexical approach [36]. It has the ability to achieve high precision and recall as compared to other tools. SourcererCC is used to detect inter-project and intra project method clones [65]. It is an open source and released under GPL license. ***CCFinderX*** is also a tool that based on lexical (token based) approach [47], [37]. To identify syntactic units such as statements or functions it parses the code. However, it does not strictly perform parsing to construct the entire AST [71]. It uses suffix trees to detect similarity [76]. This tool is openly available for education, research and in house use. Its license type is Freeware. Similarly, ***CCFinder*** is also a detector based on lexical (token based) approach [38], [68], [69]. It is able to identify type 1 and type 2 clones in a short time [39]. This tool is also openly available for education, research and in house use. Its license type is Freeware. There is another tool ***DUP*** based on tokens [69], freely available and its license type is

TABLE 11. Summary of tools based on CCD techniques utilized in selected studies.

Sr.#	Tool Name	Technique	Availability	License Type	References
1	NICAD[v]	Textual	Yes	BSD	[24][32][65][76]
2	Simian[vi]	Textual	NO	N/A	[33][47][76]
3	Duploc[vii]	Textual	Yes	GPL	[69]
4	SourcererCC[viii]	Lexical	Yes	GPL	[36][65]
5	CCFinderX[ix]	Lexical	Yes	Freeware	[37][47][71][76]
6	CCFinder[x]	Lexical	Yes	Freeware	[38][68][69][39]
7	DUP[xi]	Lexical	Yes	Freeware	[69]
8	iClones[xii]	Lexical	Yes	Apache	[76]
9	CPD[xiii]	Lexical	Yes	Apache	[47]
10	Deckard[xiv]	Tree-Based	Yes	GPL	[76]
11	CloneDR[xv]	Tree-Based	No	N/A	[45][69]
12	CLAN	Metric Based	No	N/A	[69]
13	Duplix	PGD based	No	N/A	[69]

[v] http://www.txl.ca/nicaddownload.html
[vi] http://www.harukizaemon.com/simian/index.html
[vii] https://github.com/xsgordon/duplo-fork
[viii] https://github.com/Mondego/SourcererCC
[ix] http://www.ccfinder.net/ccfinderx.html,
[x] https://github.com/radekg1000/ccfinderx
[xi] http://www.tucows.com/preview/1097147/DupDetector
[xii] http://www.softwareclones.org/geticlones.php
[xiii] https://github.com/pmd/pmd
[xiv] https://github.com/skyhover/Deckard
[xv] http://www.semdesigns.com/products/clone/JavaClo neDR.html

Freeware. Furthermore, ***IClones*** is also a token-based detector. Over several revisions of programs, it performs token-based incremental clone detection [76]. It is open source for academic utilization and released under Apache license. Another tool ***CPD*** based on the lexical approach [47] is also open source and released under Apache license. [47]. ***Deckard*** is a detector based on a tree-based approach. It converts source code into an abstract syntax tree and computes similarity by comparison of characteristic vectors generated from the abstract syntax tree. This tool identify clones based on approximate tree similarity [76]. It is freely available and released under GPL license. Similarly, ***CloneDR*** is also a clone detection tool based on a tree-based approach. It uses abstract syntax tree for CCD [45], [69]. ***CLAN*** is metric based code clone detector [69]. ***Duplex*** is a detector that detects clones based on program dependency graph [69].

Tools Developed: We identified 13 tools proposed / developed for CCD in 54 selected research studies, as presented in TABLE 13, with parameters given below: 1) **Tools Name** indicates the name of the tool which is proposed or developed. 2) **Supported Platform** evaluates OS/CPUs supported by the tool. 3) **Supported Language** shows that the language of source code used for clone detection supported by the tool.

SourcerereCC is a token-based detector of code clones and detects near-miss clones (Type3) accurately. To attain large-scale detection of clones on the standard workstation, it utilizes filtering heuristics and optimized the partial index. It works in two phases: 1) Partial Index Creation and 2) Clone Detection. It can also identify type1 and type2 clones, support Java, C and C# and executed on a standard platform with a quad-core i7 CPU, solid-state drive and 12GB of memory [41]. **A decrescendo** is a tool that is developed to detect code clones. It takes single or multiple software's, minimum clone length and maximum gap rate as inputs, and the output consist on method level, file level and code fragment level clones. By setting, it can switch detection on/off for each granularity (method, file or code fragment). It supports source code written in Java and executed on a workstation with 2.40 GHz Intel Xeon CPU (8 logic processors), and 32.0 GB of memory. Output databases and experimental targets located on SSD [40]. **VUDDY** is implemented as a technique for the discovery of vulnerable clones but compared with clone detection tools to determine accuracy. Modeling of VUDDY consists of two stages: 1) preprocessing and 2) clone detection. Its design principles are directed towards length filtering while maintaining accuracy

TABLE 12. **Summary of clone detection tools proposed/developed in selected studies.**

Sr.#	References	Tool Name	Supported Platform	Supported Language
1	[41]	SourcererCC[viii]	quad-core i7 CPU	Java, C, C#
2	[40]	Decresendo	2.40 GHz Intel Xeon CPU	Java
3	[28][25]	VUDDY[xvi]	Ubuntu 16.04	C/C++
4	[64]	LICCA	N/A	Java, C, JavaScript, Modula-2 Scheme
5	[54]	CCSharp	Ubuntu	C
6	[42]	CCFinderSW	Linux kernel	Multiple
7	[43]	CCLearner[xvii]	N/A	Java
8	[48][50]	CloneWorks[xviii]	3.6GHzquad-core-i7-2600	Java
9	[52]	Vincent	Ubuntu 16.04.1	Java
10	[39]	CCAlinger	N/A	C/Java
11	[56]	AnDarwin	quad Intel Xeon E7-4850 CPUs	Java
12	[34]	srcSlice[xix]	Linux kernel	C/C++
13	[69]	CloneManager	N/A	C, Java

[viii] https://github.com/Mondego/SourcererCC
[xvi] https://github.com/iotcube/hmark
[xvii] https://github.com/liuqingli/CCLearner
[xviii] https://github.com/jeffsvajlenko/CloneWorks
[xix] https://github.com/srcML/srcSlice

and extend scalability through functional-level granularity, so that it can manage to identify vulnerable clones from the speedily increasing pool of open source software. It supports source code written in C/C++ and executed on Ubuntu 16.04 with a 2.40 GHz Intel Zeon processor, 6 TB HDD and 32 GB RAM [28], [25]. **LICCA** is a tool designed for the detection of cross-language clones. It utilizes AST as intermediate representation and a variant of the LCS algorithm for the detection of the clone. It is utilized with the SSQSA platform. eCST representation of source code is used in LICCA for identification of clones which is provided by SSQSA eCST Generator [64]. **CCSharp**, a three-phase program dependency graph based clone detection technique that is implemented as a tool named CCSharp. These three phases consist of 1) PDG's Structure modification, to simplify overall structure of PDG, 2) Vector filtering, to cut down the pairwise comparison quantity of the PDG pairs and, 3) ARGs subgraph Isomorphism, to find clone pairs. CCSharp achieves high accuracy and recall as compared to the famous clone detection tools such as NICAD, Deckard, and SourcererCC. It supports source code written in C language. The experiment is conducted on Ubuntu with a quad-core CPU and the memory size is 8GB [54]. **CCFinderSW**, a token-based code clone detector that identifies code clones in multiple programming languages. The CCD process of CCFinderSW consists of four steps: 1) Lexical Analysis, 2) Transformation, 3) Detection, 4) Formatting. It has the ability to identify type1 and type2 clones and run on the Linux kernel [42]. **CCLearner** is designed and developed as an approach to identify clones with deep learning and token usage analysis. CCLearner is compared with CCD tools for checking its effectiveness. It consists of two steps: 1) Training, this step utilized both clones and non-clones for the training of a classifier in a deep learning framework. 2) Testing, this step utilizes codebase for the identification of clones with the trained classifier. It supports source code written in Java. With high precision and recall, it can identify different clones [43]. **CloneWork**s, a fast and flexible clone detection tool for large scale clone detection experiments. CloneWorks consist of three command-line tools 1) cwbuild for running input builder, 2) cwdetect for clone detection, 3) cwformate to formate the clone results [48]. For source code processing, before clone detection, CloneWorks gives the users full control for enabling the user to target any clone type or perform custom clone detection experiments. It supports Java, C and C# and runs on 3.6GHzquad-core-i7-2600 12GB of memory [48], [50]. **Vincent** is an image based clone detection tool. It detects clones at method level and currently supports Java. Based on visual representation it compares code fragments, which resembles how developers manually looking for clones. Vincent detects additional clone pairs that were not detected by any other tool. The experiment

TABLE 13. Summary of open source subject systems used in selected studies.

Sr.#	Subject System	Language	License Type	References
1	JDK	Java	MPL	[44][75][39][41]
2	Ant	Java	GPL	[44][27][39][59][69][74][75]
3	Tomcat	Java	GPL	[44][52]
4	ANTLR	Java	BSD	[44][61][43][75][24][70]
5	JEdit	Java	GPL	[61]
6	Eclipse	Java	EPL	[43]
7	Qpid	Java	ASL	[55]
8	Subversion	Java	GPL	[55]
9	Wookie	Java	GPL	[55]
10	Hibernate	Java	GPL	[75]
11	ArgoUML	Java	EPL	[45][75]
12	Swing	Java	GPL	[69][70][74]
13	Junit	Java	EPL	[24][52]
14	JfreeChart	Java	LGPL	[24][52]
15	Hadoop	Java	GPL,AGPL	[37]
16	Neo4j	Java	GPL,AGPL	[37]
17	EIRC	Java	GPL	[69]
18	Netbeans	Java	GPL	[69][27][74]
19	jdtcore	Java	EPL	[69][27][74]
20	JhotDraw	Java	LGPL	[69][75]
21	OpenNLP	Java	GPL	[39]
22	Maven	Java	GPL	[39]
23	RxJava	Java	N/A	[71]
24	Okhttp	Java	GPL	[71]
25	React-native	Java	GPL	[71]
26	Wget	C	GPL	[69]
27	Postgresql	C	GPL	[69][39]
28	LLVM	C/C++	GPL	[57]
29	Ffmpeg	C	LGPL	[35]
30	Firefox	C/C++ JavaScript	MPL	[35][57]
31	OpenSSL	C	BSD	[28][57][35]
32	HTTPD	C/C++	GPL	[28][35][25][69]
33	Linux	C	GPL	[28][35][57][69][39][71][60][41]
34	Mono	C#	LGPL	[41]
35	MonoDevelop	C#	LGPL	[41]
36	Webogram	Python	GPL	[29]
37	DNSjava	Java	BSD	[44][75]
38	jUDDI	Java	GPL	[40]
39	Gora	Java	GPL	[55]
40	Cook	C	N/A	[69][39]
41	netdata	Java	GPL	[71]
42	Redis	C	BSD	[72][39]
43	FREECOL	Java	GPL	[61]
44	Commons Lang	Java	GPL	[53][41]
45	Wink	Java	GPL	[40][53]
46	Lucene	Java	GPL	[47]
47	Google Android	Java C/C++ Python	GPL	[28]
48	Codeaurora Android	Java, C /C++	GPL	[28]
49	Google Chromium	C/C++ Java,Python	MPL,GPL, LGPL	[28]
50	Ubuntu-Trusty	Java C/C++ Python	GPL	[28]
51	Roller	Java	GPL	[40]
52	OpenOffice	Java	LGPL	[40]
53	OODT	Java	GPL	[40]
54	Onami	Java	GPL	[40]
55	JSPWiki	Java	GPL	[40]
56	Forrest	Java	GPL	[40][53]
57	Any23	Java	GPL	[40][55]
58	BVal	Java	GPL	[55]
59	Flume	Java	GPL	[55]
60	Giraph	Java	GPL	[55]
61	cTAKES	Java	GPL	[40]
62	FOP	Java	GPL	[70]

conducted on Ubuntu 16.04.1 machine with two 3.40 GHz processors and 8 GB of RAM [52]. **CCAlinger** can detect large gap clones which were missed by other competing tools. It consists of two phases: 1) Lexical Analysis and 2) Clone Detection. It supports source code written in C and Java and uncovers type1, type2 and type3 clones. It can achieve good recall and precision [39]. **AnDarwin** is a tool that is used to identify applications with a similar code on large scale. It accomplishes its task in two phases of clustering: 1) LHS, grouped semantic vectors into features and 2) minHash, find applications with similar features and feature sets that mostly occur together. It can also be used for the improvement of the marketing system. It is an efficient tool to find cloned and rebranded applications, support java language, runs on quad Intel Xeon E7-4850 CPUs and the memory size of 256 GB DDR3 [56]. **srcSlice**, a tool that implements a forward static slicing technique. This tool is enabled by the srcML infrastructure. srcML augments source code with abstract syntactic information. This syntactic information is utilized for the identification of program dependences as required when computing slice. Currently srcSlice support C and C++. The tool is run on the Linux kernel [34]. **CloneManager** is a tool that can identify potential clone pairs. It takes Java and C source code as input and separates the functions/methods present in it. These methods processed by utilizing built-in hand-coded parser following an island driven parsing approach. After identification of these methods, different source code matrices computed for each method and stored in the database. With the help of these matrices, the near equal methods are extracted and subject to textual comparison to identify potential clone pairs [69].

H. OPEN SOURCE SUBJECT SYSTEMS

We overall identify 62 open source subject systems whose datasets or source code used for CCD. These are summarized in TABLE 13 with the following parameters: 1) **Subject System** indicates that the name of the open source subject system whose source code utilized for CCD. 2) **Language** shows that the implementation language of these open source subject systems. 3) **License Type** represents the type of license under which they release. 4) **Reference** of the paper is provided for further detail.

We hope that TABLE 13 may help researchers in the selection of most frequently used open source subject systems as a benchmark for evaluation and empirical studies. Linux has been used commonly for CCD. Similarly, JDK, Apache-Ant, HTTPD and ANTLR are also widely used for CCD. All other open source subject systems utilized in one or two studies for CCD. Other than these subject systems, IJaDataset which is a benchmark dataset consist of java code repository is also extensively used for code clone detection.

IV. ANSWERS OF RESEARCH QUESTIONS

RQ1: What are the popular approaches utilized for CCD and which type of clones they can detect?

Answer: We overall identified six code clone detection techniques utilized in 54 selected studies. Categorization of all studies performed based on these techniques. The techniques include Text-based (Table 5), Token-based (TABLE 6), Tree-based (TABLE 7), Metric based (TABLE 8), Semantic (TABLE 9) and hybrid (Table 10). Twelve (12) studies are found based on Text-based techniques, eight (8) studies are found based on Token based techniques, three (3) studies based on Tree based techniques, seven (7) studies based on Metric based techniques, seven (7) studies based on Semantic techniques and seventeen (17) studies depend on Hybrid techniques.

It has been analyzed that text-based techniques consider the source code as a sequence of lines or strings. To find the sequences of the same lines, two code pieces are compared with each other. No or little transformation is done with source code because these are purely text-based techniques. These techniques detect type1 clones most effectively. However, they also has the ability to detect type2 and type3 clones. Similarly, token-based techniques consist of two steps lexical analysis and clone detection. They transform targeted source code into a sequence of tokens with the help of laxer or parser. The sequence of tokens is scanned to find duplicate subsequences of tokens and finally, the original code fragment that represents the duplicate subsequences will be returned as clones. They can find type2 clones effectively. However, they can also uncover type1 and type3 clones. Moreover, in tree-based techniques, the program is parsed to parse tree or abstract syntax tree with the help of laxer or parser. After that similar subtrees are searched by using a tree matching approach and subsequently, the corresponding source code of similar subtrees is returned as clone class or clone pair. These techniques uncover type3 clones effectively. Moreover, they can also identify type1, type2 and type4 clones.

In metric-based approaches, metrics are utilized to measure clones in software after the calculation from source code. For syntactic units such as function software or class, statement metrics are calculated, and after that, comparison of these metrics values is performed. Similar to tree-based, this approach can detect type3 more efficiently. They also have the ability to uncover type1, type2 and type4 clones. In semantic approaches, the program is represented as a program dependency graph (PDG). Once a set of PDGs is obtained, isomorphic subgraph matching algorithm is applied for finding similar subgraphs which are returned as clones. The semantic approach has the ability to detect type1, type2, type3 and type4 clones. They are mostly used to uncover semantically or type4 clones. Finally, in Hybrid approaches, two or more CCD techniques are combined. Types of clones detected depends on techniques, which are utilized in a hybrid. The hybrid approach holds better results than the normal one.

RQ2: What are the popular tools utilized/developed for CCD?

Answer: We overall identify 26 tools in literature for CCD. These tools categorized into existing tools (13) presented

TABLE 14. Comparative analysis of CCD techniques.

Sr.#	Technique	Accuracy	Execution Time	Clones Detected	Strength	Weakness
1	Textual	High	Low	1,2,3	Easy to implement	Cannot detect type 4 clones
2	Lexical	High	Low	1,2,3	Good Scalability	Cannot detect type 4 clones
3	Tree-Based	High	High	1,2,3,4	Contain more information about code structure	Difficult to build AST
4	Metric Based	High	Low	1,2,3,4	Can detect all type of clones	Difficult to implement for complex metrics
5	Semantic	High	High	1,2,3,4	Can detect all type of clones	Difficult to build PDG
6	Hybrid	Depends on hybrid technique	Depends on hybrid technique	Depends on hybrid technique	Depends on hybrid technique	Depends on hybrid technique

in TABLE 11 and proposed/developed tools (13) presented in TABLE 12. These tools based on CCD approaches. Analysis shows that in existing tools, three tools (NICAD, Simian and Duploc) based on text-based techniques, six tools (SourcererCC, CCFinderX, CCFinder, DUP, iClones and CPD) based on token-based techniques, two tools (Deckard, and CloneDR) based on tree-based techniques, one tool (CLAN) supported by metric based technique and one tool (Duplix) use semantic or PDG based technique. There is no existing tool found in the literature that belongs to the hybrid technique. Similarly, in proposed/developed tools, two tools (VUDDY, srcSlice) based on text-based techniques, five tools (SourcererCC, Decresendo, CCFinderSW, CCLearner, CCAlinger) supported by token-based techniques, two tools (CloneWorks, Vincent) based on metric based techniques, two (CCSharp, AnDarwin) tools supported by PDG based and two (LICCA, CloneManager) based on hybrid techniques. There is no tool proposed in the selected studies that belongs to the tree-based technique.

RQ3: What are the open source subject systems or benchmark datasets commonly used for CCD?

Answer: We have observed several open source subject systems whose source code is utilized for CCD. We overall found 62 open source subject systems (TABLE 13) in literature. The analysis shows that ant is utilized several times for CCD. Linux and its different versions are extensively utilized as open source subject system to carry out CCD. Similarly, JDK is also used to find out the code clones. Furthermore, ANTLR and HTTPD are also extensively used to identify the impact of code clones. All other open source subject systems utilized in one or two studies for CCD. Similar to software systems, two benchmark datasets (Mutation / Injection-based framework and IJaDataset) are also utilized for CCD. The comprehensive investigation of selected studies shows that IJaDataset, a big data software repository containing 25,000 open source java systems is most commonly used benchmark dataset.

RQ4: What are the advantages and limitations of CCD techniques and tools?

Answer: Before concluding the answer to this question, it is first required to investigate CCD techniques and tools from selected research studies. Therefore, we perform the comparative analysis of CCD techniques in TABLE 14 and CCD tools proposed / developed in TABLE 15 and Table 16 with the help of different evaluation parameters. Detail of this analysis is summarized in the subsequent paragraphs.

In TABLE 14, textual techniques have high accuracy, low execution time and can uncover type1, type2 and type3 clones. These techniques are easy to implement but unable to detect type 4 or semantic clones. Similarly, lexical techniques also have high accuracy, low execution time and can identify type1, type2 and type3 clones. These techniques comprise of good scalability but unable to detect type4 clones. Tree-based techniques, similar to textual and lexical techniques, have high accuracy. These techniques can reveal all types (type1, type2, type3 and type4) of clones and contain more information about the structure of code but the generation of ASTs is difficult and their time complexity is also high. Metric-based techniques comprise of high accuracy, low execution time and can detect all types (type1, type2, type3 and type4) of clones. However, when metrics become complex these techniques are difficult to implement. Similar to metric-based techniques, semantic approaches can reveal all types (type1, type2, type3 and type4) of clones and have high accuracy but PDGs generation is a difficult and time-consuming process. In hybrid techniques, accuracy, execution time, types of clones detected etc. depends on the combination of techniques that are utilized in hybrid technique.

In TABLE 15 and TABLE 16, Sajnani *et al.* [41] propose SourcererCC, a token-based code clone detector. Its scalability, execution time, precision and recall is evaluated and compared with four (NICAD, CCFinderX, Deckard

TABLE 15. Comparative analysis of CCD proposed/ developed tools w.r.t technique, Accuracy and clone detection abilities.

Sr.#	Reference No.	Tool Name	Supported Technique	Accuracy		Clone Types Detected
				Precision	Recall	
1	[41]	SourcererCC	Token-Based	83%	90%	1,2,3
2	[42]	CCFinderSW	Token-Based	N/A	N/A	1,2
3	[43]	CCLearner	Token-Based	93%	N/A	1,2,3
4	[39]	CCAlinger	Token-Based	83%	92%	1,2,3
5	[40]	Decresendo	Token-Based	N/A	N/A	1,2,3
6	[28][25]	VUDDY	Text Based	100%	82%	1,2
7	[64]	LICCA	Hybrid	N/A	N/A	1,2,3
8	[54]	CCSharp	Semantic	99.3% , 100%		4
9	[48][50]	CloneWorks	Metric Based	93%,83%	N/A	1,2,3
10	[34]	srcSlice	Text Based	N/A	N/A	N/A
11	[56]	AnDarwin	Semantic	N/A	N/A	N/A
12	[69]	CloneManager	Hybrid	97%,88% 100%,100%	95%, ,98% 95%,100%	1,2,3,4
13	[52]	Vincent	Matric Based	93%, 92%		1,2,3

and iClones) publically available state-of-the-art code clone detection tools. Results show that it can scale 100 MLOC without any issue in just 1 day 12 hours 54 minutes and 5 seconds with an overall 90% recall and 83 % precision. It has the highest scalability with the lowest execution time, second best precision and recall. SourcererCC has the ability to detect type1, type2 and type3 code clones. However, clone detection precision is very much an open problem and detection of type4 clones is outside the scope of this tool. Similarly, Semura *et al.* [42] propose CCFinderSW, a token based detector for detection of clones in multiple programming languages.

It can find type1 and type2 clones but detection of type3 and type4 clones outside the scope of this tool. Li *et al.* [43] introduce CCLearner a deep learning based approach supported by token-based approaches. It is developed as an approach but its comparison is performed against the publically available state of the art clone detection tools (NICAD, Deckard, SourcererCC). Its evaluation is performed w.r.t different aspects (e.g. precision, recall, execution time and scalability). It can scale 3.6 MLOC in just 47 minutes with 93% precision. The comparison shows that it has second-best precision, third best scalability and execution time. It can uncover type1, type2 and type3 clones. As its precision is evaluated manually therefore, precision rates may be subjected to human unintentional or bias errors. Furthermore, Wang *et al.* [39] develop another token-based clone detector, named as CCAligner. Its performance is evaluated w.r.t. to some aspects (e.g. precision, recall, execution time, scalability) and compared against popular tools (CCFinderX, Deckard, NICAD, iClones and SoursererCC). Experiment results indicate that it can scale 10 MLOC in just 24 minutes and 56 seconds with 83% precision and 92% recall. From the comparison, it is concluded that it has the lowest execution time, second best scalability, recall, and third best precision. It has the ability to detect type 1, type 2 and

TABLE 16. Comparative analysis of CCD tools w.r.t scalability, execution time and limitations.

Sr.#	Reference No	Tool Name	Scalability	Execution Time	Limitations
1	[41]	SourcererCC	100MLOC	1d 12m 54s	Precision Problems, type4 clones
2	[42]	CCFinderSW	N/A	N/A	Type 3 and Type 4
3	[43]	CCLearner	3.6MLOC	47m	Human errors
4	[39]	CCAlinger	10MLOC	24m56s	Scalability, Evaluation work
5	[40]	Decresendo	N/A	N/A	Targeted software
6	[28][25]	VUDDY	1BLOC	14h 17m	Language, Type 3, Type 4
7	[64]	LICCA	N/A	N/A	Functionally similar segments
8	[54]	CCSharp	N/A	6m42s, 0.95s. 33m1s, 14.9s	Procedure, Configuration Settings
9	[48][50]	CloneWorks	250MLOC	4h	Type 4
10	[34]	srcSlice	13000,000 LOC	7m	Language
11	[56]	AnDarwin	N/A	10h	N/A
12	[69]	CloneManager	35KLOC	1.35m	Language, high Memory, Human errors
13	[52]	Vincent	241924 LOC	5h31m	Human errors

type 3 clones. However, it has some scalability and evaluation work limitations. Yuki *et al.* [40] present a technique to detect multi-grained code clones and develop a tool Decresendo based on the proposed technique. This tool supported by token-based code clone detection approaches and can identify type1, type2 and type3 clones.

Kim *et al.* [28] and Kim and Lee [25] present VUDDY, an approach for vulnerable code clone discovery. Its comparison is performed with four (SourcererCC, CCFinderX, Deckard and ReDeBug) state of the art tools. Its scalability, execution time, recall and precision is evaluated and compared with these tools. Results indicate that it can scale 1BLOC in just 14 hours and 17 minutes with 100% precision and 82% recall. It has the highest scalability and precision, lowest execution time and second best recall. VUDDY is supported by text-based approaches and has the ability to detect type1 and type2 clones. However, its capabilities are only limited to C/C++ and detection of type3 and type4 clones outside of its scope.

Vislavski *et al.* [64] propose LICCA, a tool for cross-language clone detection. It is supported by hybrid approaches (e.g., token-based, tree-based and metric-based) and can uncover type1, type2 and type3 clones. However, currently, clone detection of LICCA is only limited to semantically similar segments while functionally similar segments are not covered yet. Wang *et al.* [54] present another tool CCSharp, a three-phase PGD based clone detection technique implemented as a tool. Its experiment is performed on PostgreSQL program and less program against three (NICAD, Deckard, SourcererCC) popular clone detection tools. Results indicate that it has 6 minutes and 42 seconds setup cost and 0.95 seconds comparison cost for less program, 33 minutes and 1-second setup cost and 14.9-second comparison cost for PostgreSQL program. As CCSharp is PDG based clone detector so we can suppose that it can detect type 4 or semantic clones. However, it has some procedural limitations as it cannot process some procedures and also have limitations in the configuration setting. Svajlenko and Roy [48] overviewed the concepts of CloneWorks, a near miss (type3) code clone detection tool. Its further details are provided by Svajlenko and Roy [50] where two configurations conservative and aggressive for detection of type3 clones are provided. Its precision is evaluated and compared with state of the art tools (iClones, NICAD, SourererCC). Results show that its precision is 93% for conservative and 83% for an aggressive configuration that is comparable with the other tools. It can detect type3 clones as large as in 250 MLOC in just 4 hours on an average workstation. It can also detect type1 and type2 clones but the detection of type4 clones is not covered.

Newman *et al.* [34] introduce a tool named srcSlice, supported by text-based approaches. Its performance is evaluated on the bases of certain parameters (e.g. execution time, scalability). From the results of the experiment, it is concluded that it can scale 13000,000 LOC of Linux kernel in just 7 minutes but currently its capabilities are limited to just C/C++. Crussell *et al.* [56] propose AnDarwin, a tool for finding applications with a similar code on large scale and supported by semantic clone detection approaches. The experiment is conducted to evaluates its performance. Results indicate that it can identify at least 4295 cloned apps and 36,106 rebranded apps in just 10 hours.

Kodhai and Kanmani [69] propose CloneManager, a tool for method level code clone detection and supported by a hybrid approach (text-based and metric-based approaches). The experiment is conducted to evaluate its performance by taking dataset from various open source systems against popular clone detection tools (NICAD and CLAN). Results shown that it has the ability to scale 35KLOC of Eclipse-ant in just 1.35 minutes and identify type1 clones with 97% precision and 95 %recall; type2 clones with 88% precision and 98% recall; type3 clones with 100% precision and 95% recall and type4 clones with 100% precision and 100% recall. However, it may have language dependency issues, utilizes slightly higher memory and based on manual analysis which may be subjected to human errors. Ragkhitwetsagul *et al.* [52] present a tool named as Vincent for image-based code clone detection. This tool supported by metric-based approaches and utilizes two similarity measures jeccard and EMD. The experiment is conducted to check out its performance. From the experiment, it is concluded that Vincent has the ability to scale 241924 LOC in 5 hours and 31 minutes. Its accuracy has been evaluated and compared with publically available tools (CCFinderX, Deckard, iClones, NICAD and Simian).

It can detect clones with 93% accuracy by using jeccard similarity measure and 92% accuracy with EMD similarity measure. Theses accuracies are comparable with other tools. It can find type1, type2 and type3 clones but its investigation is performed manually which can be subjected to human errors

From this analysis, it is concluded that each technique and tool has some strengths and limitations. Therefore, this analysis surely help the researchers in the selection of technique or tool according to their requirements.

*RQ5:*What are the key improvements required in the CCD tools / techniques in order to meet the modern technological advancements?

Answer: From the above mentioned facts, it is concluded that each technique and tool has some strengths and limitations. Therefore, some improvements can be made in future. For example, as mentioned in TABLE 14, some approaches cannot detect most difficult type of clones semantic or type 4 because these clones are functionally similar but structurally different. Although Tree based, Metric based approaches can identify type4 clones but Semantic or PDG based techniques are mainly used to identify type4 clones [64]. However, PGD generation and PGD structure modification is still a time-consuming process [54]. Therefore, there is a need to overcome this problem.

From TABLE 13, it is analyzed that mostly source code of java or C/C++ is used to identify clones. Consequently, source code of other programming languages should be target to examine the efficiency of these techniques. Moreover, it is examined that source code of open source software systems is used for CCD. Similar to open source systems, clones can also exist in commercial software systems. Therefore, in future, the commercial software systems should be target to check the validity of these approaches.

From TABLE 15 and TABLE 16, it is investigated that only one tool (CloneManager) available yet that can detect all types of clones, especially semantic or type4 clones [69]. This tool also has certain limitations like it can only deal with the C and Java code. Furthermore, this tool utilizes slightly higher memory and its verification is done by manual analysis. Therefore, it is required to develop more tools in which these limitations can be handled. Furthermore, provision of finding all types of clones especially sematic or type4 clones should be included with simplicity.

V. DISCUSSION AND LIMITATIONS

In software programs, code clone is an identical or similar segment of code. Mostly, code clones occur by copy-paste activity of code in software development. Cloning is helpful but it can also be adverse in many ways. For example, if a bug is found in one piece of code then fixing is needed in all replicated sections. Duplicated segments can also have negative impacts on software quality and software maintenance. By considering these issues, code clone detection becomes an active area of research because detecting and eliminating duplicate data or clones will improve the overall efficiency of the software, especially ease the maintenance and reuse of the components from the repositories. Therefore, in this article, we conduct an SLR to comprehensively analyze code clone detection. We selected 54 studies out of 3665 studies and provide categorization and quantitative overview. Our focus is on including the most recent research studies related to code clone detection. In this article, we analyze different code clone detection techniques and tools based on certain parameters. Our focus is broader as compared to the previous studies because we also investigate the advantages and limitations of code clone detection techniques along with tools. Furthermore, we perform detailed comparative analysis of tools and techniques that provide ease to the researchers in the selection of appropriate CCD tool or technique.

From this literature, it is analyzed that a large number of studies supports the harmfulness of code cloning in software systems. We observed that in most of the studies the detection of different types of clones is carried out. Investigation shows that there are four types (type1, type2, type3 and type4) of clones and different techniques can detect different types of clones. Text-based techniques can detect type1, type2 and type3 clones; these are most effective for the detection of type1 or exact clones [64]. Similarly, token-based techniques can identify type1, type2 and type3 clones and these are more useful for the detection of type2 or renamed clones [64]. Furthermore, tree-based and metric-based techniques can reveal type1, type2, type3 and type4 clones but they are more powerful for the identification of type3 or near miss clones [64]. Moreover, semantic approaches can find type1, type2, type3 and type4 clones but they are especially useful for the detection of type4 or semantic clones [64]. As semantic or PDG's based approaches are mainly used to identify type4 or semantic clones but PGD generation and PGD structure modification is still a time-consuming process [54]. Therefore, there is a need to overcome this problem. In hybrid approaches, detection of clone types depend on the types of techniques used as a hybrid technique. As tree based, metric based and semantic approaches can detect all types of clones. However, in only 6 studies ([49], [51], [60], [63], [67], [69]) out of 54 studies, all types of clones are detected. Therefore, this area is still open. Different types of CCD tools based on these techniques are also investigated. We found that only one tool (CloneManager) is developed that can identify four types of clones [69]. However, it has certain limitations as well, for example, source code utilized by this tool consists of only C and Java code and does not declare that the findings can be held true for other programming languages. Furthermore, the use of memory is also slightly higher and verification of tool is done by manual analysis, which means that there is a chance of human error. Therefore, there is a need to develop more tools in which these limitations can be handled and can reveal all types of clones. Furthermore, dataset or source code for detection of clones is taken from open source subject systems. Mostly, source code or dataset consists of java files, sometimes C/C++ files and rarely the files of any other programming language, as summarized in TABLE 13 . In some studies, clone detection is carried out just considering java code and their capabilities are limited to java code. Therefore, there is a need to utilize source code of other programming languages for code clone detection. From literature, it is analyzed that only open source software systems are targeted for CCD. Similar to open source systems, clones can also be present in commercial software systems. Consequently, it is required to extend this area on commercial scale.

Although we have completely followed the guidelines of SLR [21], and strictly observed the review protocol. However, there are still certain limitations:

- We have selected four well-known scientific repositories i.e., ACM, Springer, IEEE, Elsevier for this SLR. These repositories provide a huge amount of conference and journal papers. Therefore, we performed the search process by utilizing only these repositories. However, there may exist some amount of relevant stuff in other repositories. Accordingly, there is a chance we missed some related researches from other repositories. However, we consider that exclusion of such repositories does not significantly affect the outcome of this SLR because selected repositories provide good quality recent research literature.
- We have utilized relevant keywords and thoroughly checked the search results. However, few keywords returned a large number of results and cannot thoroughly analyzed. Moreover, we have rejected many research studies based on their title or abstract. Therefore, there is a chance we missed some relevant research studies.
- We have selected English language for selecting research studies. However, there is a chance we may miss some relevant researches written in other languages e.g., French, Germen etc.

VI. CONCLUSIONS AND FUTURE WORK

This research study comprehensively analyzes how code clones can be detected and which techniques and tools are utilized for this purpose. Particularly, an SLR is performed to identify and investigate 54 research articles, published during 2013-2018, in the domain of code clone detection. Six categories are defined to incorporate selected studies i.e., textual approaches (12), lexical approaches (8), tree-based approaches (3), metric-based approaches (7), semantic approaches (7) and hybrid approaches (17). Afterwards, each category is comprehensively investigated to understand how code clones can be detected and which technique detect which type of clones. Moreover, 13 existing tools and 13 proposed tools are presented. Furthermore, 62 open source subject systems are investigated. Consequently, the leading approaches, important tools and open source subject systems utilized for code clone detection are provided in a single study. This will help the researchers and practitioners to select appropriate approach or tool for code clone detection according to their requirements.

It is concluded from this SLR that the detection of type 4 clones is complex process. For this reason, very few studies are dealing with the detection of type 4 clones with certain limitations. Therefore, a main future direction is to develop a novel technique and tool to successfully detect type 4 clones with simplicity. In this regard, the findings of this article are highly beneficial. Additionally, existing studies mostly consider java and C/C++ languages for code clone detection. In this regard, it is essential to take in the other programming languages like C# etc. in the area of CCD in order to meet the real and current demands of software industry.

REFERENCES

[1] B. S. Baker, "On finding duplication and near-duplication in large software systems," in *Proc. 2nd Work. Conf. Reverse Eng.*, Jul. 1995, pp. 86–95.

[2] S. Ducasse, M. Rieger, and S. Demeyer, "A language independent approach for detecting duplicated code," in *Proc. IEEE Int. Conf. Softw. Maintenance (ICSM)*, Aug./Sep. 1999, pp. 109–118.

[3] J. Krinke, "Identifying similar code with program dependence graphs," in *Proc. 8th Work. Conf. Reverse Eng.*, Oct. 2001, pp. 301–309.

[4] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in *Proc. Int. Conf. Softw. Maintenance*, Nov. 1998, pp. 368–377.

[5] R. Komondoor and S. Horwitz, "Using slicing to identify duplication in source code," in *Proc. Int. Static Anal. Symp.* Berlin, Germany: Springer, 2001, pp. 40–56.

[6] C. K. Roy and J. R. Cordy, "A survey on software clone detection research," Queen's School Comput., Tech. Rep. 541, 2007, vol. 115, pp. 64–68.

[7] C. K. Roy, R. J. 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.

[8] N. Saini and S. Singh, "Code clones: Detection and management," *Procedia Comput. Sci.*, vol. 132, pp. 718–727, Jun. 2018.

[9] C. K. Roy and J. R. Cordy, "An empirical study of function clones in open source software," in *Proc. 15th Work. Conf. Reverse Eng.*, Oct. 2008, pp. 81–90.

[10] C. K. Roy and J. R. Cordy, "NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization," in *Proc. 16th IEEE Int. Conf. Program Comprehension*, Jun. 2008, pp. 172–181.

[11] T. Kamiya. *The Official CCFinderX WebSite*. Accessed: 2008. [Online]. Available: http://www.ccfinder.net/ccfinderx.html

[12] Y. Higo, Y. Ueda, T. Kamiya, S. Kusumoto, and K. Inoue, "On software maintenance process improvement based on code clone analysis," in *Proc. Int. Conf. Product Focused Softw. Process Improvement*. Berlin, Germany: Springer, 2002, pp. 185–197.

[13] *Tool Simian*. Accessed: Nov. 2008. [Online]. Available: http://www.redhillconsulting.com.au/products/simian/

[14] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "CP-Miner: Finding copy-paste and related bugs in large-scale software code," *IEEE Trans. Softw. Eng.*, vol. 32, no. 3, pp. 176–192, Mar. 2006.

[15] J. H. Johnson, "Visualizing textual redundancy in legacy source," in *Proc. Conf. Centre Adv. Stud. Collaborative Res.* Indianapolis, IN, USA: IBM Press, 1994, p. 32.

[16] J. H. Johnson, "Substring matching for clone detection and change tracking," in *Proc. ICSM*, vol. 94, 1994, pp. 120–126.

[17] N. Davey, P. Barson, S. Field, R. Frank, and D. Tansley, "The development of a software clone detector," in *International Journal of Applied Software Technology*. London, U.K.: University of Hertfordshire Research Archive, 1995.

[18] G. A. Di Lucca, M. Di Penta, and A. R. Fasolino, "An approach to identify duplicated Web pages," in *Proc. 26th Annu. Int. Comput. Softw. Appl.*, Aug. 2002, pp. 481–486.

[19] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *J. Syst. Softw.*, vol. 80, no. 4, pp. 571–583, 2007.

[20] D. Budgen and P. Brereton, "Performing systematic literature reviews in software engineering," in *Proc. 28th Int. Conf. Softw. Eng.*, 2006, pp. 1051–1052.

[21] S. Charters and B. Kitchenham, "Guidelines for performing systematic literature reviews in software engineering," Keele Univ. Durham Univ. Joint Rep., Newcastle, U.K., Tech. Rep. EBSE-2007-01, Version 2.3, 2007.

[22] R. Koschke, "Survey of research on software clones," in *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl, Germany: Informatik, 2007.

[23] D. Rattan, R. Bhatia, and M. Singh, "Software clone detection: A systematic review," *Inf. Softw. Technol.*, vol. 55, no. 7, pp. 1165–1199, 2013.

[24] C. Ragkhitwetsagul and J. Krinke, "Using compilation/decompilation to enhance clone detection," in *Proc. IEEE 11th Int. Workshop Softw. Clone (IWSC)*, vol. 11, Feb. 2017, pp. 8–14.

[25] S. Kim and H. Lee, "Software systems at risk: An empirical study of cloned vulnerabilities in practice," *Comput. Secur.*, vol. 77, pp. 720–736, Aug. 2018.

[26] S. Jadon, "Code clones detection using machine learning technique: Support vector machine," in *Proc. Int. Conf. Comput., Commun. Automat. (ICCCA)*, Apr. 2016, pp. 303–399.

[27] D. Yu, J. Wang, Q. Wu, J. Yang, J. Wang, W. Yang, and W. Yan, "Detecting java code clones with multi-granularities based on bytecode," in *Proc. IEEE 41st Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jul. 2017, pp. 317–326.

[28] S. Kim, S. Woo, H. Lee, and H. Oh, "VUDDY: A scalable approach for vulnerable code clone discovery," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 595–614.

[29] Y. Nakamura, E. Choi, N. Yoshida, S. Haruna, and K. Inoue, "Towards detection and analysis of interlanguage clones for multilingual Web applications," in *Proc. IEEE 23rd Int. Conf. Softw. Analysis, Evol., Reeng. (SANER)*, vol. 3, Mar. 2016, pp. 17–18.

[30] F. Lyu, Y. Lin, J. Yang, and J. Zhou, "Suidroid: An efficient hardening-resilient approach to Android app clone detection," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2016, pp. 511–518.

[31] H. Xue, G. Venkataramani, and T. Lan, "Clone-hunter: Accelerated bound checks elimination via binary code clone detection," in *Proc. 2nd ACM SIGPLAN Int. Workshop Mach. Learn. Program. Lang.*, 2018, pp. 11–19.

[32] J. Chen, M. H. Alalfi, T. R. Dean, and Y. Zou, "Detecting Android malware using clone detection," *J. Comput. Sci. Technol.*, vol. 30, no. 5, pp. 942–956, 2015.

[33] H. Thaller, R. Ramler, J. Pichler, A. Egyed, "Exploring code clones in programmable logic controller software," Jun. 2017, *arXiv:1706.03934*. [Online]. Available: https://arxiv.org/abs/1706.03934#

[34] C. D. Newman, T. Sage, M. L. Collard, H. W. Alomari, and J. I. Maletic, "srcSlice: A tool for efficient static forward slicing," in *Proc. IEEE/ACM 38th Int. Conf. Softw. Eng. Companion (ICSE-C)*, May 2016, pp. 621–624.

[35] Z. Liu, Q. Wei, and Y. Cao, "VFDETECT: A vulnerable code clone detection system based on vulnerability fingerprint," in *Proc. IEEE 3rd Inf. Technol. Mechatron. Eng. Conf. (ITOEC)*, Oct. 2017, pp. 548–553.

[36] M. A. Nishi and K. Damevski, "Scalable code clone detection and search based on adaptive prefix filtering," *J. Syst. Softw.*, vol. 137, pp. 130–142, Mar. 2018.

[37] R. Tekchandani, R. Bhatia, and M. Singh, "Code clone genealogy detection on e-health system using Hadoop," *Comput. Elect. Eng.*, vol. 61, pp. 15–30, Jul. 2017.

[38] M. R. Farhadi, B. C. M. Fung, Y. B. Fung, P. Charland, S. Preda, and M. Debbabi, "Scalable code clone search for malware analysis," *Digit. Invest.*, vol. 15, pp. 46–60, Dec. 2015.

[39] P. Wang, J. Svajlenko, Y. Wu, Y. Xu, and C. K. Roy, "CCAligner: A token based large-gap clone detector," in *Proc. 40th Int. Conf. Softw. Eng.*, 2018, pp. 1066–1077.

[40] Y. Yuki, Y. Higo, and S. Kusumoto, "A technique to detect multi-grained code clones," in *Proc. IEEE 11th Int. Workshop Softw. Clones (IWSC)*, Feb. 2017, pp. 1–7.

[41] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "SourcererCC: Scaling code clone detection to big-code," in *Proc. IEEE/ACM 38th Int. Conf. Softw. Eng. (ICSE)*, May 2016, pp. 1157–1168.

[42] Y. Semura, N. Yoshida, E. Choi, and K. Inoue, "CCFinderSW: Clone detection tool with flexible multilingual tokenization," in *Proc. 24th Asia–Pacific Softw. Eng. Conf. (APSEC)*, 2017, pp. 654–659.

[43] L. Li, H. Feng, W. Zhuang, N. Meng, and B. Ryder, "Cclearner: A deep learning-based clone detection approach," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2017, pp. 249–260.

[44] Y. Yang, Z. Ren, X. Chen, and H. Jiang, "Structural function based code clone detection using a new hybrid technique," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jul. 2018, pp. 286–291.

[45] J. Pati, B. Kumar, D. Manjhi, and K. K. Shukla, "A Comparison Among ARIMA, BP-NN, and MOGA-NN for Software Clone Evolution Prediction," *IEEE Access*, vol. 5, pp. 11841–11851, 2017.

[46] S. Chodarev, E. Pietriková, and J. Kollár, "Haskell clone detection using pattern comparing algorithm," in *Proc. 13th Int. Conf. Eng. Mod. Electr. Syst. (EMES)*, Jun. 2015, pp. 1–4.

[47] M. Tsunoda, Y. Kamei, and A. Sawada, "Assessing the differences of clone detection methods used in the fault-prone module prediction," in *Proc. IEEE 23rd Int. Conf. Softw. Anal., Evol., Reeng. (SANER)*, vol. 3, Mar. 2016, pp. 15–16.

[48] J. Svajlenko, C. K. Roy, "Fast and flexible large-scale clone detection with cloneworks," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. Companion (ICSE-C)*, May 2017, pp. 27–30.

[49] M. Sudhamani and L. Rangarajan, "Code clone detection based on order and content of control statements," in *Proc. 2nd Int. Conf. Contemp. Comput. Inform. (IC3I)*, 2016, pp. 59–64.

[50] M. Sudhamani and L. Rangarajan, "Cloneworks: A fast and flexible large-scale near-miss clone detection tool," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. Companion (ICSE-C)*, May 2017, pp. 177–179.

[51] S. M. F. Haque, V. Srikanth, and E. S. Reddy, "Generic code cloning method for detection of clone code in software development," in *Proc. Int. Conf. Data Mining Adv. Comput. (SAPIENCE)*, Mar. 2016, pp. 335–339.

[52] C. Ragkhitwetsagul, J. Krinke, and B. Marnette, "A picture is worth a thousand words: Code clone detection based on image similarity," in *Proc. IEEE 12th Int. Workshop Softw. Clones (IWSC)*, Mar. 2018, pp. 44–50.

[53] M. Sudhamani and L. Rangarajan, "Structural similarity detection using structure of control statements," *Procedia Comput. Sci.*, vol. 46, pp. 892–899, Apr. 2015.
[54] M. Wang, P. Wang, and Y. Xu, "CCSharp: An efficient three-phase code clone detector using modified PDGs," in *Proc. 24th Asia–Pacific Softw. Eng. Conf. (APSEC)*, 2017, pp. 100–109.
[55] Y. Sabi, Y. Higo, and S. Kusumoto, "Rearranging the order of program statements for code clone detection," in *Proc. IEEE 11th Int. Workshop Softw. Clones (IWSC)*, Feb. 2017, pp. 1–7.
[56] J. Crussell, C. Gibler, and H. Chen, "Andarwin: Scalable detection of Android application clones based on semantics," *IEEE Trans. Mobile Comput.*, vol. 14, no. 10, pp. 2007–2019, Oct. 2015.
[57] S. Sargsyan, S. Kurmangaleev, A. Belevantsev, and A. Avetisyan, "Scalable and accurate detection of code clones," *Program. Comput. Softw.*, vol. 42, no. 1, pp. 27–33, 2016.
[58] Y. Hu, Y. Zhang, J. Li, and D. Gu, "Binary code clone detection across architectures and compiling configurations," in *Proc. 25th Int. Conf. Program Comprehension*, 2017, pp. 88–98.
[59] C. M. Kamalpriya and P. Singh, "Enhancing program dependency graph based clone detection using approximate subgraph matching," in *Proc. IEEE 11th Int. Workshop Softw. Clones (IWSC)*, Feb. 2017, pp. 1–7.
[60] A. Avetisyan, S. Kurmangaleev, S. Sargsyan, M. Arutunian, and A. Belevantsev, "LLVM-based code clone detection framework," in *Proc. Comput. Sci. Inf. Technol. (CSIT)*, 2015, pp. 100–104.
[61] Roopam and G. Singh, "To enhance the code clone detection algorithm by using hybrid approach for detection of code clones," in *Proc. Int. Conf. Intell. Comput. Control Syst. (ICICCS)*, Jun. 2017, pp. 192–198. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8250708
[62] M. R. H. Misu and K. Sakib, "Interface driven code clone detection," in *Proc. 24th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2017, pp. 747–748.
[63] A. Sheneamer and J. Kalita, "Semantic clone detection using machine learning," in *Proc. 15th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2016, pp. 1024–1028.
[64] T. Vislavski, G. Rakić, N. Cardozo, and Z. Budimac, "LICCA: A tool for cross-language clone detection," in *Proc. IEEE 25th Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Mar. 2018, pp. 512–516.
[65] M. R. H. Misu, A. Satter, and K. Sakib, "An exploratory study on interface similarities in code clones," in *Proc. 24th Asia–Pacific Softw. Eng. Conf. Workshops (APSECW)*, Dec. 2017, pp. 126–133.
[66] J. Akram, Z. Shi, M. Mumtaz, and P. Luo, "DroidCC: A scalable clone detection approach for Android applications to detect similarity at source code level," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jul. 2018, pp. 100–105.
[67] A. Sheneamer, S. Roy, and J. Kalita, "A detection framework for semantic code clones and obfuscated code," *Expert Syst. Appl.*, vol. 97, pp. 405–420, May 2018.
[68] T. Matsushita and I. Sasano, "Detecting code clones with gaps by function applications," in *Proc. ACM SIGPLAN Workshop Partial Eval. Program Manipulation*, 2017, pp. 12–22.
[69] E. Kodhai and S. Kanmani, "Method-level code clone detection through LWH (Light Weight Hybrid) approach," *J. Softw. Eng. Res. Develop.*, vol. 2, no. 1, p. 12, 2014.
[70] R. Tekchandani, R. Bhatia, and M. Singh, "Semantic code clone detection for Internet of things applications using reaching definition and liveness analysis," *J. Supercomput.*, vol. 74, no. 9, pp. 4199–4226, 2018.
[71] K. Uemura, A. Mori, K. Fujiwara, E. Choi, and H. Iida, "Detecting and analyzing code clones in HDL," in *Proc. IEEE 11th Int. Workshop Softw. Clones (IWSC)*, Feb. 2017, pp. 1–7.
[72] H. Nasirloo and F. Azimzadeh, "Semantic code clone detection using abstract memory states and program dependency graphs," in *Proc. 4th Int. Conf. Web Res. (ICWR)*, Apr. 2018, pp. 19–27.
[73] M. Singh and V. Sharma, "Detection of file level clone for high level cloning," *Procedia Comput. Sci.*, vol. 57, pp. 915–922, Aug. 2015.
[74] A. Sheneamer and J. Kalita, "Code clone detection using coarse and fine-grained hybrid approaches," in *Proc. IEEE 7th Int. Conf. Intell. Comput. Inf. Syst. (ICICIS)*, Dec. 2015, pp. 472–480.
[75] M. White, M. Tufano, C. Vendome, and D. Poshyvanyk, "Deep learning code fragments for code clone detection," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2016, pp. 87–98.
[76] C. Ragkhitwetsagul J. Krinke, and D. Clark, "A comparison of code similarity analysers," *Empirical Softw. Eng.*, vol. 23, no. 4, pp. 2464–2519, 2018.
[77] J. Ghofrani, M. Mohseni, and A. Bozorgmehr, "A conceptual framework for clone detection using machine learning," in *Proc. IEEE 4th Int. Conf. Knowl.-Based Eng. Innov. (KBEI)*, Dec. 2017, pp. 810–817.

QURAT UL AIN received the B.S. degree in software engineering from the Government College University Faisalabad, Pakistan. She is currently pursuing the M.S. degree in software engineering with the Computer and Software Engineering Department, College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan. Her area of research is software engineering.

WASI HAIDER BUTT is currently an Assistant Professor with the Department of Computer and Software Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan. His areas of interests include model-driven software engineering, web development, and requirement engineering.

MUHAMMAD WASEEM ANWAR is currently pursuing the Ph.D. degree with the Department of Computer and Software Engineering, CEME, National University of Sciences and Technology, Pakistan. He is a Senior Researcher and an Industry Practitioner in the field of model-based system engineering (MBSE) for embedded and control systems. His major research area includes MBSE for complex and large systems. His profile can be viewed at http://ceme.nust.edu.pk/ISEGROUP/seniormembers/waseem.html.

FAROOQUE AZAM is currently an Adjunct Faculty with the Department of Computer and Software Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan. He has been teaching various software engineering courses, since 2007. His areas of interests include model-driven software engineering, business modeling for Web applications, and business process reengineering.

BILAL MAQBOOL received the M.S. degree in software engineering from the Computer and Software Engineering Department, College of Electrical and Mechanical Engineering, National University of Sciences and Technology (NUST), Pakistan, in 2018. From 2017 to 2018, he was a Research Assistant with NUST, where he is currently a Senior Researcher in the field of software engineering. His area of research is business process automation through model-driven software engineering (MDSE) and natural language processing (NLP).