Computer Science Review 47 (2023) 100528

Contents lists available at ScienceDirect

Computer Science Review

journal homepage: www.elsevier.com/locate/cosrev

Review article

A systematic literature review on the use of machine learning in code clone research

Manpreet Kaur *, Dhavleesh Rattan

Department of Computer Science and Engineering, Punjabi University Patiala, 147002, Punjab, India

ARTICLE INFO

Article history:
Received 1 March 2021
Received in revised form 25 July 2022
Accepted 24 November 2022
Available online 20 December 2022

Keywords:
Software clone
Code clone
Machine learning
Deep learning
Semantic clone detection
Systematic literature review

ABSTRACT

Context: Research related to code clones includes detection of clones in software systems, analysis, visualization and management of clones. Detection of semantic clones and management of clones have attracted use of machine learning techniques in code clone related research.
Objective: The aim of this study is to report the extent of machine learning usage in code clone related research areas.
Method: The paper uses a systematic review method to report the use of machine learning in research related to code clones. The study considers a comprehensive set of 57 articles published in leading conferences, workshops and journals.
Results: Code clone related research using machine learning techniques is classified into different categories. Machine learning and deep learning algorithms used in the code clone research are reported. The datasets, features used to train machine learning models and metrics used to evaluate machine learning algorithms are reported. The comparative results of various machine learning algorithms presented in primary studies are reported.
Conclusion: The research will help to identify the status of using machine learning in different code clone related research areas. We identify the need of more empirical studies to assess the benefits of machine learning in code clone research and give recommendations for future research.

Contents

1. Introduction 2
   1.1. Motivation for work 2
2. Related research 2
   2.1. Contributions 3
3. Research method 3
   3.1. Research questions 3
   3.2. Search strategy 3
      3.2.1. Search string 3
      3.2.2. Sources searched 4
   3.3. Selection process 4
      3.3.1. Inclusion and exclusion criteria 4
   3.4. Quality assessment 5
   3.5. Data extraction 5
4. Results 6
   4.1. RQ1: What are the different areas of code clone research in which machine learning techniques have been applied? 6
      4.1.1. Code clone detection 6
      4.1.2. Semantic clone detection 7
      4.1.3. Binary code clone detection 8
      4.1.4. Cross language code clone detection 9
      4.1.5. Source code representation for clone detection 9
      4.1.6. Code clone classification/validation 9

* Corresponding author.
E-mail addresses: manpreet.kaur09@gmail.com (M. Kaur), dhavleesh@gmail.com (D. Rattan).

https://doi.org/10.1016/j.cosrev.2022.100528
1574-0137/

4.1.7. Clone recommendation for refactoring ... 9
4.1.8. Clone evolution ... 10
4.1.9. Evaluating clone detection precision ... 10
4.1.10. Prediction of harmfulness of cloning operations ... 10
4.1.11. Prediction of code clone quality ... 11
4.1.12. Detection of bugs in clones ... 11
4.1.13. Prediction of risky clones ... 11
4.2. RQ2: What machine learning algorithms have been used in different primary studies? ... 11
4.3. RQ3: What features of code have been used to develop machine learning based models? ... 17
4.4. RQ4: What are evaluation metrics used to measure the performance of machine learning based models? ... 18
4.5. RQ5: Which datasets have been used in the primary studies? ... 20
4.6. RQ6: Which machine learning techniques have been used in semantic clone detection? ... 23
4.7. RQ7 Which machine learning algorithm gives better results when compared using the same dataset for the same problem? ... 24
5. Discussion ... 28
6. Gaps and limitations of current research ... 29
7. Threats to validity ... 30
8. Conclusion ... 30
Declaration of competing interest ... 30
Acknowledgement ... 30
Appendix A. Acronyms ... 31
Appendix B. Quality assessment score ... 31
References ... 31

1. Introduction

Over the past 25 years, the detection of clones in software systems has gained attention of many researchers. Code clones are copies of existing source code with or without modifications that exist in the same source code file or in different files [1]. Existence of clones in source code can be a matter of concern during software maintenance. The presence of clones in source code affects the bug removal and refactoring related tasks. If a bug is removed from one clone instance, it is necessary to remove the same bug in other instances of the clone [2]. Otherwise, it will lead to an increase in maintenance work in future due to inconsistent changes. Similarly, if a clone is being removed using refactoring, the information of presence of other similar instances of the clone, can help to remove clones from other locations of source code also. So, a developer can perform software maintenance related tasks in a better way if he/she knows the presence of code clones in the software [3].

The code clone related research can be divided into different categories as shown in Fig. 1. A number of techniques and tools [4–7] have been developed so far in each area of code clone related research. The existing survey related to code clone research [1,8,9] presented a detailed review of research in each area along with the basics of code clones. The recent research in code clones has attracted the use of machine learning techniques in different code clone research areas. The current study aims to identify the usage of machine learning techniques in code clone research. To conduct this study, we follow the procedure given by [10–13] to identify, evaluate and interpret all the relevant existing literature.

1.1. Motivation for work

Machine learning and especially deep learning techniques have gained popularity in recent years in different fields of research like natural language processing, image processing and speech recognition. The research in code clones have benefitted from these recent developments in machine learning techniques. Machine learning is being applied in semantic clone detection, validation of clone detection results and recommendation of clones for maintenance related tasks. This recent surge in use of machine learning techniques in code clone research has motivated us to conduct systematic literature review. The aim of the review is to understand and report current status of research in the field along with gaps and future challenges so that directions in future research can be reported.

The remainder of the article is organized as follows: Section 2 mentions about the related surveys conducted in code clone research. Section 3 describes the research method used to conduct the review. Section 4 reports the results of the systematic literature review. Section 5 discusses the major findings and provides recommendations for future work whereas Section 6 summarizes gaps and limitations of current research. Section 7 discusses threats to the literature review and Section 8 presents the conclusion of the study. Glossary of acronyms used in this paper is listed in Appendix A.

2. Related research

To the best of our knowledge, this is the first systematic literature review of machine learning techniques usage in code clone research. It evaluates all relevant articles available in different journals, workshops and conferences proceedings in terms of defined research questions to investigate the current state of research in the field. Our review is different from previous review studies because its main focus is to report the extent of application of machine learning techniques in code clone research.

Roy and Cordy [8] and Koschke [9] discussed the definition of clones and its types, reasons, advantages and disadvantages of clones. They also discussed the clone detection process, techniques, tools, clone evolution, clone analysis, clone visualization, clone management and applications of clone detection. Pate et al. [14] presented systematic review on clone evolution.

The systematic literature review of Rattan et al. [1] in code clone research reported various clone detection techniques and tools including status of research in semantic and model clone detection. The review identified various sub-areas of clone detection related research and reported status of clone management.

Roy et al. [15] presented a survey on clone management. This survey elaborated about various clone management activities like integrated clone detection, clone documentation, visualization, clone tracking, clone evolution and clone refactoring.

Mondal et al. [16] presented a survey on clone refactoring and tracking. They categorized the studies related to clone refactoring and tracking into various research areas. The survey provided qualitative analysis of clone refactoring and tracking tools and proposed future research directions in the field of clone refactoring and tracking.

Our survey is mainly focused on use of machine learning in code clone related research.

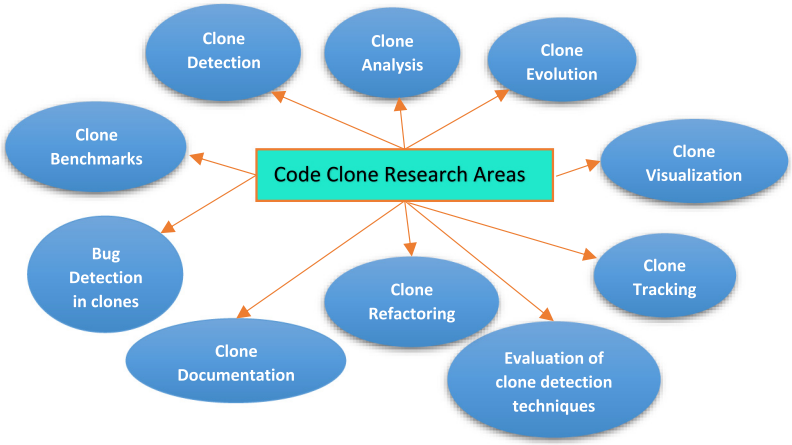

Fig. 1. Code clone research areas.

2.1. Contributions

Our SLR has the following contributions:

1. It identifies and categorizes the existing studies related to use of machine learning techniques in different code clone research areas. The research community can use them to know the current status of machine learning techniques usage in these areas of code clone research.
2. Identify the dataset and clone features used so far in training of machine learning models used in code clone research.
3. It presents machine and deep learning algorithms used so far along with evaluation metrics used to evaluate the performance of these algorithms.
4. Comparative study of various machine learning algorithms performed in various clone research areas is presented.
5. It provides recommendations based on our findings for future research.

3. Research method

The systematic literature review(SLR) was conducted following the guidelines given by [17,18]. These guidelines specify the systematic way of identifying, analysing and evaluating the existing published articles to get the answers of specific research questions. Fig. 2 elaborates the steps of systematic literature review. The need for systematic review is presented in Section 1.1. The rest of the steps followed are discussed below:

3.1. Research questions

The aim of this study is to investigate use of machine learning in various areas of code clone research. To plan the review, various research questions were framed. Table 1 describes the various research questions and the motivation behind framing the questions.

3.2. Search strategy

We framed a search strategy to collect all relevant articles required to give answers of research questions of the review. The search strategy is comprised of identification of search string and digital sources to be searched. The search string is used to search all relevant papers from different search repositories to prepare a set of all available relevant articles required for the topic of our review.

Table 1
Research questions for our systematic literature review.

Research Question	Motivation
RQ1: What are the different areas of code clone research in which machine learning techniques have been applied?	To explore the current status and future directions of machine learning techniques usage in different areas of code clone research.
RQ2: Which machine learning algorithms have been used in different primary studies?	To know the type of machine learning algorithms used in different areas of code clone research till date.
RQ2.1: What are the training strategies employed in different studies?	To explore which type of training used along with machine learning algorithms in various code clone research areas to exhibit best results.
RQ2.2: What are the validation techniques used?	To identify various validation strategies used to recognize the best strategies to gain high accuracy.
RQ3: What features of code have been used to develop machine learning based models?	To identify the popular code features that can be used to train the models.
RQ4: What are evaluation metrics used to measure the performance of machine learning based models?	To identify metrics used for evaluation of machine learning models.
RQ5: Which datasets have been used in the primary studies?	To recognize the type of subject systems considered in research conducted on code clones using machine learning.
RQ5.1: What are various existing clone detection tools used in primary studies?	To know which existing clone detection tools are popular/used in code clone studies where machine learning was applied and what is the purpose of their use.
RQ6: Which machine learning techniques have been used in semantic clone detection?	To know which machine learning algorithms have potential to detect semantic clones.
RQ7: Which machine learning algorithm gives better results when compared using the same dataset for the same problem?	To report the performance of various machine learning algorithms used in a particular area related to code clones.

3.2.1. Search string

A query string is required to extract suitable articles which can answer the research questions from different search resources. Firstly, we identified the major search terms from research questions. Secondly, synonymous and alternate terms of major search terms were listed and then, combined with each major search

Fig. 2. Steps of systematic literature review.

term using OR operator and lastly all major search terms are combined using AND operator. The result of all these steps generates the following query string that has been searched in different databases.

("clone" OR "code clone" OR "software clone" OR "code duplication") AND ("machine learning" OR "supervised Learning" OR "unsupervised learning" OR "classification" OR "regression" OR "classifier" OR "prediction") AND ("detection" OR "tracking" OR "refactoring" OR "evolution" OR "recommend" OR "analysis" OR "management")

Due to search term limitations of certain databases following short query is also defined:

("code clone" OR "clone") AND ("classifier" OR "training" OR "learning" OR "regression" OR "classification" OR "prediction") AND ("detection" OR "tracking" OR "refactoring" OR "evolution" OR "recommend" OR "analysis" OR "management")

3.2.2. Sources searched

We need to select proper sources to obtain all the available relevant articles related to research questions. Here, we selected those databases which are popular among researchers to conduct survey and systematic literature review and those search sources are listed below:

- ACM Digital Library (www.acm.org/dl)
- IEEE Xplore (ieeexplore.ieee.org)
- ScienceDirect (www.sciencedirect.com)
- Springer (www.springerlink.com)
- arXiv by Cornell University (www.arxiv.org)
- Google Scholar (www.scholar.google.co.in)
- Wiley online library (https://onlinelibrary.wiley.com/)
- Taylor and Francis (https://www.tandfonline.com/search/advanced)
- World Scientific (https://www.worldscientific.com/search/advanced)
- Inderscience (https://www.inderscienceonline.com/action/doSearch)

3.3. Selection process

Selection process used to implement the review is depicted in Fig. 3. The relevant literature used in the systematic literature review are first searched in the resources mentioned in Section 3.2.2 using the search string mentioned in Section 3.2.1. The review included the studies published up to May 2020 from the initial date of the digital library. The initial search in different data resources generates many search articles which are reported in Table 2. We found 2996 articles in respect to the search string from all sources listed in Table 2. There were many irrelevant articles which were filtered after reading the title of the paper. If the title of the article contains the word "code clone" or "clone" or "duplicate", then, the abstract was read to know whether the article applied any machine learning technique or not. The process helps to identify papers which belong to code clones and applied machine learning. This initial selection step, after applying exclusion criteria, generates 83 articles. These 83 articles were selected for the second stage of selection process and inclusion criteria was applied that led to the final selection of 57 articles. We found 11,531 articles from Taylor and Francis and 237 articles from Inderscience. But we excluded these articles because none of the articles passed initial selection. We also performed snowballing to search for any missing paper. We checked the references of finally selected 57 papers but we are not able to find any relevant paper related to the study in addition to the already selected 57 articles.

3.3.1. Inclusion and exclusion criteria

Inclusion and exclusion criteria define when an article should be selected for extracting relevant data and under what circumstances an article can be rejected for inclusion in the literature review.

Inclusion Criteria

1. All articles that have used any machine learning technique and belong to the code clone research area.

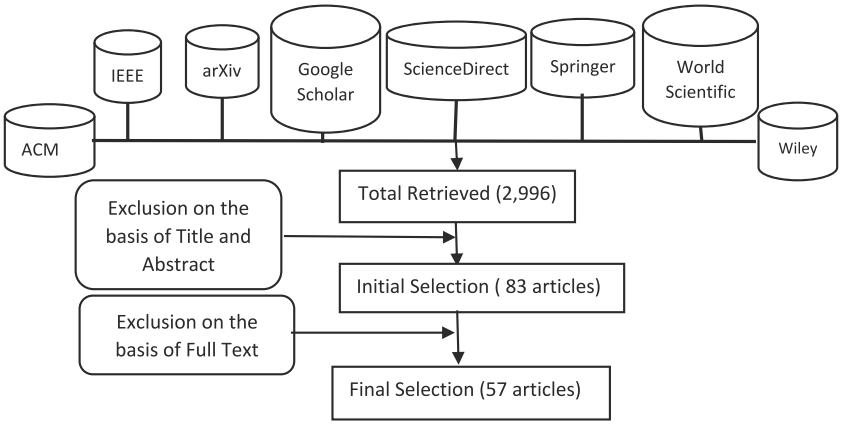

Fig. 3. Article selection process.

Table 2
Search results from different data sources.

Resource name	Search string in	Total results found	Initial selection	Final selection
ACM Digital Library	Abstract	73	19	16
IEEE Xplore Digital Library	Abstract	164	36	28
arXiv by Cornell University	Abstract	43	6	3
Google Scholar	Title, Exact phrase: search string	44	7	3
ScienceDirect Digital Library	Abstract: search string Title: Clone	2028	4	2
SpringerLink Digital Library	Exact phrase: search string, Title: clones OR code clones	504	9	4
World Scientific	Abstract	2	1	1
Wiley online library	Abstract	138	1	0
Total		2996	83	57

2. All articles written in English.
3. Articles published in journals, conferences and workshops.
4. Technical reports and position papers which give important research insights related to machine learning are included.

Exclusion Criteria

1. Any article that belongs to applications of clone detection like plagiarism detection, resource requirement prediction or malware detection. The review does not exclude any article that applies clone detection and presents machine learning approach.
2. Any article that does not use machine learning but belongs to the code clone research area.
3. Any article that is not written in English.
4. If an article is published in conference as well as in a journal then, an extended version of the article was selected.
5. If an article is repeating in different resources, then, duplicate was excluded.

3.4. Quality assessment

The quality of articles was checked after applying inclusion/exclusion criteria. Quality step helps to find all the final articles that provide information required to answer the research questions. The quality of an article was measured using the questions listed in Table 3. All the finally selected articles passed the quality assessment step and these are used for data extraction reported in Section 3.5.

The answers to these questions are reported as "yes", "no" or "partial". Here "Yes" represents score 1, "No" represents score "0" and "Partial" represents score 0.5. Using these scores, a rank is assigned to a study as follows:

Table 3
Screening questions for quality assessment.

Screening Questions
Q1: Does the study belong to code clones?
Q2: Does the study apply any machine learning algorithm?
Q3: Does the study specify the features of source code used in applying machine learning?
Q4: Does the study specify the training strategies used?
Q5: Does the study report validation methods and evaluation metrics?
Q6: Does the study clearly specify about the used dataset?
Q7: Does the study belong to semantic clone detection?
A) If yes, does the study clearly mention the machine learning technique used?
Q8: Does the study compare different machine learning algorithms?

Excellent: $6 \leq \text{score} \leq 8$
Good: $2 < \text{score} < 6$
Poor: $\text{score} \leq 2$

We selected 57 studies as final primary studies which are assigned rank "Excellent" and "Good". The result of quality assessment for each primary study is presented in Appendix B.

3.5. Data extraction

After the selection of final primary studies that have passed the quality assessment criteria, the data is extracted from these studies so that answers to research questions can be derived. Table 4 shows the data extraction form used for extracting data from the primary studies. The first author extracted the data from all the primary studies. The sample of extracted data was cross-checked by the second author of the paper. If there was any

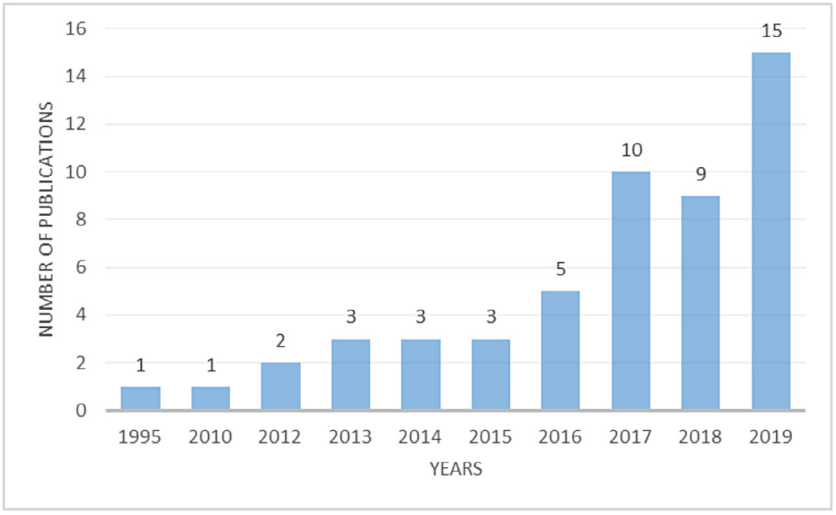

Fig. 4. Number of primary studies published in different years.

Table 4
Data extraction form.

Data item extracted	Description
Bibliographic data Type of article	Title, Author, Year, Source Journal, Workshop paper, conference article
Domain of study	Which code clone research area is the focus of the study?
Source code features	What features of source code are used for applying a machine learning technique?
Machine learning algorithm used	What type of machine learning algorithm has been used in the study?
Training strategy	Which strategy is used for training the model?
Validation technique	What validation strategy has been used to validate the model?
Evaluation metrics	What type of metrics have been used to evaluate the performance of the machine learning algorithm?
Dataset	What dataset has been used by the study for evaluating machine learning techniques?
Clone detection tool	Which existing clone detection tool has been used in the study for clone detection or for comparing the proposed approach of the study?
Comparative study	Which machine algorithms are compared in the study?
Limitations of the study	What limitations of the proposed approach are mentioned by the study?

disagreement, consensus meetings were used to resolve them. Limitations of a study were reviewed by both authors so that future research guidelines can be derived.

4. Results

This section provides answers to the research questions framed in the previous section. To answer these questions, 57 primary studies were selected. Out of these primary studies, 15 were published in journals and 42 were published in conferences and workshops. The number of research publications related to SLR in different years is shown in Fig. 4. Clearly, 77% of the primary studies belong to the year 2015 onwards which shows the growing interest of machine learning in the code clone research in recent 5 years. (Note: There are 5 primary studies belonging to the year 2020 up to May, 2020).

4.1. RQ1: What are the different areas of code clone research in which machine learning techniques have been applied?

As shown in above Fig. 4, there is a growing trend towards use of machine learning techniques [19] in code clone related research. After reading the primary studies, we have classified the studies into different categories of code clone related research (shown in Table 5) in which machine learning techniques have been applied. First five areas listed in Table 5 belong to detection of clones from software. As it is possible to observe, the machine learning techniques are mostly used in these areas i.e., 37 primary studies used machine learning for clone detection. Recent research in code clone detection focuses on detecting semantic clones using machine learning techniques. 13 primary studies used machine learning for semantic clone detection. This trend in research of code clones shows the capability of machine learning techniques for detection of semantic clones which was considered a challenging task a decade ago. A few tools are freely available that can be used to detect semantic clones.

In the following subsections, we discuss the studies according to different areas of code clone research listed in Table 5. We discussed all the studies in different categories in chronological order so that it becomes easy to identify the latest research in each category.

4.1.1. Code clone detection

There are a number of studies which focus on clone detection using machine learning techniques. The studies focusing on detection of semantic clones [1,8] are discussed in a separate category named semantic clone detection (discussed in next subsection). Here, we discuss the studies which focus on detection of clones other than semantic clones. Also, the research related to binary clone detection and cross-language clone detection is discussed separately in following subsections.

Shawky and Ali [35] proposed an approach which used clustering to detect clones. The approach used a set of six metrics extracted from function level granularity of source code. Then metric vectors were generated using different permutations of extracted metrics values on which subtractive clustering was applied to generate clusters of similar codes. The approach was evaluated with two subject systems written in C using precision and recall.

Cesare et al. [34] proposed an automated system based on machine learning to detect package level clones. Supervised learning algorithms, Naïve Bayes, Multilayer Perceptron, C4.5 [77]

Table 5
Code clone research areas where machine learning has been applied.

Sr. No.	Code clone research area	Code	Count	Citation
1.	Code clone detection	C1	16	[20–35]
2.	Semantic clone detection	C2	13	[36–48]
3.	Binary code clone detection	C3	2	[49,50]
4.	Cross-language code clone detection	C4	2	[51,52]
5.	Source code representation for clone detection	C5	4	[53–56]
6.	Code clone classification/validation	C6	3	[57–59]
7.	Clone recommendation for refactoring	C7	3	[60–62]
8.	Clone evolution	C8	3	[63–65]
9.	Evaluating clone detection precision	C9	3	[66–68]
10.	Prediction of harmfulness of cloning operations	C10	5	[69–73]
11.	Prediction of code clone quality	C11	1	[74]
12.	Detection of bugs in clones	C12	1	[75]
13.	Prediction of risky clones	C13	1	[76]

and Random Forest were used to detect shared and embedded package clone detection. They obtained highest accuracy using random forest classifier in case of shared package level clone detection and in case of embedded package level clone detection, C4.5 achieved higher accuracy as compared to other classifiers.

Joshi et al. [33] developed a clone detection approach based on clustering. The technique detected Type-1 and Type-2 clones [1] using the DBSCAN clustering algorithm. The approach extracted features from all the functions present in the subject system and applied clustering to partition the dataset into clusters. Each cluster represented the group of all functions with high similarity with each other. The technique was evaluated using open-source software, Bitmessage written in Python containing 260 functions. The study shows that if the value of Epsilon in DBSCAN is set to one, the results are achieved with sufficient number of clones and high value of precision.

Keivanloo et al. [22] proposed threshold free approach for Type-3 clone [1] detection at function level granularity. The approach applied K-means clustering to separate true and false clones. Friedman method was used to select the different values of k to evaluate the quality of clustering. The approach improves the F-measure by 12% when approach is applied on large scale heterogeneous repository and also eliminates the developer's concern about misconfiguration of Type-3 clone detection tools.

White et al. [20] presented a deep learning-based approach where representations of clone fragments are automatically mined from repository. Lexical analysis is paired with recurrent neural network [78] and syntactic analysis is paired with recursive neural network so that automatic linking can be established with patterns mined at lexical level with patterns mined at syntactic level. The clone detection results of the learning-based technique show that 93% of file and method level pairs are true positive and map to all four clone types.

Li et al. [31] developed a tool CCLearner which is based on token-based clone detection technique using deep learning. The tool extracts tokens from method code clones and non-clones to train a binary class classifier. The classifier was trained and tested using clones of the BigCloneBench dataset. ANTLR lexer and Eclipse ASTparser were used to extract features that characterize the clone and non-clone relationship of method pairs. An open-source library, Deeplearning4j, was used to train the classifier. The study also compared CCLearner with NiCad, Deckard and SourcererCC clone detection tools and reported that CCLearner achieved high recall and precision with low time cost.

Shi et al. [28] proposed a framework to find vulnerable operating system code clones which used learning on correlations between functions. In the training phase, functions related to vulnerabilities are extracted from the commits of the latest operating system code repository and function features are extracted from their AST structure. Then, graph modelling of function calls is used to explore internal and external correlations of functions. The graph convolutional neural network for code clone detection is trained using both function correlations and features. During the detection phase, the trained graph convolutional network detects the vulnerable code clones after extracting functions from the to-be detected code repository of the operating system. The study conducted experiment on five real operating systems and found that the proposed approach is more effective in vulnerable code clone detection than other methods.

Zeng et al. [23] proposed a method for fast clone detection using weighted recursive autoencoders(RAE) to detect similarity of code fragments at function level. The deep learning-based approach learned features automatically. Firstly, program abstract syntax tree is analysed using weighted RAE, program features are extracted and encoded the functions into vectors. The approach used the TF-IDF model to weight different nonterminal types in AST. It increases the accuracy of RAE by amplifying the contributions of more important nodes to the final vector representations of programs. The approach is scalable to large datasets and detects clones in a short time.

Wang et al. [24] developed a tool, Go-Clone, for detection of clones in Go programming language. The approach calculated labelled semantic flow graph for each function in a program. The tool trains a deep neural network to encode labelled semantic flow graph for similarity classification. Go-Clone achieved 83.80% accuracy and 89.61% AUC (Area under curve).

Gao et al. [25] proposed a clone detection approach for Type-3 clones based on AST. Initially code is converted into AST. After that AST is mapped to a vector based on deep learning technique and then code clones are detected by comparing the Euclidean distance of the vectors. The approach was evaluated with BigCloneBench and 7 large Java projects. The results showed that the proposed approach outperforms the other Type-3 clone detection tools like NiCad, CCLearner and CCAligner [79] in terms of recall and accuracy.

4.1.2. Semantic clone detection

Semantic clones are those code fragments which are syntactically dissimilar but have the same semantics [1]. Since semantics of code fragments need to be extracted to check a cloned fragment, detection of semantic clones is treated more difficult as compared to detection of other types of clones [8,9]. Here we discuss studies which used machine learning in detection of semantic clones.

Sheneamer et al. [42] applied supervised learning classifiers like XgBoost [80], Random Forest [81], Rotation forest [82] etc. to detect Type-3 and Type-4 clones. The approach captures the syntactic and semantic features of code fragments from AST and PDG of the methods and represent them as vectors to train the machine learning models. They extracted 10 traditional features, 32 syntactic features and 28 semantic features of methods that belong to different clone types from the BigCloneBench

benchmark. The results show that as more features are used to train the model, all classifiers show an increase in performance. They concluded XgBoost as an excellent algorithm for detecting Type-3 and Type-4 clones when compared with NiCad [83], SourcererCC [84], CCFinder [85], Deckard [86] and iClones [87].

Sheneamer et al. [44] proposed methods for labelling Type-4 clones in Java. Firstly, they used an unsupervised approach to label Type-4 clones and validate the labelled clones using Java expert programmers. Then, a supervised scheme was used for labelling unknown samples. Feature vectors are created for each block after extracting features from AST, PDG and BDG (Byte code dependency graph) of the blocks. The classifiers used by the approach are Instance Based Learner, Random Forest, Rotation Forest, Random Committee and Bagging. Results show that ensemble approach perform better in terms of MAE (Mean Absolute Error) and all classifiers show similar performance in terms of KSA (Kappa Statistic Agreement) and Accuracy.

Ghofrani et al. [43] proposed deep neural networks to detect semantically similar code clones. They used convolutional neural network to generate summaries of code fragments and then, used these summaries as metrics to find semantic similarity between the code fragments. Cosine similarity was used as a similarity indicator between the metrics of code fragments which classify whether the code fragment is a clone or not. But the evaluation results of the proposed technique are not reported.

Sheneamer et al. [40] explored the use of an ensemble classifier based on majority voting approach for detection of semantic clones and obfuscated code. The approach obtained the features of code fragments from PDG, BDG and AST and then, feature vectors were used to train classifiers. The approach used Naïve Bayes, Random Forest, LibLinear SVM [88], Instance Based Learner [89], LogitBoost [90], Bagging [91], Random Committee and Rotation forest classifiers and ensemble [92] them to obtain final class label.

Wei and Li [47] used supervised deep learning to learn deep features to detect functional clones especially Type-3 and Type-4 clones. They formulated the clone detection as a supervised learning problem which learns hash functions and representations of code fragments using AST based LSTM (Long Short-Term Memory). The framework named CDLH (Clone Detection with Learning to Hash) learns hash codes by using lexical and syntactical information of code fragments to compute functional similarity. The experimental results on clone detection benchmarks showed that CDLH is effective and outperforms the other approaches in functional clone detection.

Sheneamer [45] proposed the CCDLC framework to improve detection of Java code clones and obfuscated code. Features are extracted from AST, PDG and BDG representation of source code. Sequential information bottleneck (sIB) clustering method is used as an additional step to detect clones using convolution neural network. The results show that adding clustering to classification algorithms is very helpful in clone detection. The approach improved the accuracy by 5.44% and 12% for detection of Java obfuscated code and clone detection respectively.

Saini et al. [36] proposed a tool named Oreo that was based on deep learning to detect code clones. Deep neural network with Siamese architecture was used to detect hard-to-detect clones, specially, Type-4 clones. They compared the Siamese deep neural network with other machine learning models such as Logistic regression, Random Forest, Shallow NN with single hidden layer and plain fully connected network with the same layer sizes as the full Siamese architecture. The comparison results depict that the Siamese Neural network gives more precision and recall than other machine learning models.

Zhao and Huang [46] proposed a tool named DeepSim to detect semantic clones. The approach encoded the control and data flow of source code into a semantic feature matrix. The control and data flow graphs were generated from bytecode of source code. The approach uses a Deep neural network that learns features from sematic matrices and performs binary classification. They evaluated DeepSim with two datasets BigcloneBench [93] and Google Code Jam and results showed that DeepSim performed better than Deckard, CDLH [47] and DLC [20].

Chen et al. [37] proposed semantic clone detection approach using tree based convolution over API-enhanced AST. The proposed approach does not require conversion of AST into a full binary tree as required in tree-based LSTM. The approach was compared with a tree-based LSTM based approach named CDLH and results showed that the proposed approach outperformed in terms of F1-score when compared using OJClone and BigCloneBench datasets.

Yu et al. [41] proposed an approach to detect semantic clones using deep learning. The approach applied tree-based convolution over the token enhanced AST by capturing the structural and lexical information of code fragments from AST and code tokens respectively. AST was enhanced by adding tokens as child nodes to the corresponding AST nodes. New embedding technique called PACE (Position aware character embedding) was also proposed to overcome the problem of highly dissimilar code fragments that were used to test the model. The experimental results showed that the proposed approach outperforms the CDLH with an increase of 0.42 F1-score on OJClone benchmark and 0.15 F1-score on BigCloneBench benchmarks.

Wehr et al. [38] proposed the use of LSTM based recurrent neural network to capture the semantics of source code for detection of semantic clones and other software engineering tasks like code recommendation, code search and bug detection. The proposed model encoded SBT (Structure based traversal) representation of code fragment AST into a vector using a single layer RNN with a 128-dimensional LSTM cell. The model was trained and tested using Python projects available on Github.

Yuan et al. [48] proposed an approach for semantic clone detection. They divided the semantic clones into local semantic clones and global semantic clones. Local semantic clones are sets of lines which are functionally similar whereas global semantic clones are methods which are functionally identical but implementation is different. The approach used control flow graph as intermediate representation of source code methods. The approach combined the classical dynamic time wrapping (DTW) algorithm with bidirectional RNN autoencoder and graph convolutional network to detect clones. The experimental results showed that the proposed approach achieved good results in detecting local and global semantic clones.

Wang et al. [39] proposed an approach to detect semantic clones. The approach is based on using graph neural networks to capture the structural and semantic information of code fragments. Firstly, graphical representation of source code is created using flow augmented abstract syntax tree (FA-AST) which adds data and control flow edges to original AST. Second step calculates vector representation of source code fragments using a deep learning model called graph neural network. Then, clone detection is performed by calculating cosine similarity of code vector representation. They used Google Code Jam and BigCloneBench datasets to evaluate the approach and results of evaluation showed that the proposed technique outperformed the existing semantic clone detection techniques based on AST based deep learning.

4.1.3. Binary code clone detection

Here we discuss the primary studies that used machine learning based approaches to detect clones from binary code. Binary clone detection is important for those applications for which only binary executables are available.

Xue et al. [50] proposed an approach, Clone-hunter, that eliminates redundant bound checks in binary applications through code clone identification. The clones in binary executables were identified using a machine learning based approach. The code regions are embedded into feature vectors on which Affinity Propagation clustering algorithm is applied for binary clone detection. The clone detector was implemented in Python using the machine learning tool Scikit-learn.

Xue et al. [49] presented a framework, Clone-slicer, to detect domain specific binary code clones from legacy applications. The approach considered the pointer related domain to detect code clones in binary executables. They used program slicing to filter out non-domain specific instructions and then used deep learning algorithms, Recursive Neural Network and Recursive Autoencoder, to generate feature vectors of the remaining domain specific instructions. These feature vectors were used to deploy clustering algorithm that formed clusters and detected code clones. The approach was evaluated using SPEC 2006 benchmark and results showed that Clone-Slicer can detect more binary code clones in less time as compared to a similar approach named Clone-Hunter [50].

4.1.4. Cross language code clone detection

Perez and Chiba [51] presented a clone detection approach to detect clones in different programming languages having similar syntax. The approach used unsupervised learning to learn token level vector representation and then used LSTM based neural network to detect clones. The cross-language clone dataset was created containing 45,000 code fragments written in Java and Python.

Nafi et al. [52] proposed a cross-language clone detection approach using deep learning. Using deep neural network, the proposed approach can detect cross language clones with greater accuracy as compared to other techniques [51]. CLCDSA can detect cross language clones in two ways. First, it can detect clones by measuring cosine similarity between matrices generated by capturing nine features from code fragments. Secondly, using Siamese architecture based deep neural network that learns values of syntactical features automatically from labelled data with greater accuracy. Cross-language dataset in Java, Python and C# programming languages, which was used for training and testing deep neural network model, was collected from AtCoder, Google CodeJam and CoderByte.

4.1.5. Source code representation for clone detection

To detect clones from source code, the source code is converted into suitable intermediate form. Different intermediate representations are used for clone detection like token sequences, trees, graphs, assembly code, etc. The section explores how machine learning algorithms are being used to learn features automatically from these intermediate representations of source code that can further be used for clone detection.

Tufano et al. [53] represented a deep learning approach that can automatically learn features of source code from various source code representations like tokens, CFG, AST and bytecode. They demonstrated that use of models that learn code similarities from different code representations perform better as compared to the models based on single code representations. The approach used Recurrent Neural Network and Recursive Autoencoder for automatically learning embeddings. Euclidean distance between embeddings was computed to detect similar code fragments.

Zhang et al. [54] proposed a AST-based neural network for source code representation. The approach splits large AST into sequences of small statement trees and these statement trees are encoded into vectors. These vectors are used to produce the vector representation of the code fragments using bidirectional recurrent neural network. Evaluation results using benchmarks, OJClone and BigCloneBench show that the proposed approach gives better results as compared to DLC [20] and CDLH [47] for code clone detection

Buch and Andrzejak [55] studied supervised learning-based aggregation schemes for code clone detection. They showed that error scaling is an effective way to address the problem of class imbalance in supervised code clone detection. They also mentioned the importance of splitting training and test data by clusters to measure generalization of supervised learning models.

Keller et al. [56] presented a novel approach to learn semantic representation of source code that is based on visual representations of code fragments. The approach is based on transfer learning from pre-trained image classification neural networks. The pre-trained model was used to build a deep feature extractor that was used to produce feature vectors from images renderings of source code fragments. This novel embedding approach had been used for semantic code clone detection and achieved better results in terms of recall and F1-score as compared to ASTNN [54].

4.1.6. Code clone classification/validation

Usefulness of clone detection results depends upon the type of software engineering task in which clones' results are being used. Clone detection results can be used for clone tracking [94], refactoring [95,96], bug detection [97] etc. For each type of task, the individual user can have different views about a clone. A clone being considered useful by one user may be not suitable for another user. So, clone detection results involve manual validation which is very time-consuming task and error-prone. To ease the task of manual validation, certain approaches were proposed that use machine learning to validate clone detection results. The following paragraphs summarize those studies.

Yang et al. [58] proposed the web-based system named FICA that can filter useful clones for individual users using a supervised machine learning approach. The user first classifies a small portion of input clones manually and submits those to FICA which studies the characteristics of all marked clones using machine learning. After learning, FICA ranks the remaining unmarked clones based on results of machine learning. The user can adjust the marking on clones and resubmit them to FICA to obtain better prediction. The proposed classification model showed more than 70% average accuracy when evaluated using four open-source systems written in C and more than 90% accuracy for source code projects and individual users.

Mostaeen et al. [59] proposed an approach to validate the code clones whether useful or not using machine learning. The approach works on top of clone detection tools to filter the useful clones as per user preferences. The machine learning models were first trained using a subset of user's validated clones. Then, trained models were used to report the useful clones. The study reported that Artificial neural network achieved good results for predicting useful clones as compared to other machine learning algorithms.

Mostaeen et al. [57] presented a machine learning tool, CloneCognition, for validating code clones. The tool uses an artificial neural network for learning human validated clones from IJaDataset 2.0 and then uses its prediction results to automate the clone validation process. The tool shows accuracy of 87.4% and also gives better results as compared to FICA [58].

4.1.7. Clone recommendation for refactoring

Clone refactoring [98] is one of the major activities for clone management. All clones, detected by a clone detection tool, are not suitable for refactoring. Some approaches are required to filter useful clones for refactoring. Here we discuss the primary studies which address this issue using machine learning.

Wang and Godfrey [61] proposed an approach to recommend clones for refactoring by training decision tree [99] based classifier. They analysed more than 600 clone instances to collect features of source code, context and history of clones. Metric based approach was used to collect clone refactoring instances from three open-source subject systems. Decision tree-based classifier was trained from features of equal number of both refactored and unrefactored instances found in clone evolution history. iClones, a token-based clone detection tool, was used to detect clones from ArgoUML, Apache Ant and Lucene. The results showed that more than 80% precision was achieved to recommend clones for refactoring using decision tree classifier.

Yue et al. [62] developed a learning-based approach that recommends clones for Extract method refactoring of clones. The recommendation approach is based on 34 features that are extracted from current status and past history of software. They include 17 features which belong to clone content and evolution history and 17 features that belong to relative location, syntactic difference and co-change among clones. By default, the approach used AdaBoost [100] machine learning algorithm but they also evaluated suitability of other machine learning algorithms like Random Forest, C4.5, SMO [101] and Naïve Bayes. The results show that AdaBoost suggests clones for refactoring with high accuracy and also tree based algorithms perform better than the other machine learning algorithms for recommending clones for refactoring.

Rongrong et al. [60] used Naïve Bayes, C4.5 and Bayesian network machine learning algorithms for recommending clones for refactoring. The approach extracted 13 static features and 3 evolution features of code clones to train the machine learning algorithms. The experimental work used 7 open source software written in C language and concluded that decision tree and Bayesian Network [102] achieved the highest accuracy in recommending clones for refactoring but decision tree was more stable and therefore, worked best for recommending clones for refactoring.

4.1.8. Clone evolution

Clone management cannot be completed without understanding the evolution of the clones across different versions of its source code. History of changes that are experienced by a clone helps to extract useful features that can help to take various decisions of clone management like clone tracking or clone refactoring. In the following paragraphs, we discuss the studies related to clone evolution that used machine learning.

Pati et al. [63] used ARIMA and Neural network model for software clone evolution prediction that helps to reduce software maintenance effort. The training of neural network was performed using BFGS quasi-newton method based back propagation and prediction accuracy of the model was improved using Multi-Objective Genetic Algorithm based Neural Network training algorithm. ArgoUML dataset for 31 versions was divided into training and test dataset with 25 samples and 6 samples respectively. The study results confirmed that MOGA-NN model is the most suitable predicting model for clone evolution prediction.

Zhang et al. [64] proposed an approach to perform analysis of clone evolution using X-means clustering. They extracted metrics from clone fragments, clone groups and clone genealogies to represent cloned code and then applied X-means clustering to explore characteristics of clone evolution. The study depicts that clones are stable during evolution and clones do not undergo changes at early stages of evolution. Therefore, maintainers should pay more attention to clones that have existed in a genealogy after several evolutions.

Zhang et al. [65] used Fuzzy C-Means clustering to cluster clones and explored the relationship between clones and their evolution. NiCad was used to detect clones from six open-source software written in C, C# and Java. Clone clustering vectors were generated using various static and evolutionary clone metrics. The result of empirical study showed that short lived clones experience more inconsistent changes, exact clones have a very short life and concluded that more focus should be given to near miss clones for clone management.

4.1.9. Evaluating clone detection precision

Performance of clone detection tools is measured in terms of recall and precision. Recall represents the ratio of the number of clones detected by a tool to the total number of clones in the software. Precision measures the accuracy of detected clones i.e., ratio of true positive to the number of clones detected by a tool. Measuring precision of a tool has attracted the use of machine learning techniques because manual validation of detected clones is difficult and time consuming. The following paragraphs discuss the studies that involve machine learning to calculate precision of clone detection tools.

Svajlenko and Roy [67] proposed an approach to cluster similar clones using unsupervised learning. The approach can be used to increase the variety of clones examined while measuring precision. The motive of approach is to decrease effort while measuring precision of a clone detection tool and doubling the variety of clones validated. The study used NiCad as a clone detection tool and Java class library as a subject system. The results showed that the proposed approach more effective in measuring an accurate and generalized precision of a clone detection tool than a pure random sampling approach.

Arammongkolvichai et al. [66] proposed a machine learning based approach to improve precision of a clone detection tool. The study used iClones to detect clones. Decision tree classifier was used to remove false positive results of clone detection from JFreechart software written in Java. The decision tree classifier was trained using 19 clone class metrics. The approach improved the precision of iClones from 0.94 to 0.98. But they also found that their designed approach is not suitable for Django Python projects.

Saini et al. [68] presented a web based tool, InspectorClone, that enables to find precision of clone detection tools and creates a dataset of validated code clones which has been validated through the tool or through manual inspection. Siamese architecture was used to train deep neural networks model. The experiment shows that the precision of InspectorClone is very high when the automatic clone resolution part of this approach is evaluated on seven different clone detectors.

4.1.10. Prediction of harmfulness of cloning operations

All clones cannot be considered harmful. Some clones evolve independently and some require consistent changes to decrease maintenance effort. In the following paragraphs, we discuss the studies on harmfulness prediction of clones using machine learning.

Wang et al. [69,70] proposed an approach that automatically predicts the harmfulness of cloning operation at copy-paste time using Bayesian Networks [102]. The approach used history features, destination features and code features to predict the harmfulness of intended cloning operations. When the developer tries to perform any cloning operation, the values of features are extracted and the trained predictor will generate a score which shows the harmfulness of the intended cloning operation. The approach [69] was evaluated using two Microsoft software projects written in C# based on two scenarios. In the first scenario, the approach approved 52% to 60% of harmfulness of cloning operation with the precision value of 94.9% and in the second scenario, the approach blocked the harmful cloning operations from 48% to 67%.

Zhang et al. [72] proposed an approach to predict clone consistency requirements for a clone group when one of its clone fragments undergoes any change. The code, context and evolution features were extracted from changed code fragments and the data was used to build a Bayesian network. The approach was evaluated using four open-source software written in Java and showed that good precision and recall was achieved.

Yan et al. [71] used an ensemble feature selection model for optimizing the training dataset in the harmful prediction of code clones. Random Forest classifier was used for evaluating the optimized training dataset. The results depict that integrated classifier algorithm increases the accuracy, F1-score and AUC evaluation index when applied for prediction of code clone harmfulness.

Zhang et al. [73] used Bayesian Network to predict clone consistency requirements during software development. The model predicts whether a clone creating instance can cause consistency requirement or remains consistency free at clone creation time. The approach model trained the prediction model with code and context attributes of clone instances. They evaluated the prediction model with four open-source projects written in Java and showed that the model achieved good precision and recall.

4.1.11. Prediction of code clone quality

Liu et al. [74] used a Bayesian network to predict the quality of code clones. The results of prediction can be used to help developers to take refactoring decisions. The metrics related to maintenance overhead (MO) and lack of software quality of the clone (LSQ) were used to train the model. They concluded that if results show low value of MO and high value of LSQ, then clones need to be restructured.

4.1.12. Detection of bugs in clones

Steidl and Göde [75] used a machine learning technique to identify incomplete bug fixes in cloned code. Global context features and local gap features were identified to find incomplete bug fixes in code clones. These features of cloned code were used to train the decision tree classifier which automates and speeds up the process of bug detection in gapped clones.

4.1.13. Prediction of risky clones

Imazato et al. [76] proposed an approach to identify risky clones in the source code using machine learning techniques. Clone genealogies were extracted and marked as risky if the genealogy had undergone bug fix during its evolution. Then, features of risky and not risky clones were used to construct a learning model. The results show that J48 algorithm gives 93% precision and 89% recall, SVM gives 95% precision but recall is lowest (83%) among three algorithms and Bayesian network gives highest recall (90%) but lowest precision (83%) in prediction of risky clones.

Inferences of RQ1:

In clone detection area, the approach proposed by Zeng et al. [23] achieved highest precision (99.62%) on BigCloneBench. The approach used recursive autoencoders to detect function level clones and does not require labelled dataset. Machine learning based clone detection tool, TBCCD [41] and FA-AST+ GMN [39] achieved 99% precision on OJClone dataset and GoogleCodeJam respectively. Both tools used deep learning-based approaches. TBCCD [41] used Tree-based Convolution neural network whereas FA-AST+ GMN [39] used graph neural networks for semantic clone detection.

Binary clone detection tool, Clone-Slicer [49] combines clustering and deep learning approaches for clone detection and achieved better results than Clone-hunter [50] which used only clustering for detecting binary code clones.

Deep neural network based cross-language clone detection tool, CLCDSA [52] achieved greater accuracy than approach [51] that used LSTM based neural network.

Code semantics are learned from different code representations for clone detection. Semantic learning from visual representation of source code and use of pretrained image classification model for generating embeddings have shown potential for code clone detection and code classification. Keller et al. [56] proposed an approach that learned semantics from visual representation of source code and it performed similarly well as ASTNN (based on AST-based neural embeddings on small statement trees) in terms of F-Measure for clone detection.

AdaBoost and feature set proposed by [62] gives better performance than Decision tree and feature set proposed by [61] for recommending clones for refactoring.

Similarly, machine learning algorithms are indeed useful in areas like automating clone recommendation for refactoring, calculating precision of clone detectors, predicting harmfulness of cloning operations, prediction of bugs in clones and prediction of risky clones. A software can contain a large number of clones but all clones cannot be removed through refactoring. Machine learning based recommendation can help to automatically filter unimportant clones for refactoring. Evaluating the precision of a clone detection tool is a big challenge because all clones cannot be evaluated as true or false positives. Automation in calculating precision using machine learning approaches can decrease time and effort spent during measuring precision of a clone detection tool. Similarly, maintenance cost caused by harmful cloning operations, bugs and risky clones can be decreased using machine learning based approaches for prediction.

Summary of Results for RQ1:

We observed that machine learning algorithms are widely used in certain areas of code clone research, such as clone detection, especially semantic clone detection and prediction of harmfulness of cloning operations. But there are very few studies which applied machine learning in areas such as prediction of code clone quality, prediction of risky clones and detection of bugs in clones. Research can be conducted in these areas using machine learning.

4.2. RQ2: What machine learning algorithms have been used in different primary studies?

There is a rapid increase in the use of machine learning in the software engineering field in the last decade. After review of primary studies, we have noticed that a large variety of machine learning algorithms have been used for code clone research. Table 6 shows the different machine learning algorithms along with frequency of use in various primary studies. We have classified existing machine learning algorithms from our literature into three categories: Conventional Machine learning, Clustering and Deep Learning. We intentionally classified some of the machine learning algorithms into the category 'Clustering'. We understood the importance of clustering algorithms in clone detection. We want to tell the readers that some clustering algorithms may fall in the category of Conventional machine learning algorithms or vice-versa.

Conventional machine learning algorithms are used most frequently in primary studies as compared to clustering and deep learning algorithms as shown in Table 6. Fig. 5 shows deep learning has been started to use in code clone research in recent years. There is an increasing trend in use of deep learning algorithms from the past five years whereas conventional machine learning algorithms are most popular over the years. (Note: In the year 2020 up to May 2020, deep learning algorithms are used in 4 studies out of total 5 primary studies, conventional machine

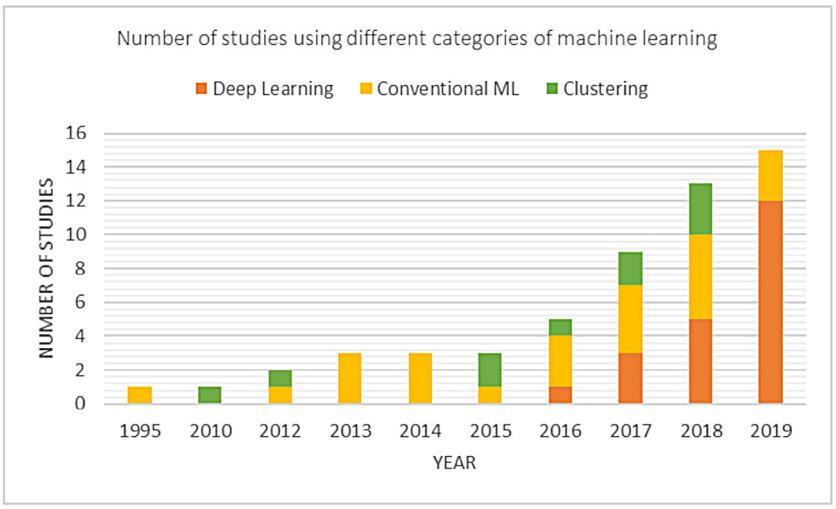

Fig. 5. Studies using different machine learning algorithms over the years.

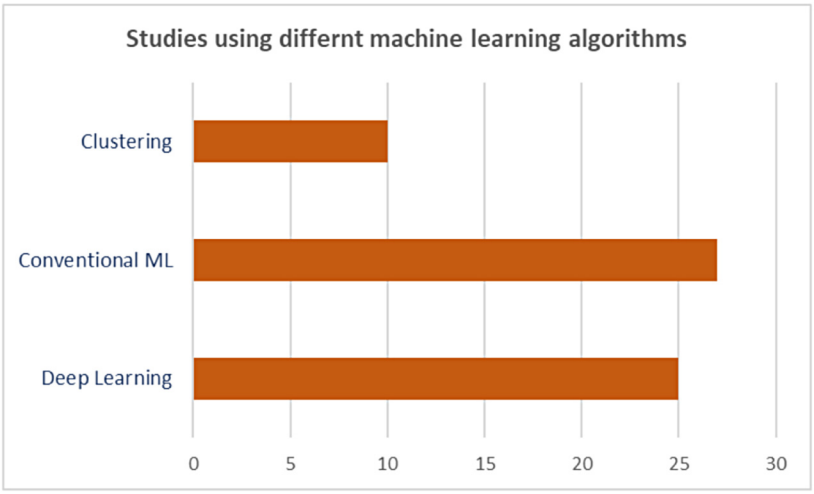

Fig. 6. Studies using different machine learning algorithms.

learning algorithms are used in 3 primary studies and no primary study used clustering).

Fig. 6 shows that 27 out of 57 primary studies used conventional machine learning algorithms, 25 out of 57 primary studies used deep learning algorithms and 10 studies used clustering algorithms. There are four primary studies [36,45,49,56] which used more than one category of machine learning algorithms. Study [49] used clustering with deep learning to detect binary code clones. Study [56] used deep residual network to extract semantics of source code and used conventional machine learning algorithms(SVM, K-nearest neighbour and neural network) for clone detection. Study [45] used clustering with deep learning to detect semantic clones and then compare the approach with other conventional machine learning algorithms. Study [36] compared the performance of deep neural network with conventional machine learning algorithms for clone detection.

Fig. 7 shows most of the machine learning algorithms used in different categories of code clone research. Blue nodes in Fig. 7 depict code clone research areas whereas grey and pink nodes represent machine learning algorithms. We have observed that deep learning and clustering algorithms are mostly used for detection of clones whereas conventional machine learning algorithms have been used in all categories of code clone research except Cross-language clone detection and Binary code clone detection. LSTM based neural network and Deep neural network have been used in cross-language clone detection.

Fig. 8 depicts a number of machine learning algorithms applied in various code clone research areas. Most of the machine learning algorithms are applied for semantic clone detection whereas prediction of code clone quality and detection of bugs in clones are the areas where least number of machine learning algorithms are used.

Fig. 9 maps the machine learning algorithms with different code clone research areas. Decision tree [99] is used 10 times in different areas of code clone research. Maximum conventional machine learning algorithms are used in semantic clone detection(C2) area of research. Fig. 9 concludes that clustering algorithms are mostly used in clone detection and maximum deep learning algorithms are used for semantic clone detection.

Fig. 10 shows different machine learning algorithms used per year. These algorithms are reported along with their category and year in which these are used in primary studies. Studies published in 2018 have used a maximum number of machine learning algorithms. In this year, a study by Sheneamer [45] combined clustering with 6 conventional machine learning algorithms and 1 deep learning algorithm to check the performance for semantic clone detection whereas Mostaeen et al. [59] compared performance of 12 conventional machine learning algorithms for

Table 6
Machine learning algorithms.

Category	Machine learning algorithm	Code	Count	Citation
Conventional machine learning algorithms	AdaBoost	ML1	1	[62]
	Adaptive K-Nearest Neighbour	ML2	1	[32]
	ARIMA	ML3	1	[63]
	Artificial Neural Network (ANN)	ML4	3	[56,57,59]
	Bagging	ML5	3	[40,42,44]
	Bayesian Network	ML6	8	[59,60,69,70,72–74,76]
	Decision Table	ML7	1	[59]
	Decision Tree	ML8	10	[34,40,42,59–62,66,75,76]
	Dynamic Competitive learning	ML9	1	[27]
	Extra Trees	ML10	1	[42]
	Instance Based Learner (IBK)	ML11	4	[40,42,44,45]
	K* Classifier	ML12	2	[42,59]
	K-Nearest Neighbours algorithm	ML13	1	[56]
	LibLinear	ML14	2	[40,45]
	Linear Discriminant Analysis (LDA)	ML15	1	[42]
	Logistic regression	ML16	2	[36,59]
	LogitBoost	ML17	2	[40,42]
	Multiclass classifier	ML18	1	[59]
	Multilayer Perceptron (MLP)	ML19	4	[34,42,45,63]
	Naïve Bayes	ML20	7	[34,40,42,45,59,60,62]
	Near Neighbour Query Algorithm	ML21	1	[26]
	Random Committee	ML22	3	[40,42,44]
	Random Forest	ML23	8	[34,36,40,42,44,59,62,71]
	Random Subspace	ML24	1	[40]
	Random Tree	ML25	2	[42,59]
	RBF Classifier	ML26	1	[45]
	RBF Network	ML27	1	[45]
	Rotation Forest	ML28	3	[40,42,44]
	SMO	ML29	1	[62]
	SOM	ML30	1	[27]
	Stochastic Gradient Descent	ML31	1	[59]
	Support Vector Machine	ML32	4	[21,42,56,76]
	Xgboost	ML33	1	[42]
Clustering	Affinity Propagation (AP) clustering	MLA1	1	[50]
	Clustering	MLA2	1	[49]
	DBSCAN clustering	MLA3	1	[33]
	Fractal Clustering	MLA4	1	[30]
	Fuzzy C-Means Clustering	MLA5	1	[65]
	GMM Clustering	MLA6	1	[67]
	K-means Clustering	MLA7	2	[22,67]
	Sequential Information Bottleneck (sIB) Clustering	MLA8	1	[45]
	Subtractive Clustering	MLA9	1	[35]
	X-means Clustering	MLA10	1	[64]
Deep Learning	Convolution Neural Network	DL1	3	[29,43,45]
	Deep Neural Network	DL2	6	[24,31,36,46,52,68]
	Deep Residual Networks	DL3	1	[56]
	Graph convolutional network	DL4	2	[28,48]
	Graph Neural network	DL5	1	[39]
	Long Short-Term Memory/Tree-based Long Short-Term Memory	DL6	3	[47,51,55]
	Recurrent Neural Network	DL7	4	[20,38,53,54]
	Recursive Autoencoders	DL8	3	[23,49,53]
	Recursive Neural Network	DL9	4	[20,48,49,55]
	Tree-based Convolution	DL10	2	[37,41]

Table 7
Tools used to implement machine learning algorithms.

Tool	# Studies	Citations
Weka [103]	14	[34,42,44,45,60–62,64,69–73,75]
Tensor flow [104]	2	[29,46]
RNNLM Toolkit [105]	2	[20,49]
Scikit-learn [106]	2	[50,56]
Eclipse Deeplearning4j [107]	1	[31]
Libsvm V3.16 [108]	1	[21]
PyTorch [109,110]	1	[56]

automatic validation of code clones. Maximum machine learning algorithms are used in study [42] which compared accuracy of 15 machine learning algorithms for clone detection.

There are different tools available to implement machine learning algorithms. Table 7 summarizes the tools used in primary studies to implement machine learning algorithms. 22 primary studies out of 57 mentioned about the tool used to implement machine learning algorithm. 14 primary studies used weka to implement machine learning algorithms.

RQ2.1 What are the training strategies employed in different studies?

Machine learning models can be trained using data from the same project under analysis (with-in project) or using data from external projects(cross-project). Table 8 shows the training strategies adopted by different primary studies.

RQ2.2: What are the validation techniques used?

The validation techniques used in various primary studies are listed in Table 9. Most of the primary studies used k-fold cross validation with k = 10. There are 10 primary studies which used percentage split validation technique and only 5 primary studies used random sampling technique. Here, we observed that the primary studies listed against percentage split and random sampling used deep learning algorithms.

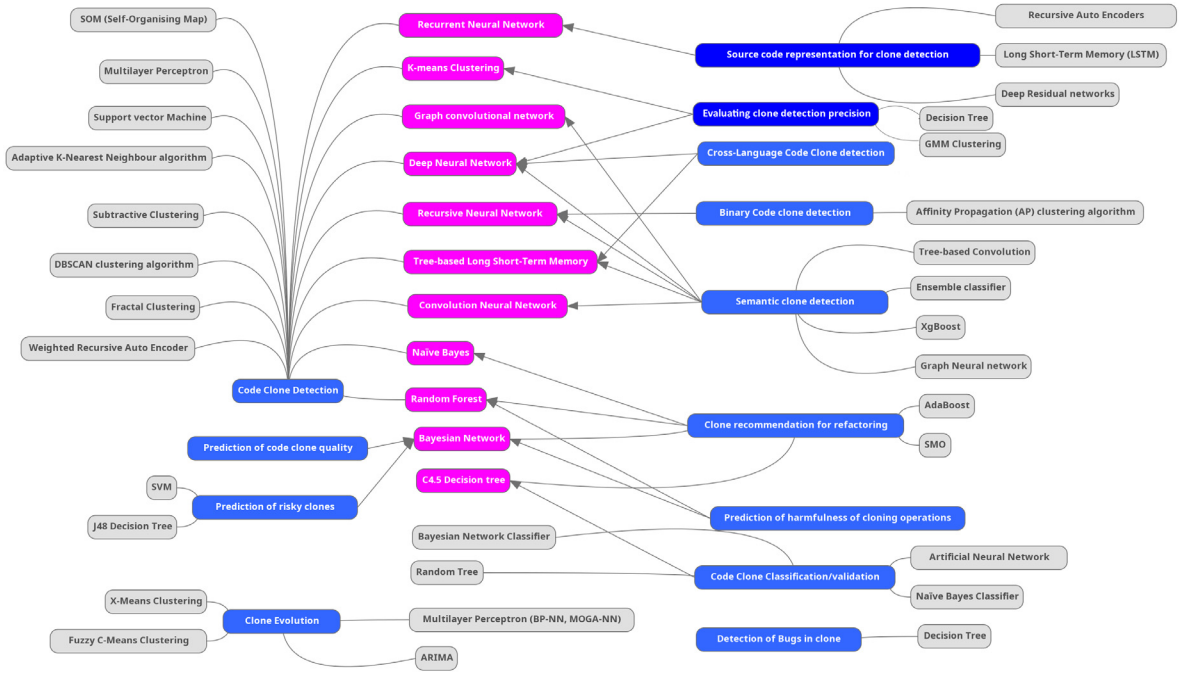

Fig. 7. Machine learning algorithms used in different code clone research areas.

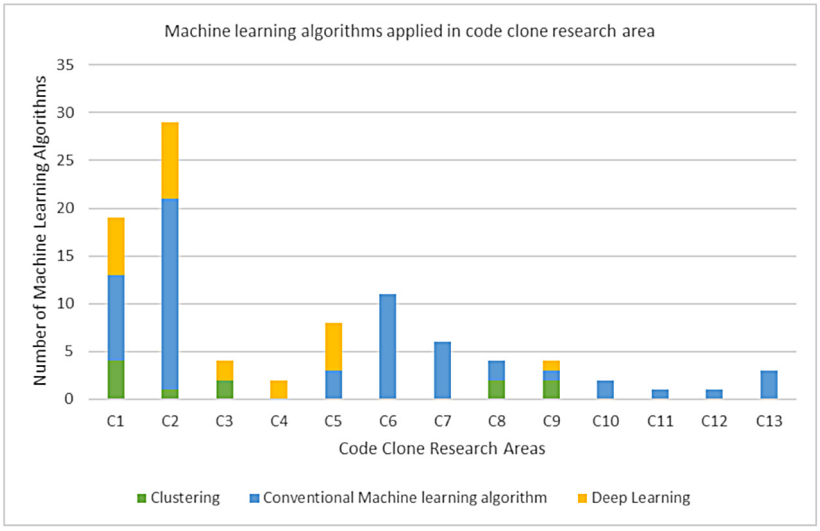

Fig. 8. Machine learning algorithms used in code clone research areas.

Table 8
Training strategy.

Training strategy	Area of study	Study
With-in project and Cross-project	Clone recommendation for refactoring	[61,62]
With-in project and Cross-project	Code clone classification/validation	[58,59]
With-in project and Cross-project	Prediction of harmfulness of cloning operations	[69,70,72,73]
With-in project	Prediction of risky clones	[76]
With-in project	Evaluating clone detection precision	[66]

Table 9
Validation techniques.

Validation technique	#	Study
K-fold cross validation	19	[21,34,40,42,44–46,53,57,59–62,69,70,72,73,75,76]
Percentage split	10	[36,37,39,41,51,54–56,63,66]
Random sample	5	[23,29,48,52,68]

Machine Learning Algorithms			Code Clone Research Areas	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
	Code	Count		20	48	4	2	9	12	9	4	4	5	1	1	3
Conventional Machine Learning	AdaBoost	ML1	1							1						
	Adaptive K-Nearest Neighbour	ML2	1	1												
	ANN	ML3	3					1	2							
	ARIMA	ML4	1								1					
	Bagging	ML5	3		3											
	Bayesian Network	ML6	8						1	1			4	1		1
	Decision Table	ML7	1						1							
	Decision Tree	ML8	10	1	2				1	3		1			1	1
	Dynamic Competitive Learning	ML9	1	1												
	Extra Trees	ML10	1		1											
	IBK	ML11	4		4											
	K* Classifier	ML12	2		1				1							
	K-Nearest Neighbours	ML13	1					1								
	LDA	ML14	1		1											
	LibLinear	ML15	2		2											
	Logistic Regression	ML16	2		1				1							
	Logit Boost	ML17	2		2											
	MLP	ML18	4	1	2						1					
	Multiclass classifier	ML19	1						1							
	Naïve Bayes	ML20	7	1	3				1	2						
	Near-Neighbor query algorithm	ML21	1	1												
	Random Committee	ML22	3		3											
	Random Forest	ML23	8	1	4				1	1			1			
	Random Subspace	ML24	1		1											
	Random Tree	ML25	2		1				1							
	RBF Network	ML26	1		1											
	RBFClassifier	ML27	1		1											
	Rotation Forest	ML28	3		3											
	SMO	ML29	1							1						
	SOM	ML30	1	1												
	Stochastic Gradient Descent	ML31	1						1							
	SVM	ML32	4	1	1			1								1
	XgBoost	ML33	1		1											
	Total			**9**	**38**	**0**	**0**	**3**	**12**	**9**	**2**	**1**	**5**	**1**	**1**	**3**
Clustering	Affinity Propagation (AP) Clustering	MLA1	1			1										
	Clustering	MLA2	1			1										
	DBSCAN Clustering	MLA3	1	1												
	Fractal Clustering	MLA4	1	1												
	Fuzzy C-Means Clustering	MLA5	1								1					
	GMM Clustering	MLA6	1									1				
	K-means Clustering	MLA7	2	1								1				
	Sequential Information Bottleneck (sIB)	MLA8	1		1											
	Subtractive Clustering	MLA9	1	1												
	X-Means Clustering	MLA10	1								1					
	Total			**4**	**1**	**2**	**0**	**0**	**0**	**0**	**2**	**2**	**0**	**0**	**0**	**0**
Deep Learning	Convolution Neural Network	DL1	3	1	2											
	Deep Neural Network	DL2	6	2	2		1					1				
	Deep Residual Networks	DL3	1					1								
	Graph Convolutional Network	DL4	2	1	1											
	Graph Neural Network	DL5	1		1											
	LSTM/Tree-based LSTM	DL6	3		1		1	1								
	Recurrent Neural Network	DL7	4	1	1			2								
	Recursive Autoencoders	DL8	3	1		1		1								
	Recursive Neural Network	DL9	4	1	1	1		1								
	Tree-based Convolution	DL10	2		2											
	Total			**7**	**11**	**2**	**2**	**6**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**

Fig. 9. Mapping of machine learning algorithms with code clone research areas.

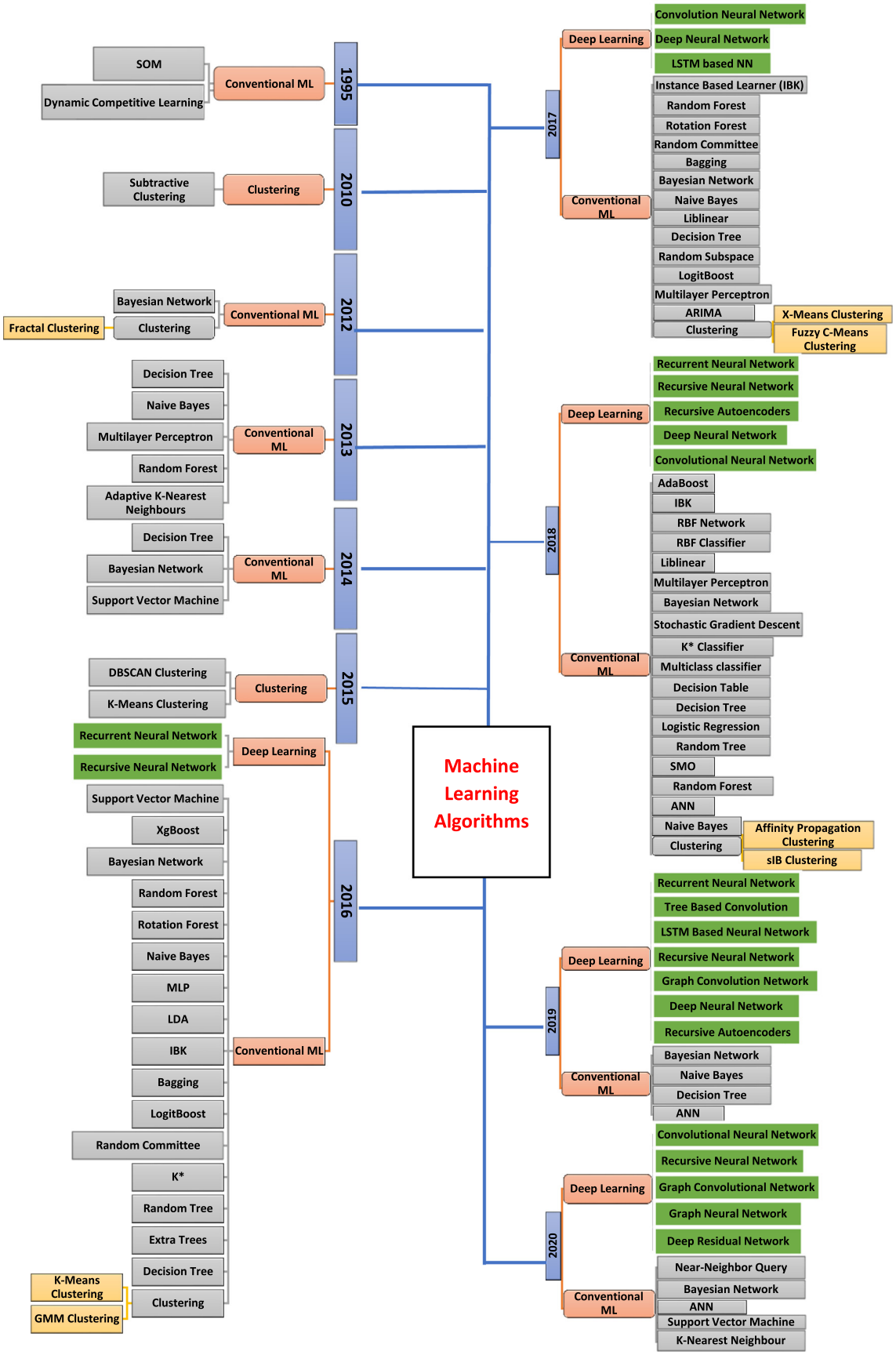

Fig. 10. Use of Machine learning algorithms in different years.

Table 10
Features used to train machine learning algorithms for recommending clones for refactoring.

Sr. No	Name/Type of features	Machine learning algorithm used	Study/ Citation
1.	Static and Evolutionary Feature	C4.5 Decision Tree, Naïve Bayes and Bayesian Network	[60]
2.	Cloning Relation, Context of clone, Cloned code snippet	C4.5 Decision Tree	[61]
3.	History Features, Co-change Features, Clone-code Features, Relative location among clones, Syntactic differences among clones	C4.5 Decision Tree, Naïve Bayes, Random Forest, SMO and AdaBoost	[62]

Table 11
Features used to train machine learning algorithms for clone evolution.

Sr. No	Name/Type of features	Machine learning algorithm used	Study/ Citation
1	Metrics related to code fragments, clone groups and clone genealogies	X-means Clustering	[64]
2	Static and evolutionary metrics	Fuzzy C-Means Clustering	[65]

Summary of Results for RQ2

There are 52 machine learning algorithms that have been applied in code clone research areas. Among these, conventional machine algorithms are used most frequently. Deep learning algorithms are gaining interest of researchers in recent years and these are mostly used for clone detection. Only conventional machine learning algorithms have been applied in some areas of code clone research. These areas are clone recommendation for refactoring, code clone classification/validation, detection of bugs in clones, prediction of harmfulness of cloning operations, prediction of code clone quality and prediction of risky clones.

4.3. RQ3: What features of code have been used to develop machine learning based models?

Feature extraction plays a significant role to train machine learning models. The accuracy of a machine learning model depends highly on the features on which the model is trained. Tables 10 to 13 shows the list of features of source code that has been used in different primary studies to train the machine learning models.

Table 10 lists the features and machine learning algorithms used in clone recommendation for refactoring studies. These studies trained machine learning algorithms using clone code, clone context and clone history features to recommend clones for refactoring. Wang et. al [61] observed that features from cloned code fragments alone can give useful results for recommending clones for refactoring. Also, performance of classifier without two features (cyclomatic complexity of cloned fragment and whether cloned fragment contains complete control flow block) is same as performance using all features. So, researchers can experiment without extracting these features to recommend clones for refactoring. Yue et. al [62] indicated that performance of their machine learning based clone recommendation for refactoring approach remains same after excluding features related to syntactic differences of clone peers. They also indicated that clone history features and co-change features of clones are more important features that increase effectiveness of their approach. Yue et. al [62] observed that using decision tree classifier with their set of features improves results as compared to results of study [61].

Clustering algorithms are used in the clone evolution area. Table 11 highlights the features that are used to perform clustering on clones. Metrics like clone life, clone group life, genealogy life related to code fragments, clone groups and clone genealogies respectively are used in study [64]. Clone clustering helps to analyse various relationship between clones and their evolution. For analysing clones using clustering, it is difficult to determine the number of clusters required. X-Means clustering algorithm helps to mitigate this problem as it automatically determines the number of required clusters before clustering. Static metrics represent static features of a clone like clone similarity and clone granularity whereas evolutionary metrics represent evolution information of a clone like clone life, number of times clone fragment changed.

Studies [40,42,44,45] related to semantic clone detection used features that are extracted from AST, PDG and BDG representation of source code (Table 12). Study [36,52] used method-level software metrics to generate feature vectors. The studies [30,33,35] used source code attributes related to data and control flow to implement clustering for clone detection as shown in Table 12.

Studies related to prediction of harmfulness of cloning operation used features related to clone code, clone context and clone history as shown in Table 13. Studies [70,73] observed that clone history features are not important for maintenance-consistency prediction during clone creation time. Exclusion of history attributes can save time and effort spent on extracting history features from historical versions of the software.

Wang et al. [69,70] observed that only code features and destination features can be used to predict harmfulness of cloning operations whereas Zhang et al. [72] indicated that all types of features (code, context and destination) are important for prediction of change consistency in a clone group. Performance of the classifier on the basis of a subset of features varies with the subject system. Zhang et al. [73] observed that context features have a positive impact on precision whereas code features improve recall. So, code and context attributes have positive impact on prediction of consistency needs

After analysing primary studies, it is observed that clone code, clone context and clone evolution related features are mostly used for prediction of results like harmfulness of cloning operations [69,70,72] and filtering of clones for refactoring [60–62]. Table 14 lists the attributes used to measure clone code, context and evolution features of source code in various primary studies. The table lists only those attributes which are used in more than one primary study. There are a number of attributes which are repeatedly used in different code clone research areas. Fig. 11 shows the use of such different attributes in different code clone research areas. If more than one attribute is used together in a particular research area then, these attributes are shown as overlapped bubbles in Fig. 11. For example, attributes having code M34, M35 and M36 are used collectively in two code clone research areas i.e., Clone recommendation for refactoring (C7) and Prediction of harmfulness of cloning operations (C10).

Summary of Results for RQ3:

We observed that feature selection to train machine learning algorithms depends on the type of code clone research area in which machine learning is being applied. Cloned code related features are most widely used in different code clone research areas. Clone evolution features are required for studies belonging to clone recommendation for refactoring, prediction of harmfulness of cloning operations, prediction of code clone quality and prediction of risky clones.

Table 12
Features used to train machine learning algorithms for clone detection.

Sr. No	Name/Type of features	Machine learning algorithm used	Study/ Citation
	Semantic clone detection		
1.	Syntactic and Semantic features of code fragments from ASTs and PDGs of its method	Xgboost, Rotation Forest, Random Tree, Random Forest, Extra Trees [111], Random Committee, SVM, Naïve Bayes, LDA, IBK, K*, Decision Tree, MLP, Bagging and LogitBoost	[42]
2.	Features are extracted from AST, PDG and BDG (Bytecode Dependency graph) representation of source code	Rotation Forest, Random Forest, Random Committee, Liblinear, Naïve Bayes, IBK, J48 Decision Tree, MLP, Bagging, LogitBoost, Random Subspace, RBF Network, RBF Classifier, sIB and Convolutional Neural Network	[40,44, 45]
3.	Features are extracted from control flow graph and data flow graph of source bytecode.	Deep Neural Network	[46]
4.	Number of external methods called, Number of local methods called, Number of variables referenced, Number of variables declared, Number of statements, Number of operators, Number of operands, Number of arguments, Number of expressions, Number of loops (for, while), Number of exceptions thrown, Number of exceptions referenced, Number of classes referenced, Number of class casts, Number of Boolean literals, Number of Character literals, Number of String literals, Number of Numerical literals, Number of Null literals, Maximum depth of nesting, McCabe's cyclomatic complexity, Halstead effort to implement, Halstead difficulty to implement, Halstead vocabulary	Deep Neural Network	[36]
	Code clone detection		
5	Features from AST representation of functions	Graph convolutional network	[28]
6	Number of declaration statements, Number of executable statements, Number of conditional statements, Number of looping statements, Maximum nesting level of control constructs, Number of return statements, Number of parameters, Number of called functions, Number of unique paths, McCabe cyclomatic complexity	Fractal Clustering	[30]
7	Number of declarative statements, Number of conditional statements, Number of looping statements, Number of parameters, Number of called functions, Number of return statements, Number of lines of code (LOC).	DBSCAN clustering algorithm	[33]
8	The number of inputs (parameters and global variables) a function uses., The number of outputs, Number of declarative and executable statements, Average number of lines containing source code for all nested functions, Cyclomatic (Edges - Nodes + Connected Components), Maximum nesting level of control constructs in the function.	Subtractive Clustering	[35]
	Cross-language code clone detection		
9	Total number of operators, Total number of operands, Number of variables declared, Number of arguments, Number of expressions, Number of loops, Number of exceptions thrown, Number of exceptions referenced, McCabe's Cyclomatic complexity	Deep Neural Network	[52]

4.4. RQ4: What are evaluation metrics used to measure the performance of machine learning based models?

Table 15 shows the evaluation metrics used in primary studies to evaluate the performance of machine learning models. Most of the primary studies, i.e., 39 studies used precision for evaluation. Recall is used by 36 primary studies. So, most of the primary studies used precision and recall to evaluate performance of machine learning algorithms used in code clone related research. The third most frequently used metric is F1-measure. Accuracy is also used as an evaluation metric in 13 primary studies.

AUC (Area under curve) value ranges from 0 to 1. Greater value is desired as it represents a better classification effect [71]. It is a good metric to measure the overall summary of the quality of the vectors that separates the clone and non-clone pairs [55]. Normalized root mean square error (NRMSE) metric was used in a study related to clone evolution prediction [63]. According to the study [63], the model that gives the minimum value of NRMSE is desirable for prediction of clone evolution. MAE(Mean Absolute error) metric was used in study [44] related to labelling semantic clones using machine learning. MAE calculates the difference between the measured value and the true value. The study [44] reported that ensemble classifier achieved lowest MAE with value 0.07% for labelling clone datasets.

The metric silhouette is used to evaluate clustering algorithms [67]. The value of silhouette of a data point in a cluster

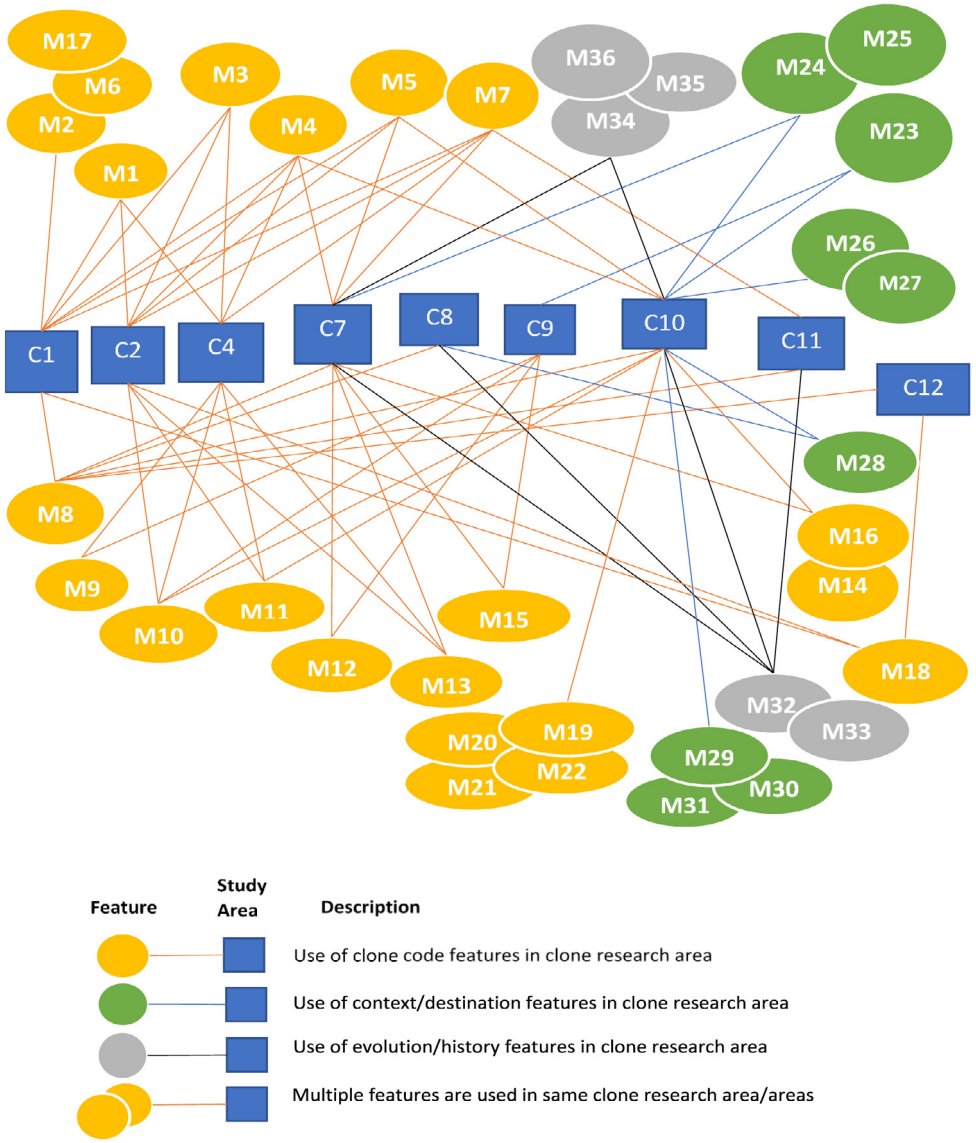

Fig. 11. Use of different features in various clone research areas.

Table 13
Features used to train machine learning algorithms for predictions related to clones and evaluating clone detection precision.

Sr. No	Name/Type of features	Machine learning algorithm used	Study/Citation	Area of study
1.	History features, code features, and destination features	Bayesian Network	[69,70,72]	Predicting harmfulness of cloning operation
2.	Static and Evolutionary Features	Random Forest	[71]	
3.	Code Features and Context Features	Bayesian Network	[73]	
4.	Maintenance Overhead for Code Clone, Lack of Software Quality of the Clone	Bayesian Network	[74]	Predicting code clone quality
5.	Local and Global Features	Decision Tree	[75]	Detection of bugs in clones
6.	Clone Class Metrics	Decision Tree	[66]	Evaluating clone detection precision

varies from −1.0 to 1.0. The silhouette of a cluster is measured as the average silhouette of its data points. Silhouette closer to 1.0 is desirable as it depicts that a data point is more similar to the data points in its own cluster. The metrics namely, warning rate and recommending rate are used in studies related to prediction of harmfulness of cloning operations.

Summary results for RQ4

Precision and recall are the metrics which are used to evaluate machine learning based approaches belonging to all code clone research areas (listed in RQ1) except approaches used in binary clone detection and clone evolution. Similarly, F-Measure was

Table 14
Code, context and evolution features used to train machine learning algorithms.

Sr. No	Features/Attribute	Code	Count	References
	Clone code features			
1	Number of declaration statements	M1	5	[30,33,35,36,52]
2	Number of conditional statements	M2	2	[30,33]
3	Number of looping statements	M3	4	[30,33,36,52]
4	Number of parameters	M4	8	[30,33,35,36,52,60,72,73]
5	Number of called functions	M5	9	[30,33,36,61,62,69,70,72,73]
6	Number of return statements	M6	2	[30,33]
7	McCabe cyclomatic complexity	M7	7	[30,35,36,52,61,62,74]
8	Number of lines of code (LOC)	M8	11	[33,60–62,65,69,70,72–75]
9	Number of identifiers	M9	2	[52,66]
10	Number of operators	M10	4	[36,52,66,72]
11	Number of operands	M11	3	[36,52,72]
12	Number of tokens that are clone	M12	3	[61,62,66]
13	Number of expressions	M13	4	[36,52,61,62]
14	Clone group size	M14	4	[61,62,72,73]
15	Clone type	M15	2	[61,66]
16	Whether a clone is test code	M16	3	[62,69,70]
17	Number of executable statements	M17	2	[30,35]
18	Maximum nesting depth of control statements	M18	4	[30,35,36,75]
19	Number of field accesses	M19	2	[69,70]
20	Number of parameter accesses	M20	2	[69,70]
21	Average number of Halstead	M21	2	[72,73]
22	Average number of important syntactic constructs	M22	2	[72,73]
	Context/Destination features			
23	File name similarity	M23	5	[66,69,70,72,73]
24	Method name similarity	M24	6	[61,62,69,70,72,73]
25	Locality of clone	M25	6	[61,62,69,70,72,73]
26	Sum of parameter similarity	M26	4	[69,70,72,73]
27	Maximal parameter Similarity	M27	4	[69,70,72,73]
28	Clone similarity	M28	3	[65,72,73]
29	Difference in only postfix numbers	M29	2	[69,70]
30	Sum of parameter type similarity	M30	2	[72,73]
31	Is-same-block	M31	2	[72,73]
	Evolution/History features			
32	Clone age	M32	7	[60,64,65,69,70,72,74]
33	Clone change frequency	M33	8	[60,62,64,65,69,70,72,74]
34	File existence time containing clone	M34	4	[61,62,69,70]
35	Number of file changes	M35	3	[62,69,70]
36	Number of recent file changes	M36	3	[62,69,70]

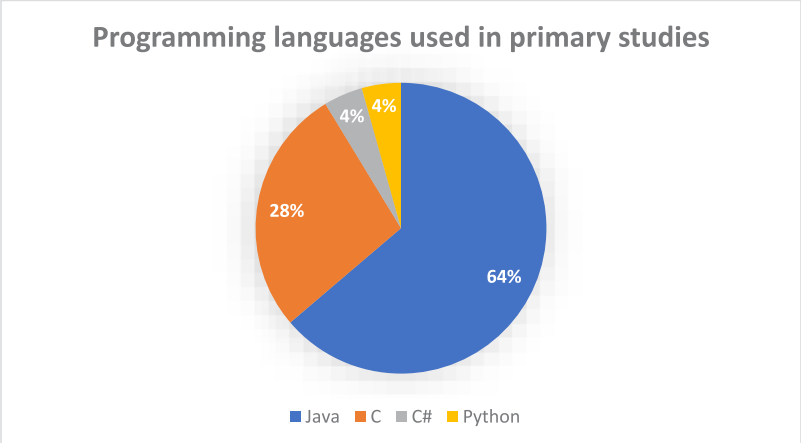

Fig. 12. Programming languages of used subject systems.

used in a most of the research areas (8 areas (C1, C2, C4, C5, C6, C7, C10 and C11) related to code clones to evaluate machine learning based approaches. Accuracy and AUC metric are used in 5 code clone research areas (C1, C2, C5, C6 and C10). Silhouette is used to evaluate clustering algorithms. Mean Absolute Error (MAE) and Normalized root mean square error (NRMSE) are used to evaluate performance of conventional machine learning algorithms. Precision, Recall, F-measure and Accuracy are used to evaluate approaches based on clustering, conventional machine learning and deep learning algorithms whereas AUC used to evaluate conventional machine learning and deep learning-based approaches.

4.5. RQ5: Which datasets have been used in the primary studies?

Table 16 includes the details of subject systems used in different primary studies. It also specifies the area of study in which the subject system was used. The studies used these subject

Table 15
Evaluation metrics.

Evaluation metrics	Definition	Count	Citation
Precision	Ratio of relevant instancing to the retrieved instances. $\text{Precision} = \frac{TP(True\ Positives)}{TP\ (True\ Positives) + FP(False\ Positives)}$	39	[20,22,23,25,28–31,33–37,39,41,42,46–48,51–54,56–62,66,68–70,72–76]
Recall	Ratio of retrieved relevant instances to total relevant instances $\text{Recall} = \frac{TP(True\ Positives)}{TP\ (True\ Positives) + FN(False\ Negatives)}$	36	[22,23,25,28–31,34–37,39,41,42,46–48,51–54,56–62,68–70,72–76]
F-measure/F1-score	Weighted Average of Recall and Precision $\text{F1-Score} = \frac{2 * (Recall * Precision)}{(Recall + Precision)}$	23	[22,25,28,29,34,37,39,41,42,46–48,51–54,56,57,60–62,71,74]
Accuracy	Ratio of correctly retrieved instances to the total instances $\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$	13	[21,24,28,34,36,40,44,45,56–59,71]
AUC (Area under Curve)	Degree of separability i.e., capability of distinguishing between classes.	7	[23,24,36,38,55,57,71]
Normalized root mean square error (NRMSE)	$NRMSE = \frac{RMSE}{\bar{y}}$ $NRMSE = \frac{1}{y_{max} - y_{min}}(\sqrt{\frac{\sum_{t=1}^{n}(\hat{y_t} - y_t)^2}{n}})$	1	[63]
Mean Absolute Error (MAE)	$MAE = \frac{\sum_{t=1}^{n} \lvert \hat{y}_t - y_t \rvert}{n}$	1	[44]
Silhouette	Degree of similarity of an object to its own cluster as compared to other clusters. $s\ (i) = \frac{b\ (i) - a(i)}{max\ \{a\ (i)\ ,\ b(i)\}}$ Here *a(i)* is the average distance between data point *i* and other data points in its own cluster and *b(i)* is the lowest average distance between data point *i* and data points in other clusters	1	[67]
Warning rate/Blocking rate	Ratio of number of warnings (harmful cloning operations) raised by prediction model to all cloning operations.	4	[69,70,72,73]
Recommending rate/Approval rate	Ratio of number of recommendations (harmless cloning operations) raised by prediction model to all cloning operations.	4	[69,70,72,73]

systems to create dataset related to code clones according to the problem being handled by the study. For example, study [61] used Apache Ant, ArgoUML and Lucene as subject systems to create a dataset of clones having refactoring history. Fig. 12 illustrates that Java open-source subject systems are the most popular choice of researchers for code clone related research. Most frequently used open-source subject systems of Java are ArgoUML, JEdit and JFreechart. ArgoUML is used in 7 primary studies, JEdit used in 6 primary studies and JFreechart used in 4 primary studies. IJadataset 2.0 [112] was used in 3 primary studies. IJadataset is a large inter-project repository consisting of about 25,000 open-source projects written in Java and mined from SourceForge, Google Code and Github. The primary studies [33,38,52,66] used Python projects for evaluation of their approach. C# subject systems were used in three primary studies. Most of the primary studies have used open-source software as subject systems. There are only three studies [69,70,75] which have used proprietary software.

Table 17 specifies the datasets/benchmarks used in primary studies. These datasets are publicly available and used for training machine learning based models that are used in code clone research areas. BigCloneBench [93] is the most popular Java clones benchmark/dataset which is used in research on code clones using machine learning techniques. It is used in 15 primary studies. BigCloneBench was built by mining clones from 25,000 Java projects and clones are verified by three domain experts. The BigCloneBench was used for two purposes in primary studies. It is used to evaluate performance of machine learning based clone detection approaches. Secondly, it is used to prepare training and test data for machine learning models used in code clone research. Major advantage of the BigCloneBench dataset is its large size. It consists of 8,584,153 true clone pairs and 279,032 false clone pairs from 43 different functionalities. The tagged clone pairs map to all clone types and therefore this benchmark is also popular to evaluate semantic clone detection tools. The disadvantage of this benchmark is that the clones are validated by functionality only. Open Judge is a benchmark/dataset for clone detection in C programs and it is published by Mou et al. [113]. It consists of solutions of 104 different programming problems written in C. This benchmark was also used by 5 primary studies to evaluate machine learning based approaches on clones in C programming language. Dataset GCJ (GoogleCodeJam) is used for semantic clone detection and cross-language clone detection. This dataset consists of 1669 java projects from 12 different competition problems. These problems are part of an online programming competition held annually by Google.

RQ5.1: What are various existing clone detection tools used in primary studies?

Table 18 shows the various clone detection tools which are used in primary studies. These clone detection tools are used for two purposes in these studies. (1) To detect clones from subject systems and then use clone detection results as datasets for applying machine learning algorithms. (2) To compare proposed approach of the study with existing clone detection tools for checking effectiveness of the proposed approach. NiCad is used

Table 16
Subject systems used in primary studies.

Code clone area	Subject system used	Count	Citation	Language
Code clone detection	ANTLR, Apache Ant, ArgoUML, CAROL, Dnsjava, Hibernate, JHotdraw	1	[20]	Java
	JDK	2	[20,25]	Java
	Abyss	1	[35]	C
	JEdit, Ant, Maven, OpenNLP 1.8.1, Commons Lang 3-3.7	1	[25]	Java
	Weltab	2	[30,35]	C
	SNNS	1	[30]	C
	Linux 2.6.6/arch, Linux 2.6.6/net, Linux 2.6.6/kernel, Linux2.6.6/crypto	1	[32]	C
	Debian Linux	1	[34]	C
	Bitmessage	1	[33]	Python
	hmmer, sphinx3 and bzip2 from SPEC2006 benchmark suite	1	[26]	C
Semantic clone detection	Supple, Netbeans-javadoc, Eclipse-ant, EIRC, Sample j2sdk1.4.0-javax-swing, Sample eclipse-jdtcore	3	[40,44,45]	Java
	Apache commons imaging, Apache commons math3, Catalano Framework, Colt, Weka (without gui)	1	[48]	Java
	Python projects on public GitHub	1	[38]	Python
Binary code clone detection	hmmer, sphinx3 and bzip2 from SPEC2006 benchmark suite	2	[49,50]	C
Source code representation for clone detection	ANTLR, ArgoUML, JHotdraw, Ant, Hibernate, Hadoop, Maven, Pmd, Tomcat, Qualitas.class Corpus, Apache commons libraries	1	[53]	Java
Code clone classification	git v1.7.9-rc1, xz 5.0.3, bash 4.2, e2fsprogs 1.41.14	1	[58]	C
	IJaDataset 2.0	2	[57,59]	Java
Clone recommendation for refactoring	Apache Ant, ArgoUML	1	[61]	Java
	JFreechart, Axis2, Eclipse.jdt.core, Elastic Search, JRuby	1	[62]	Java
	Lucene	2	[61,62]	Java
	clawsmail, dovecot, emacs, fdisk, ffmpeg, lighttpd, smalltalk	1	[60]	C
Clone evolution	Wget, conky	1	[65]	C
	ProcessHacker	1	[65]	C#
	iTextsharp			
	JEdit	2	[64,65]	Java
	ArgoUML	2	[63,64]	Java
	Dnsjava	1	[65]	Java
Evaluating clone detection precision	Java 8 class library	1	[67]	Java
	JFreechart	1	[66]	Java
	Django	1	[66]	Python
Prediction of harmfulness of cloning operations	JEdit, ArgoUML,	2	[72,73]	Java
	JFreechart, Tuxguitar	2	[70,72,73]	Java
	Clawsmail, Dovecot, Emacs, Fdisk, Ffmpeg, Lighttpd, smalltalk	1	[71]	C
	Microsoft software projects in C#	2	[69,70]	C#
Prediction of code clone quality	Xorriso, Smalltalk, Bison	1	[74]	C
Prediction of risky clones	JEdit	1	[76]	Java

in most of the primary studies for generating datasets of clones whereas Deckard has been used for comparison with most of the clone detection approaches based on machine learning.

Summary results for RQ5

Subject systems written in Java are widely used in machine learning based research in code clones. Most of the research has been carried out using open-source subject systems. Studies related to prediction of risky clones and code clone quality used only Java and C subject systems respectively. Existing machine learning based approaches in code clone research can be evaluated with industry-related datasets in various programming languages. Nicad, Deckard and SourcererCC are widely used tools in studies related to code clones where machine learning has been applied. In terms of publicly available Java clone datasets, BigCloneBench is better than GoogleCodeJam in terms of size and number of functionalities supported. BigCloneBench also gives a large percentage (98%) of semantic clones. In C language, only a single publicly available clone dataset named OJClone is used in primary studies.

Table 17
Dataset used in primary studies.

Dataset used in primary studies	Programming language	Area of code clone research	Machine learning algorithm used	Study reference
BigCloneBench	Java	Code clone detection	K-means Clustering Recursive Autoencoders Convolution Neural Network Deep Neural Network	[22,23,25,29,31]
		Semantic clone detection	Deep Neural Network Tree-based Convolution Graph Neural network Tree based LSTM Xgboost Random Forest Rotation Forest	[36,37,39,41,42,46,47]
		Source code representation for clone detection	Recurrent Neural Network Recursive Neural Network Neural Network	[54–56]
		Evaluating clone detection Precision	Deep Neural Network	[68]
GoogleCodeJam	Java	Semantic clone detection	Deep Neural Network Graph Neural network	[39,46]
	Java, C# and Python	Cross language clone detection	Deep Neural Network	[52]
OJClone	C	Semantic clone detection	Tree-based Convolution Tree-based Long Short-Term Memory	[37,41,47]
		Source code representation for clone detection	Neural Network Recurrent Neural Network	[54,56]

Table 18
Clone detection tools used in primary studies.

S. No.	Clone detection tool	Purpose of Use	Citation
1.	ConQAT [114]	Clone detection	[70,73,75]
2.	NiCad [115]	Clone detection Comparison	[22,48,57,59,60,64,65,67,68,72,73] [23,25,31,36,42]
3.	Deckard [86]	Clone detection Comparison	[57,59] [23,26,31,36,37,41,42,46,47,55]
4.	iClones [87]	Clone detection Comparison	[57,59,61,66,68] [42]
5. 6.	CCFinderX [85] CCFinder [85]	Clone detection Comparison	[57,59] [42]
7.	SourcererCC [84]	Clone detection Comparison	[36,57,59,62,68] [23,31,36,37,41,42,47]
8.	CloneDr [116]	Clone detection	[63]
9.	Oreo [36]	Clone detection Comparison	[68] [23,29]
10.	CCLearner [31]	Comparison	[23,25,29]
11.	CCAligner [79]	Comparison Clone detection	[25] [68]
12.	CloneWorks [117]	Comparison Clone detection	[23,36] [59,68]
13. 14. 15. 16.	SimCad [118] Simian [119] Ctcompare [120] CDLH [47]	Clone detection Clone detection Clone detection Comparison	[59,68] [59] [59] [23,37,41,46,47]

4.6. RQ6: Which machine learning techniques have been used in semantic clone detection?

We observed that machine learning algorithms are used most frequently in semantic clone detection as compared to other code clone research areas (see Figs. 8 and 9). There are 13 primary studies related to semantic clone detection. In those 13 studies, conventional machine learning algorithms are used 38 times, deep learning algorithms are used 11 times and clustering algorithms are used 1 time. Table 19 lists these machine learning algorithms that are used in various semantic clone detection studies. As observed, most studies i.e., 10 out of 13 used deep learning algorithms for semantic clone detection. Three studies [40,42,44] used only conventional machine learning algorithms for semantic clone detection. Ensemble approach/classifier

Table 19
ML algorithms used in semantic clone detection.

Machine learning algorithm	Count	Citation
Deep Neural Network	2	[36,46]
Tree Based Convolution Network	2	[37,41]
Convolution Neural Network	2	[43,45]
Tree Based LSTM	1	[47]
Recursive Neural Network	1	[48]
Recurrent Neural Network	1	[38]
Graph Convolution Network	1	[48]
Graph Neural Network	1	[39]
Ensemble Classifier	2	[40,44]
XgBoost, Decision Tree, Naïve Bayes, Support Vector Machine, Multilayer Perceptron, Random Tree, Random Forest, Rotation Forest, Linear Discriminant Analysis (LDA), Instance Based Learner (IBK), Bagging, Logit Boost, Random Committee, K* Classifier, Extra Trees	1	[42]
RBF Network, Liblinear, RBF Classifier, IBK, MLP, Naïve Bayes, Sequential Information Bottleneck (sIB) Clustering	1	[45]
Logistic Regression and Random Forest	1	[36]

using majority voting among Naïve Bayes, Random Forest, LibLinear SVM, Instance Based Learner, Logit Boost, J48, Bagging, Random subspace [121], Random Committee and Rotation forest classifiers algorithms was used in study [40] whereas Instance Based Learner (IBK), Random Committee, Bagging, Rotation Forest and Random Forest classifiers are used as ensemble approach in [44]. Sheneamer et al. [45] combined clustering with deep learning and other conventional machine learning algorithms to detect semantic clones. Study [36] compared Logistic regression, random forest and deep neural network to select a suitable model for clone detection.

Selection of code representation plays an important role to capture semantics of source code. Current machine learning based semantic clone detection approaches used metrics [36], tokens [43], AST [38–40,42,44,45,47], CFG [46,48], DFG [46], PDG [40,42,44,45] and BDG [40,44,45] to learn semantics of source code. Two studies [37,41] enhanced AST with API(Application Programming Interface) call information [37] and tokens [41] to capture more information about the semantics of source code.

After analysing evaluation results of semantic clone detection studies, we observed that Deep Neural Network based tool DeepSim [46] achieved 98% F1-score on BigCloneBench for detecting semantic clones whereas Tree-based Convolution Neural Network based TBCCD [41] achieved 96% F1-score and Graph Neural Network based, FA-AST+GMN [39], clone detection approach achieved 95% F1-score on BigCloneBench. So, these three approaches perform better on BigCloneBench as compared to other machine learning based approaches for semantic clone detection. Performance of these machine learning based approaches on other datasets is presented in detail in Section 4.7 in Table 21.

Summary results for RQ6

Deep learning algorithms are applied in most of the primary studies related to semantic clone detection whereas more conventional machine learning algorithms are experimented in studies related to semantic clone detection. Deep Neural network based clone detection approaches achieved better performance on BigCloneBench as compared to other machine learning algorithms.

4.7. RQ7 Which machine learning algorithm gives better results when compared using the same dataset for the same problem?

To answer the above question, we have reported the results of those primary studies where more than one machine algorithm has been applied with the same dataset. Tables 20–24 summarizes the results of those primary studies. The tables include the machine learning algorithms compared in the study, values of evaluation parameters and subject systems used in the study. Tables 20–24 also includes those studies where a machine learning based approach was compared with any traditional clone detection technique.

Rongrong et al. [60] compared Naïve Bayes, Decision tree and Bayesian Network to select the best classifier for recommending refactoring clones (Table 20). The comparison results revealed that Decision tree performs best among other machine learning algorithms in terms of accuracy. Yue et al. [62] compared AdaBoost, Decision tree C4.5, Random forest, SMO and Naïve Bayes to know suitability for recommending refactoring clones. The results show that AdaBoost performed best in terms of F1-score within cross project setting and gave second highest value of F1-score i.e., 83% within-project setting whereas Naïve Bayes performed poorly in terms of F1-score. AdaBoost performed better as compared to decision tree C4.5 for recommending clones for refactoring when the same set of features are used to train the classifier in the cross-project setting [62].

Table 21 summarizes comparison studies related to semantic clone detection using machine learning. Sheneamer et al. [42] compared Xgboost, Random forest and Rotation forest for detection of semantic clones. Xgboost obtained highest precision and Rotation Forest obtained highest recall whereas Random Forest achieved highest F1-score when compared for detecting semantic clones. Wei and Li [47] proposed an approach named CDLH that used tree-based LSTM for clone detection. CDLH performed better than DLC (Deep learning for code clones) [20], Deckard and SourcererCC in terms of F1-score. DLC is the first deep learning-based clone detection approach that used RNN to learn token embeddings of source code and then used Recursive autoencoder to learn vector representation of AST. DeepSim performed better than Deckard, CDLH and DLC on BigCloneBench. Yu et al. [41] proposed an approach named TBCCD (Tree based Convolution for clone detection). This approach used tree-based convolution over a token enhanced AST to detect semantic clones. They compared the TBCCD with CDLH, DLC, Deckard and SourcererCC and results of comparison show that TBCCD performs better than all other techniques as shown in Table 21. Chen et al. [37] compared TBCAA(Tree-based convolution over API-enhanced AST) with CDLH, the approach based on tree-based LSTM. There is an increase of F1-score from 82% to 94% on BigCloneBench and from 57% to 96% on OJClone. They also compared TBCAA with DLC, Deckard and SourcererCC. Wang et al. [39] constructed a graph representation of programs named FA-AST (Flow-augmented abstract syntax tree), that adds control and data flow edges to the original AST. The approach applied two types of graph neural networks, graph matching network (GMN) and gated graph neural network (GGNN) on FA-AST to identify clone pairs from the source code. The study compared the approach with Deckard, DLC [20] and ASTNN [54] and showed that FA-AST+GMN performed better than other approaches. Zhang et al. [54] proposed AST-based neural network(ASTNN) for representing source code. The approach decomposed large AST of code fragments into sequence of small statement trees and performed tree based neural embeddings on all statement trees. They compared the approach with CDLH [47] and DLC [20] and results show that ASTNN performs better than other machine learning based clone detection approaches on OJClone and BigCloneBench datasets.

Table 22 summarizes the comparison of different machine learning based clone detection techniques. Cesare et al. [34] used Naïve Bayes, Multilayer Perceptron, C4.5 and Random Forest for detecting package level clones. The results of comparison show that Random Forest gives highest values of precision, recall and

Table 20
Clone recommendation for refactoring studies comparing multiple machine learning algorithms.

Study	Algorithms/Tools compared	Outcome		Subject system/Dataset/Data source used
		Accuracy		
[60]	Naïve Bayes	78.8%		ffmpeg, lighttpd, Smalltalk,
	Decision Tree	95.9%		fdisk, emacs, dovecot,
	Bayesian Network	94.8%		clawsmail
		F1-Score (Cross- Project)	*F1-Score (With-in Project)*	
[62]	AdaBoost	76%	83%	
	C4.5	58%	84%	Axis2, Eclipse.jdt.core, Elastic
	Random Forest	61%	84%	Search, JFreeChart, JRuby,
	SMO	53%	78%	Lucene
	Naive Bayes	53%	64%	

Table 21
Comparison studies related to semantic clone detection using machine learning.

Study	Algorithms/Tools compared	Outcome						Subject system/Dataset/Data source used
		Precision		*Recall*		*F-Measure*		
[42]	Xgboost	88%		85.7%		86.82%		
	Random Forest	87.53%		87.45%		87.47%		BigCloneBench
	Rotation Forest	87.38%		87.55%		87.45%		
		BigCloneBench			*OJClone*			
		Precision	*Recall*	*F1-Score*	*Precision*	*Recall*	*F1-Score*	
[47]	CDLH	92%	74%	82%	47%	73%	57%	BigCloneBench and OJClone
	SourcererCC	88%	2%	3%	7%	74%	14%	
	Deckard	93%	2%	3%	99%	5%	10%	
	DLC [20]	95%	1%	1%	71%	0%	0%	
		BigCloneBench			*Google Code Jam*			
		Precision	*Recall*	*F1-Score*	*Precision*	*Recall*	*F1-Score*	
[46]	DeepSim	97%	98%	98%	71%	82%	76%	BigCloneBench Google Code Jam
	Deckard	93%	2%	3%	45%	44%	44%	
	DLC [20]	95%	1%	1%	20%	90%	33%	
	CDLH [47]	92%	74%	82%	–	–	–	
		BigCloneBench			*OJClone*			
		Precision	*Recall*	*F1-Score*	*Precision*	*Recall*	*F1-Score*	
[41]	TBCCD	97%	96%	96%	99%	99%	99%	BigCloneBench and OJClone
	CDLH [47]	92%	74%	82%	47%	73%	57%	
	Deckard	93%	2%	3%	99%	5%	10%	
	SourcererCC	88%	2%	3%	7%	74%	14%	
	DLC [20]	95%	1%	1%	71%	0%	0%	
		BigCloneBench			*OJClone*			
		Precision	*Recall*	*F1-Score*	*Precision*	*Recall*	*F1-Score*	
[37]	TBCAA	95%	93%	94%	97%	95%	96%	
	CDLH	92%	74%	82%	47%	73%	57%	
	Deckard	93%	2%	3%	99%	5%	10%	BigCloneBench and OJClone
	SourcererCC	88%	2%	3%	7%	74%	14%	
	DLC [20]	95%	1%	1%	71%	0%	0%	
		BigCloneBench			*OJClone*			
		Precision	*Recall*	*F1-Score*	*Precision*	*Recall*	*F1-Score*	
[54]	ASTNN [54]	99.8%	88.4%	93.8%	98.9%	92.7%	95.5%	BigCloneBench and OJClone
	CDLH [47]	92%	74%	82%	47%	73%	57%	
	RAE+	76.4%	59.1%	66.6%	52.5%	68.3%	59.4%	
		BigCloneBench			*GoogleCodeJam*			
		Precision	*Recall*	*F1-Score*	*Precision*	*Recall*	*F1-Score*	
[39]	FA-AST+ GMN	96%	94%	95%	99%	97%	98%	BigCloneBench and GoogleCodeJam
	FA-AST+ GGNN	85%	90%	88%	96%	100%	97%	
	Deckard	93%	2%	3%	45%	44%	44%	
	DLC [20]	95%	1%	1%	20%	90%	33%	
	ASTNN [54]	92%	94%	93%	98%	93%	95%	

F1-score in shared package clone detection whereas Decision tree C4.5 performs better than all other classification algorithms in embedded package clone detection. Li et al. [31] compared CCLearner with NiCad, Deckard and SourcererCC. CCLearner is a token-based tool that uses deep learning to detect clones. The comparison results show that CClearner achieved the highest C-score i.e., 93% and second highest precision i.e., 93%. In terms of time cost, CCLearner was slower than NiCad and SourcererCC.

Zeng et al. [23] compared their fast clone detection method that is based on weighted recursive autoencoders with other traditional and deep learning based clone detection methods. The results show that their approach achieved 99.62% precision which is comparable to NiCad that obtained 99% precision whereas deep learning-based clone detection methods, Oreo achieved 89.5%, DeepSim obtained 97%, CCLearner obtained 93% and CDLH obtained 92% precision.

Table 22
Comparison studies related to code clone detection using machine learning.

Study	Algorithms/Tools compared	Outcome						Subject system/Dataset/Data source used
		Shared package clone detection			*Embedded package clone detection*			
		Precision	*Recall*	*F1-Score*	*Precision*	*Recall*	*F1-Score*	
[34]	Naïve Bayes	47%	57%	52%	10%	94%	18%	Debian Linux
	Multilayer Perceptron	80%	26%	40%	75%	43%	54%	
	C4.5	85%	68%	76%	89%	75%	81%	
	Random Forest	89%	70%	78%	89%	72%	80%	
		Precision			*C-Score*		*Time*	
[31]	CCLearner	93%			93%		47 m	*BigCloneBench*
	NiCad	68%			76%		33 m 40 s	
	Deckard	71%			77%		4 h 24 m	
	SourcererCC	98%			88%		13 m 15 s	
		Precision						
[23]	[23]	99.62%						BigCloneBench
	NiCad	99%						
	Deckard	34.8%						
	SourcererCC	97.8%						
	CloneWorks	98.7%						
	Oreo [36]	89.5%						
	DeepSim [46]	97%						
	CCLearner [31]	93%						
	CDLH [47]	92%						
		Precision						
[25]	TECCD	88%						BigCloneBench
	NiCad	86%						
	CCLearner	84%						
	CCAligner	72%						

Table 23
Comparison studies related to code clone classification using machine learning.

Study	Algorithms/ Tools compared	Outcome	Subject System/Dataset/Data source used
		Accuracy	
[59]	J48 decision tree	84.82%	IJaDataset 2.0
	Random Forest	84.47%	
	Random Tree	79.84%	
	Naïve Bayes	83%	
	Bayesian network	81.79%	
	Logistic regression	85.06%	
	Stochastic Gradient	84.85%	
	K* Classifier	81.79%	
	Multiclass Classifier	85.06%	
	Decision Table	85%	
	ANN	87.4%	
	C4.5	84%	

Gao et al. [25] showed that their tool TECCD obtained highest precision on BigCloneBench dataset when compared with CCLearner and NiCad. TECCD used tree embedding to get node vectors of intermediate nodes in the AST. Then, the approach measured the Euclidean distance between tree vectors to find clones.

Mostaeen et al. [59] showed that they have achieved best results for automatic code clone validation using Artificial Neural Network when comparison was done with other machine learning algorithms (Table 23). ANN achieved highest accuracy i.e., 87.4% whereas Random Tree achieved lowest accuracy i.e., 79.84%.

Nafi et al. [52] compared their approach CLCDSA with ASTLearner [51] to detect cross language clones (Table 24). CLCDSA used deep neural network to learn features from labelled data and then detect cross language clones whereas ASTLearner used LSTM-based neural network to predict clones. As shown in Table 24, CLCDSA achieved better precision, recall and F1-score than ASTLearner.

Figs. 13, 14 and 15 compare the results of different primary studies that applied machine learning for code clone detection using dataset, BigCloneBench, OJClone and GoogleCodeJam. We observed that recurrent neural network-based approach, ASTNN [54] achieved better precision and deep neural network based DeepSim [46] achieved better recall and F1-score as compared to other approaches of clone detection on BigCloneBench. Tree based convolution approach [41] performed well on OJClone dataset and Graph neural network based approach [39] performed well on Google Code Jam for clone detection.

Inferences from RQ7

AdaBoost performed better than decision tree, Random Forest, SMO and Naïve Bayes for recommending clones for refactoring.

Datasets BigCloneBench, GoogleCodeJam and OJClone are used in semantic clone detection. BigCloneBench is used in most of the studies related to semantic clone detection. Deep Neural Network based semantic clone detection approach (DeepSim) achieved better performance (98% F-score) on BigCloneBench, Tree based Convolution neural network based approach (TBCCD) performed better (99% F-score) on OJClone whereas Graph Neural network based approach (FA-AST + GMN) achieved highest (98%) F-score on GoogleCodeJam when compared with other machine learning based clone detection approaches.

Random Forest achieved better performance for package clone detection as compared to Naïve Bayes, MLP and Decision Tree. Recursive Autoencoders based clone detection approach [23] achieved highest (99.62%) precision as compared to approaches based on deep neural network (CCLearner, DeepSim and Oreo) and Tree-based LSTM (CDLH) on BigCloneBench.

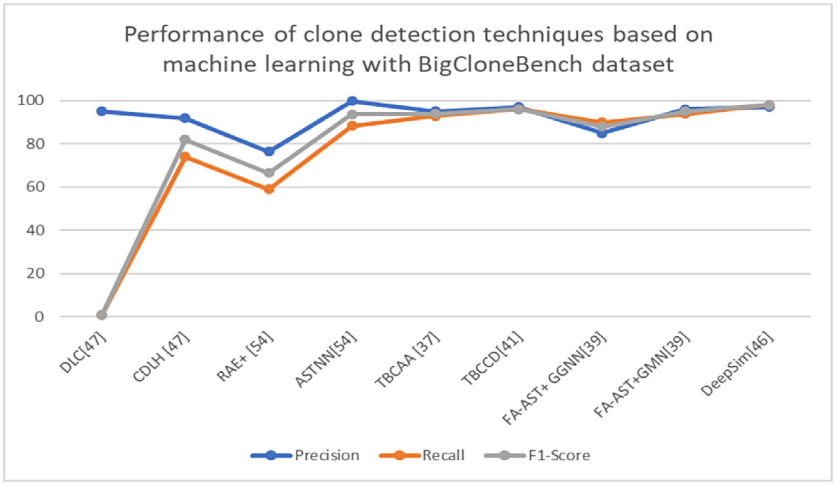

Fig. 13. Comparison of clone detection techniques based on machine learning with BigCloneBench.

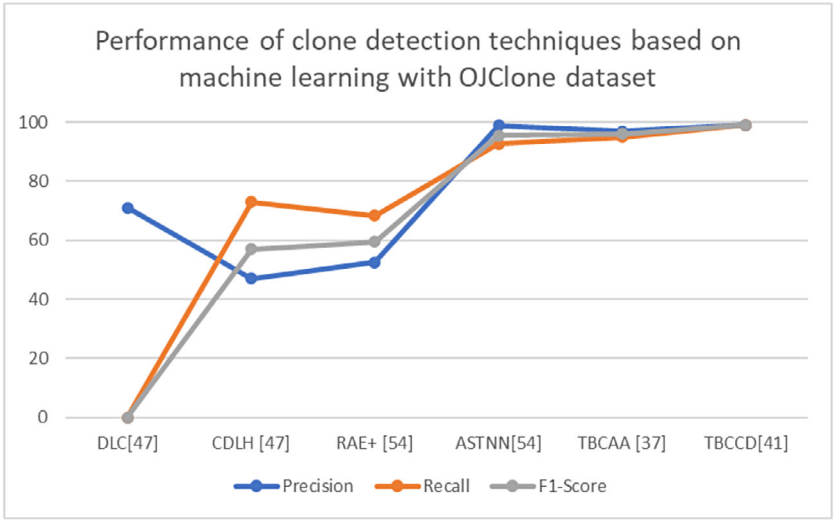

Fig. 14. Comparison of clone detection techniques based on machine learning with OJClone dataset.

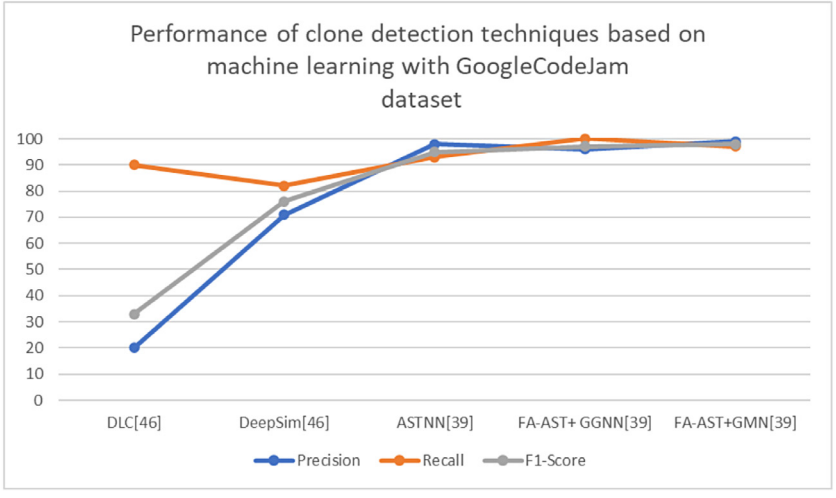

Fig. 15. Comparison of clone detection techniques based on machine learning with GoogleCodeJam.

Table 24
Comparison studies related to cross language clone detection using machine learning.

Study	Algorithms/ Tools compared	Outcome			Subject system/Dataset/Data source used
[52]		*Precision*	*Recall*	*F1-Score*	GoogleCodeJam AtCoder, CoderByte
	CLCDSA [52]	61%	93%	71%	
	ASTLearner [51]	18%	81%	30%	

An Artificial Neural Network achieved highest accuracy as compared to other conventional machine learning algorithms for code clone validation.

Summary results for RQ7

Performance of various machine learning algorithms is compared in certain areas of code clone research such as code clone detection, semantic clone detection, cross language clone detection, code clone classification/validation and clone recommendation for refactoring. But when to apply deep learning and when to choose deep learning over conventional machine learning algorithms is an open area of research.

5. Discussion

We studied various primary studies in order to know the extent of research that has been carried out in the field of code clones using machine learning techniques. The main findings of the work and guidelines for future work are discussed in the following section. All the findings are reported in the order of the research questions of the SLR.

RQ1- Deep learning techniques have become more popular as compared to other machine learning techniques to detect hard to detect clones, especially Type- 4 clones. Empirical studies are required to compare the effectiveness of deep learning and conventional machine learning approaches for clone detection. Studies related to binary code clone detection have used deep learning and clustering algorithms for clone detection. Cross language clone detection and then recommending cross-language clones for refactoring using machine learning algorithms has also emerged as a future area of research that has been explored very less. AST is widely used for representing source code for clone detection using machine learning. However, code similarities can also be learned from other representations of source code like tokens, control flow graph and bytecode. Tufano et al. [53] mentioned that combined models that use multiple representations of source code can be effective in code clone detection. Only one study [56] assesses the potential of visualization and transfer learning in code clone detection and more research is required in the same direction to solve the problem of availability of large datasets required for training of machine learning algorithms. Studies to assess whether code clone detection can benefit from multitask learning, can also be conducted.

Mostaeen et al. [57,59] and Yang et al. [58] developed tools, CloneCognition and FICA respectively, for automatically validating code clones that are detected by a clone detection tool. We invite the research in the direction of creating user specific validated clone dataset using these tools that can be used as benchmark to evaluate clone detection tools. Studies [60–62] show that Decision Tree and AdaBoost machine learning algorithms perform well for recommending clones for refactoring in Java and C. We call for assessing the potential of deep learning in recommending suitable clones for refactoring in different programming languages. Also, user studies are required to be conducted to obtain the developer's opinion on the recommended clones for refactoring to know the validity of refactoring suggestions in a real scenario. K-means clustering, Decision Tree and Siamese Deep Neural Network are used to measure precision of clone detection tools on Java subject systems. Studies using other machine learning algorithms like Random Forest are required to know the strength of machine learning in measuring precision of clone detection tools on subject systems written in different programming languages.

In the future, we need empirical studies that can evaluate precision of modern clone detection tools using machine learning based approaches [66–68] of calculating precision. Bayesian Network is used in prediction of harmfulness of cloning operations, prediction of code quality and prediction of risky clones whereas decision tree is used in prediction of risky clones and bug fixes in clones. These prediction models require a large dataset and time to train the model. Building cross-project predictor can help in this direction. Studies involving developers for evaluating prediction results and comparative studies showing the effectiveness of Bayesian network, decision tree, SVM and other machine learning algorithms in prediction of harmfulness of cloning operations are required. Such studies can help to integrate effective prediction models into an IDE so that risk of potential bugs can be avoided at development time.

RQ2- Decision Tree, Random Forest, Bayesian Network and Naïve Bayes are most popular machine learning algorithms in code clone research. Decision tree classifier and Bayesian network have been used in predicting risky clones, refactorable clones and for validating code clones. But the decision tree classifier gives better results as compared to Bayesian network in these areas of code clone research. Which machine learning algorithm is more suitable in which area is still an open issue because of lack of comparison between various machine learning algorithms in areas like evaluating clone detection precision, prediction of harmfulness of cloning operations. Also, performance of single machine learning algorithms is checked in research areas like prediction of code clone quality and detection of bugs in clones.

Deep learning algorithms are mostly used in the area of code clone detection, especially in semantic clone detection. This shows the capability of deep learning to deal with semantic clones and shows the future possibility of using deep learning algorithms in prediction of bugs in semantic clones and prediction of refactorable semantic clones. Future studies can evaluate the potential of deep learning in prediction of bugs in clones, refactorable clones and harmfulness of cloning operations.

RQ3- Different code features have been used in different studies. The choice of features mainly depends on the area of code clone research. Clone detection techniques used syntactical and semantic features of code and these features are derived from AST, PDG and Bytecode dependency graph of the source code. All these features/metrics represent data, information and control flow of the source code. Metrics related to cloned code, clone context and clone history are used in studies related to prediction of harmful clones and refactorable clones. To train machine learning models, the features/metrics extracted from source code play an important role. We have observed that there is a lack of tools which can enable fast and automatic generation of metric values. All studies mentioned about the features used to train the model but lacks a standard set of tools or methodologies used to get

these metrics values. The future research towards building of metric extraction tools related to all types of features of source code can save time and help in comparison of different prediction models that are based on the same set of features/metrics.

RQ4- Most popular metrics of evaluating machine learning models are precision, recall and F1-score. The studies which are evaluated using BigCloneBench may report different recall from actual recall. BigCloneBench does not label all true clones and the evaluation results are conducted on partially labelled dataset. Similarly, the precision evaluation method is not universal among all studies due to manual effort involved to validate all clones.

RQ5- Most of the primary studies have used subject systems written in Java and C. The reason behind this is the availability of clone benchmarks, BigCloneBench and OJClone. We urge the need of benchmarks in programming languages like C#, Python to increase the research in code clones in these languages which can assist in maintenance tasks like bug detection and refactoring. Also, evaluation of machine learning based clone detection techniques on programming languages other than Java and C is required to be done to assess their effectiveness. Steidl and Gode [75] mentioned the challenge of creating a large and balanced data set for training machine learning algorithms. Availability of large and balanced datasets of clones can increase recall and precision of machine learning prediction models. Future work towards creating large labelled data sets of clones can invite the use of machine learning algorithms in different research fields of clones like semantic clone detection, bug detection in clones and recommending refactoring of clones.

NiCad, iClones and SourcererCC are popular tools to detect clones from Java subject systems which are further used as dataset of machine learning models. Saini et al. [36] highlighted the future possibility of training deep neural networks using clones detected by multiple clone detection tools to improve recall and precision of semantic clone detection. Evaluation and comparison of machine learning based clone detection tools like Oreo, CCLearner, CDLH etc. are required to assess their performance for building large clone datasets.

RQ6- Deep learning and conventional machine learning algorithms are used to detect semantic clones. Deep learning based approaches achieved high values of precision, recall and F1-score as compared to other machine learning algorithms for detecting semantic clones. Most of the semantic clone detection studies used AST representation of source code to learn semantics. Semantic learning can be improved by enhancing AST with tokens [41], control and data flow information [39]. Approaches [39, 41] that enhance AST with code tokens or control and data flow information perform better than approaches that use only AST representation [20,47]. Empirical evaluation of existing machine learning based semantic clone detection approaches can help to understand pros and cons of various code representations and machine learning algorithms.

RQ7- Studies comparing machine learning based approaches are lacking in various areas of code clone research like binary code clone detection, clone evolution, prediction of harmfulness of cloning operations and evaluating precision of clone detection tools. Future work can be carried out to fill the gap in this direction.

6. Gaps and limitations of current research

This section presents the holistic view of major gaps and limitations of current research derived from the discussion of the review.

1. *Lack of availability of large datasets:* Existing machine learning based clone detection techniques used three datasets available for Java and C programming language namely, BigCloneBench, GoogleCodeJam and OJClone. But there are no publicly available datasets for languages like Python, C++ etc. Standard dataset for using machine learning in cross-language clone detection is not available. Also, there is no clear taxonomy about cross-language clones. Similarly, semantic clone detection using machine learning faces the challenge of lacking large semantic clone datasets. There is a need to build large semantic clones and cross-language clone datasets for different programming languages.
2. *Performance of existing machine learning based techniques for different programming languages and clone types:* Existing machine learning based clone detection techniques should be evaluated for different clone types and subject systems written in different programming languages. For example, K-means clustering was used to detect up to Type-3 clones. It can be explored for Type- 4 clone detection [22]. DBSCAN clustering was used to detect Type-1 and Type-2 clones [33] in subject systems written in Python. The effectiveness of this approach can be tested on subject systems written in other programming languages, like Java. Some studies [36,40,68] are applicable to Java subject systems only. Training and evaluation of machine learning models is done on Java methods. Adaptation of these models to other languages is required to be tested. Automatic precision calculation techniques based on machine learning are evaluated using open-source projects written in Java or Python. Implementation of these approaches in different programming languages for different clone granularities can be tested on commercial software.
3. *Setting of hyper-parameters:* Performance of machine learning-based clone detection techniques depends on setting of hyper-parameters like number of hidden layers, number of activation units in each layer and number of iterations. Different settings of these hyper-parameters can change the recall and precision of machine learning based clone detectors. It can effect the comparison of machine learning based clone detectors. Tuning of these hyper-parameters is required to achieve desired performance of machine learning based clone detection techniques.
4. *Type of learning method used:* Learning method used in machine learning based approaches can be supervised or unsupervised. For example DeepSim [46] and CDLH [47] used supervised method whereas Zeng et al. [23] and TECCD [25] used unsupervised method. Empirical studies assessing the effect of these learning methods on recall and precision can be conducted.
5. *Evaluation of machine learning based clone detectors:* Most machine learning based clone detection studies used BigCloneBench for measuring recall. Since BigCloneBench does not label all clone pairs, therefore, actual recall of these machine learning based clone detection approaches can be different. Calculation of precision using random sampling is also subject to human bias. The studies try to mitigate this problem by involving more than two judges for checking valid clones. Machine learning based automatic approaches for calculating precision can be tested to further mitigate the problem of measuring precision.
6. *Availability of existing machine learning based clone detectors for public use:* Most of the clone detectors based on machine learning are not available for public use. Therefore, comparison results of new machine learning based clone detectors [37,41] with existing ones [20,47] are based

on clone detection results published in original studies. Comparative/empirical studies can be conducted if these machine learning based clone detectors become available for researchers and industry.

7. *Selection of suitable source code representation and machine learning algorithm:* As we have observed that use of machine learning has increased for clone detection. Selecting a machine learning algorithms is a challenging problem because training time and performance of algorithms on unseen data varies. Training time can be decreased if the algorithm supports parallelism. For example, convolution neural network takes less time for training as compared to recurrent neural network because convolution steps can be performed in parallel [37]. Performance of machine learning algorithm on unseen data depends on the type of source code representation used. Current machine learning based clone detection approaches used different representations of source code like tokens, AST, control flow graph. But approaches representing code fragments as sequences of tokens may not be generalized to source code having out-of-vocabulary tokens [37] because vocabulary of tokens is unlimited. There is only one study that assesses the use of visual representation of source code for clone detection. Future research that helps to meet the challenge of selecting a suitable source code representation and machine learning algorithm is required.
8. *Challenges to build datasets for different areas of code clone research:* Effectiveness of machine learning based approaches is limited by the size and quality of the training dataset. Building large and representative dataset is a challenging problem in different areas of clone research where machine learning has been applied. For example, there are no datasets publicly available for recommending clones for refactoring, predicting risky clones and bugs in clones.

   Quality of the dataset used to train machine learning models is affected by the clone detection tool used to build the dataset. A clone detection tool can generate a different set of clones with different configuration settings. Some studies like [36] have used a single clone detection tool (SourcererCC) to get a clone dataset for training the machine learning algorithm. Use of a single clone detection tool limits the variety of clones in the dataset. Therefore, multiple clone detectors need to be used so that if one clone detector missed some clones, the other clone detector can find the same. This can help to generate training dataset containing all types of clones.

   Machine learning algorithms are used for recommending clones for refactoring but training data for these algorithms is based on historical refactoring of clones. Labelling clone instances with its refactoring history increases training cost and effort. This cost can be decreased either using user specific labelling of clones based on refactoring benefits, cost and risks or using cross-project testing.

   Detection of bugs in clones using machine learning approaches requires a larger and balanced dataset of inconsistent clones for training machine learning models which is not available. Future work to fill this gap can help researchers to evaluate more machine learning algorithms for detection of bugs in clones.
9. *Issues related to prediction of harmfulness of cloning operations:* Studies related to prediction of harmfulness of cloning operations have used Bayesian Network classifier on C# and Java subject systems. Effectiveness of other machine learning algorithms needs to be evaluated. Also, effectiveness of current approaches on cross-prediction is not desirable. Research towards building a universal prediction model needs to be carried out so that prediction of harmful cloning operations can be done in new repositories (without version history) or repositories having insufficient training data.

7. Threats to validity

There are few threats that can affect the validity of systematic literature review. The first threat is related to the selection of all relevant articles. While we have thoroughly examined the retrieved literature from data sources, we may have missed some publications related to code clone research using machine learning. Use of secondary data sources, snowballing process and cross-checking of selected articles by second author can mitigate the effect of this threat. The other threat is related to quality assessment and data extraction process. First author performed the quality check using a set of questions verifying the presence of required information from selected articles and extract data using data extraction form. The second author carefully checked the quality assessment criteria, data extraction form and cross-checked the extracted data.

8. Conclusion

This paper reported the systematic literature review which is based on 57 primary studies related to use of machine learning in code clone research. We formulated the research questions to divide the existing literature into various subareas of code cloning to highlight the current status of machine learning related research in these areas. We reported how previous research was conducted in these fields, i.e. (i) what machine learning algorithms are used, (ii) what are the training strategies, validation strategies and evaluation metrics that have been used, (iii) which code features have been used, (iv) which datasets are popular in these studies, (v) which machine algorithms have been used for semantic clone detection, (vi) which machine learning algorithm gives better results when compared with other algorithms and (vii) how existing clone detection tools are being used in studies related to machine learning. We argue the need of standardizing clone datasets and feature extraction tools for comparing and analysing the strength of machine learning algorithms in code clone research. Empirical studies related to use of machine learning algorithms in various code clone research areas can help to take benefits from the recent developments in the machine learning field, especially from deep learning. Also, applications of machine learning techniques in clone detection and management of clones in programming languages other than Java and C are required to be assessed. We conclude that our analysis represents the current state of research related to the use of machine learning in code cloning and reports important findings which can help researchers to find the future possibilities in the field.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

We are very thankful to the editor and appreciate the timely reply of queries during the review process. We pay a sincere gratitude to the anonymous reviewers for their valuable comments and suggestions which guided us to make the final manuscript.

Appendix A. Acronyms

API	Application Programming Interface
ANN	Artificial Neural Network
ARIMA	Autoregressive Integrated Moving Average
AST	Abstract Syntax Tree
ASTNN	Abstract Syntax Tree based Neural Network
AUC	Area Under Curve
BDG	Bytecode Dependency Graph
BP-NN	Back Propagation Neural Network
CDLH	Clone Detection with Learning to Hash
CCDLC	Combining Clustering with Deep Learning Classification
CFG	Control Flow Graph
CLCDSA	Cross Language Code Clone Detection using Syntactical Features and API Documentation
DLC	Deep learning code fragments
DTW	Dynamic Time Wrapping
FA-AST	Flow Augmented-Abstract Syntax Tree
FICA	Filter for Individual user on code Clone Analysis
GMN	Graph Matching Network
GGNN	Gated Graph Neural Network
IBK	Instance Based Learner
KSA	Kappa Statistic Agreement
LDA	Linear Discriminant Analysis
LOC	Lines of Code
LSH	Locality Sensitive Hashing
LSTM	Long Short-Term Memory
LSQ	Lack of software quality of the clone
MAE	Mean Absolute Error
MLP	Multilayer Perceptron
MO	Maintenance Overhead
MOGA-NN	Multi-Objective Genetic Algorithm
NN	Neural Network
NRMSE	Normalized Root Mean Square Error
OS	Operating System
PACE	Position aware character embedding
PDG	Program Dependency Graph
RAE	Recursive autoencoders
RNN	Recurrent neural networks
SBT	Structure based traversal
sIB	Sequential information bottleneck
SLR	Systematic Literature Review
SMO	Sequential Minimal Optimization
SOM	Self Organized Mapping
SVM	Support Vector Machine
TBCAA	Tree-based Convolution over API-enhanced AST
TBCCD	Tree-Based Convolution for Clone Detection
TECCD	Tree Embedding Approach for Code Clone Detection
TF-IDF	Term Frequency-inverse document frequency

Appendix B. Quality assessment score

Study Id	Ref	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Total score	Rating
S1	[27]	Y	Y	N	N	N	N	N	Y	3	Good
S2	[35]	Y	Y	Y	N	N	Y	N	N	4	Good
S3	[30]	Y	Y	Y	N	N	Y	N	N	4	Good
S4	[69]	Y	Y	Y	Y	Y	Y	N	N	6	Excellent
S5	[75]	Y	Y	Y	N	Y	Y	N	N	5	Good
S6	[32]	Y	Y	N	N	N	Y	N	N	3	Good
S7	[34]	Y	Y	N	N	N	Y	N	Y	4	Good
S8	[61]	Y	Y	Y	Y	Y	Y	N	N	6	Excellent
S9	[76]	Y	Y	N	Y	Y	Y	N	Y	6	Excellent
S10	[70]	Y	Y	Y	Y	Y	Y	N	N	6	Excellent
S11	[33]	Y	Y	Y	N	Y	Y	N	N	5	Good
S12	[58]	Y	Y	N	Y	Y	Y	N	N	5	Good
S13	[22]	Y	Y	N	N	Y	Y	N	N	4	Good
S14	[20]	Y	Y	Y	N	Y	Y	Y	N	6	Excellent
S15	[21]	Y	Y	P	N	Y	P	N	N	4	Good
S16	[42]	Y	Y	Y	N	Y	Y	Y	Y	7	Excellent
S17	[67]	Y	Y	N	N	Y	Y	N	N	4	Good
S18	[74]	Y	Y	Y	N	Y	Y	N	N	5	Good
S19	[63]	Y	Y	N	N	Y	Y	N	Y	5	Good
S20	[43]	Y	Y	N	N	N	N	Y	N	3	Good
S21	[64]	Y	Y	Y	N	Y	Y	N	N	5	Good
S22	[44]	Y	Y	Y	N	Y	Y	Y	Y	7	Excellent
S23	[31]	Y	Y	Y	N	Y	Y	N	N	5	Good
S24	[47]	Y	Y	Y	N	Y	Y	Y	Y	7	Excellent
S25	[72]	Y	Y	Y	Y	Y	Y	N	N	6	Excellent
S26	[40]	Y	Y	Y	N	Y	Y	Y	N	6	Excellent
S27	[65]	Y	Y	Y	N	N	Y	N	N	4	Good
S28	[49]	Y	Y	N	N	N	Y	N	N	3	Good
S29	[50]	Y	Y	N	N	N	Y	N	N	3	Good
S30	[53]	Y	Y	N	N	Y	Y	N	N	4	Good
S31	[36]	Y	Y	Y	N	Y	Y	Y	N	6	Excellent
S32	[46]	Y	Y	Y	N	Y	Y	Y	Y	7	Excellent
S33	[62]	Y	Y	Y	Y	Y	Y	N	Y	7	Excellent
S34	[45]	Y	Y	Y	N	Y	Y	Y	Y	7	Excellent
S35	[71]	Y	Y	Y	N	Y	Y	N	N	5	Good
S36	[59]	Y	Y	N	Y	Y	Y	N	Y	6	Excellent
S37	[60]	Y	Y	Y	N	Y	Y	N	Y	6	Excellent
S38	[54]	Y	Y	N	N	Y	Y	N	Y	5	Good
S39	[37]	Y	Y	N	N	Y	Y	Y	Y	6	Excellent
S40	[57]	Y	Y	N	N	Y	Y	N	N	4	Good
S41	[51]	Y	Y	N	N	Y	Y	N	Y	5	Good
S42	[41]	Y	Y	N	N	Y	Y	Y	Y	6	Excellent
S43	[66]	Y	Y	Y	Y	Y	Y	N	N	6	Excellent
S44	[55]	Y	Y	N	N	Y	Y	N	N	4	Good
S45	[28]	Y	Y	Y	N	Y	Y	N	N	5	Good
S46	[52]	Y	Y	Y	N	Y	Y	N	Y	6	Excellent
S47	[23]	Y	Y	N	N	Y	Y	N	Y	5	Good
S48	[24]	Y	Y	N	N	Y	Y	N	N	4	Good
S49	[25]	Y	Y	N	N	Y	Y	N	Y	5	Good
S50	[68]	Y	Y	N	N	Y	Y	N	N	4	Good
S51	[38]	Y	Y	N	N	Y	Y	Y	N	5	Good
S52	[29]	Y	Y	N	N	Y	Y	N	N	4	Good
S53	[48]	Y	Y	N	N	Y	Y	Y	N	5	Good
S54	[26]	Y	Y	N	N	N	Y	N	N	3	Good
S55	[39]	Y	Y	N	N	Y	Y	Y	Y	6	Excellent
S56	[56]	Y	Y	N	N	Y	Y	N	N	4	Good
S57	[73]	Y	Y	Y	Y	Y	Y	N	N	6	Excellent

References

[1] D. Rattan, R. Bhatia, M. Singh, Software clone detection: A systematic review, Inf. Softw. Technol. 55 (2013) 1165–1199, http://dx.doi.org/10.1016/j.infsof.2013.01.008.

[2] C.J. Kapser, M.W. Godfrey, Cloning considered harmful considered harmful: Patterns of cloning in software, Empir. Softw. Eng. 13 (2008) 645–692, http://dx.doi.org/10.1007/s10664-008-9076-6.

[3] M. Kaur, D. Rattan, Towards an approach to recommend and rank clones for refactoring, Adv. Math. Sci. J. 9 (2020) 3783–3788, http://dx.doi.org/10.37418/amsj.9.6.56.

[4] J. Krinke, G. Antoniol, E. Merlo, R. Koschke, S. Bellon, Comparison and evaluation of clone detection tools, IEEE Trans. Softw. Eng. 33 (2007) 577–591, http://dx.doi.org/10.1109/tse.2007.70725.

[5] D. Rattan, R. Bhatia, M. Singh, Detecting high level similarities in source code and beyond, Int. J. Energy Inf. Commun. 6 (2015) 1–16, http://dx.doi.org/10.14257/ijeic.2015.6.2.01.

[6] D. Rattan, M. Singh, R. Bhatia, Design and Development of an Efficient Software Clone Detection Technique (Dissertation), Thapar Institute of Engineering and Technology, 2015.

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

[8] C.K. Roy, J.R. Cordy, A survey on software clone detection research, Queen's Sch. Comput. TR. 541 (2007) 64–68.

[9] R. Koschke, Survey of research on software clones, in: Dagstuhl Semin. Proc., 2007.

[10] B. Kitchenham, P. Brereton, Z. Li, D. Budgen, A. Burn, Repeatability of systematic literature reviews, in: 15th Annu. Conf. Eval. Assess. Softw. Eng. (EASE 2011), 2011, pp. 46–55, http://dx.doi.org/10.1049/ic.2011.0006.

[11] B. Kitchenham, Procedures for Performing Systematic Reviews, Vol. 33, Keele Univ., Keele, UK, 2004, pp. 1–26.

[12] P. Brereton, B.A. Kitchenham, D. Budgen, M. Turner, M. Khalil, Lessons from applying the systematic literature review process within the software engineering domain, J. Syst. Softw. 80 (2007) 571–583, http://dx.doi.org/10.1016/j.jss.2006.07.009.

[13] B. Kitchenham, O.P. Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkman, Systematic literature reviews in software engineering–a systematic literature review, Inf. Softw. Technol. 51 (2009) 7–15.

[14] J.R. Pate, R. Tairas, N.A. Kraft, Clone evolution: A systematic review, J. Softw. Evol. Process. 25 (2013) 261–283, http://dx.doi.org/10.1002/smr.579.

[15] C.K. Roy, M.F. Zibran, R. Koschke, The vision of software clone management : Past, present, and future, in: 2014 Softw. Evol. Week - IEEE Conf. Softw. Maintenance, Reengineering, Reverse Eng., 2014, pp. 18–33.

[16] M. Mondal, C.K. Roy, K.A. Schneider, A survey on clone refactoring and tracking, J. Syst. Softw. 159 (2020) 110429.

[17] B. Kitchenham, S. Charters, Guidelines for Performing Systematic Literature Reviews in Software Engineering, Vol. 5, Tech. Rep. EBSE-2007-01, 2007, p. 65.

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

[19] K.P. Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, 2012.

[20] M. White, M. Tufano, C. Vendome, D. Poshyvanyk, Deep learning code fragments for code clone detection, in: ASE 2016 - Proc. 31st IEEE/ACM Int. Conf. Autom. Softw. Eng., 2016, pp. 87–98, http://dx.doi.org/10.1145/2970276.2970326.

[21] S. Jadon, Code clones detection using machine learning technique: Support vector machine, in: Proceeding - IEEE Int. Conf. Comput. Commun. Autom. ICCCA 2017, 2016, pp. 303–309, http://dx.doi.org/10.1109/CCAA.2016.7813733.

[22] I. Keivanloo, F. Zhang, Y. Zou, Threshold-free code clone detection for a large-scale heterogeneous java repository, in: 2015 IEEE 22nd Int. Conf. Softw. Anal. Evol. Reengineering, 2015, pp. 201–210.

[23] J. Zeng, K. Ben, X. Li, X. Zhang, Fast code clone detection based on weighted recursive autoencoders, IEEE Access 7 (2019) 125062–125078.

[24] C. Wang, J. Gao, Y. Jiang, Z. Xing, H. Zhang, W. Yin, M. Gu, J. Sun, Go-clone: graph-embedding based clone detector for golang, in: Proc. 28th ACM SIGSOFT Int. Symp. Softw. Test. Anal., 2019, pp. 374–377.

[25] Y. Gao, Z. Wang, S. Liu, L. Yang, W. Sang, Y. Cai, TECCD: A tree embedding approach for code clone detection, in: 2019 IEEE Int. Conf. Softw. Maint. Evol., 2019, pp. 145–156.

[26] H. Xue, Y. Mei, K. Gogineni, G. Venkataramani, T. Lan, Twin-finder: Integrated reasoning engine for pointer-related code clone detection, in: 2020 IEEE 14th Int. Work. Softw. Clones, 2020, pp. 1–7.

[27] P. Barson, N. Davey, R. Frank, D. Tansley, Dynamic competitive learning applied to the clone detection problem, in: Proc. Int. Work. Appl. Neural Networks to Telecommun., University of Hertfordshire, 1995, pp. 234–241.

[28] H. Shi, R. Wang, Y. Fu, Y. Jiang, J. Dong, K. Tang, J. Sun, Vulnerable code clone detection for operating system through correlation-induced learning, IEEE Trans. Ind. Inform. 15 (2019) 6551–6559.

[29] C. Guo, H. Yang, D. Huang, J. Zhang, N. Dong, J. Xu, J. Zhu, Review sharing via deep semi-supervised code clone detection, IEEE Access 8 (2020) 24948–24965.

[30] S.K. Abd-El-Hafiz, A metrics-based data mining approach for software clone detection, in: 2012 IEEE 36th Annu. Comput. Softw. Appl. Conf., 2012, pp. 35–41.

[31] L. Li, H. Feng, W. Zhuang, N. Meng, B. Ryder, CCLearner: A deep learning-based clone detection approach, in: Proc. - 2017 IEEE Int. Conf. Softw. Maint. Evol. ICSME 2017, 2017, pp. 249–260, http://dx.doi.org/10.1109/ICSME.2017.46.

[32] P. Ma, Y. Bian, X. Su, A clustering method for pruning false positive of clonde code detection, in: Proc. 2013 Int. Conf. Mechatron. Sci. Electr. Eng. Comput., 2013, pp. 1917–1920.

[33] B. Joshi, P. Budhathoki, W.L. Woon, D. Svetinovic, Software clone detection using clustering approach, in: Int. Conf. Neural Inf. Process., 2015, pp. 520–527.

[34] S. Cesare, Y. Xiang, J. Zhang, Clonewise–detecting package-level clones using machine learning, in: Int. Conf. Secur. Priv. Commun. Syst., 2013, pp. 197–215.

[35] D.M. Shawky, A.F. Ali, An approach for assessing similarity metrics used in metric-based clone detection techniques, in: 2010 3rd Int. Conf. Comput. Sci. Inf. Technol., 2010, pp. 580–584.

[36] V. Saini, F. Farmahinifarahani, Y. Lu, P. Baldi, C.V. Lopes, Oreo: Detection of clones in the twilight zone, in: Proc. 2018 26th ACM Jt. Meet. Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng., 2018, pp. 354–365.

[37] L. Chen, W. Ye, S. Zhang, Capturing source code semantics via tree-based convolution over API-enhanced AST, in: Proc. 16th ACM Int. Conf. Comput. Front., 2019, pp. 174–182.

[38] D. Wehr, H. Fede, E. Pence, B. Zhang, G. Ferreira, J. Walczyk, J. Hughes, Learning semantic vector representations of source code via a siamese neural network, 2019, ArXiv Prepr. arXiv:1904.11968.

[39] W. Wang, G. Li, B. Ma, X. Xia, Z. Jin, Detecting code clones with graph neural network and flow-augmented abstract syntax tree, in: 2020 IEEE 27th Int. Conf. Softw. Anal. Evol. Reengineering, 2020, pp. 261–271.

[40] A. Sheneamer, S. Roy, J. Kalita, A detection framework for semantic code clones and obfuscated code, Expert Syst. Appl. 97 (2018) 405–420.

[41] H. Yu, W. Lam, L. Chen, G. Li, T. Xie, Q. Wang, Neural detection of semantic code clones via tree-based convolution, in: 2019 IEEE/ACM 27th Int. Conf. Progr. Compr., 2019, pp. 70–80.

[42] A. Sheneamer, J. Kalita, Semantic clone detection using machine learning, in: 2016 15th IEEE Int. Conf. Mach. Learn. Appl., 2016, pp. 1024–1028.

[43] J. Ghofrani, M. Mohseni, A. Bozorgmehr, A conceptual framework for clone detection using machine learning, in: 2017 IEEE 4th Int. Conf. Knowledge-Based Eng. Innov., 2017, pp. 810–817.

[44] A. Sheneamer, H. Hazazi, S. Roy, J. Kalita, Schemes for labeling semantic code clones using machine learning, in: 2017 16th IEEE Int. Conf. Mach. Learn. Appl., 2017, pp. 981–985.

[45] A. Sheneamer, CCDLC detection framework-combining clustering with deep learning classification for semantic clones, in: 2018 17th IEEE Int. Conf. Mach. Learn. Appl., 2018, pp. 701–706.

[46] G. Zhao, J. Huang, DeepSim: Deep learning code functional similarity, in: ESEC/FSE 2018 - Proc. 2018 26th ACM Jt. Meet. Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng., 2018, pp. 141–151, http://dx.doi.org/10.1145/3236024.3236068.

[47] H. Wei, M. Li, Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code, in: IJCAI, 2017, pp. 3034–3040.

[48] Y. Yuan, W. Kong, G. Hou, Y. Hu, M. Watanabe, A. Fukuda, From local to global semantic clone detection, in: 2019 6th Int. Conf. Dependable Syst. their Appl., 2020, pp. 13–24.

[49] H. Xue, G. Venkataramani, T. Lan, Clone-slicer: Detecting domain specific binary code clones through program slicing, in: Proc. 2018 Work. Form. an Ecosyst. Around Softw. Transform., 2018, pp. 27–33.

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

[51] D. Perez, S. Chiba, Cross-language clone detection by learning over abstract syntax trees, in: 2019 IEEE/ACM 16th Int. Conf. Min. Softw. Repos., 2019, pp. 518–528.

[52] K.W. Nafi, T.S. Kar, B. Roy, C.K. Roy, K.A. Schneider, CLCDSA: Cross language code clone detection using syntactical features and API documentation, in: 2019 34th IEEE/ACM Int. Conf. Autom. Softw. Eng., 2019, pp. 1026–1037.

[53] M. Tufano, C. Watson, G. Bavota, M. Di Penta, M. White, D. Poshyvanyk, Deep learning similarities from different representations of source code, in: Proc. - Int. Conf. Softw. Eng., 2018, pp. 542–553, http://dx.doi.org/10.1145/3196398.3196431.

[54] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, X. Liu, A novel neural source code representation based on abstract syntax tree, in: 2019 IEEE/ACM 41st Int. Conf. Softw. Eng., 2019, pp. 783–794.

[55] L. Büch, A. Andrzejak, Learning-based recursive aggregation of abstract syntax trees for code clone detection, in: 2019 IEEE 26th Int. Conf. Softw. Anal. Evol. Reengineering, 2019, pp. 95–104.
[56] P. Keller, L. Plein, T.F. Bissyandé, J. Klein, Y. Le Traon, What you see is what it means! semantic representation learning of code based on visualization and transfer learning, 2020, ArXiv Prepr. arXiv:2002.02650.
[57] G. Mostaeen, J. Svajlenko, B. Roy, C.K. Roy, K.A. Schneider, CloneCognition: machine learning based code clone validation tool, in: Proc. 2019 27th ACM Jt. Meet. Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng., 2019, pp. 1105–1109.
[58] J. Yang, K. Hotta, Y. Higo, Classification model for code clones based on machine learning, Empir. Softw. Eng. 20 (2015) 1095–1125, http://dx.doi.org/10.1007/s10664-014-9316-x.
[59] G. Mostaeen, J. Svajlenko, B. Roy, C.K. Roy, K.A. Schneider, On the use of machine learning techniques towards the design of cloud based automatic code clone validation tools, in: 2018 IEEE 18th Int. Work. Conf. Source Code Anal. Manip., 2018, pp. 155–164.
[60] S. Rongrong, Z. Liping, Z. Fengrong, A method for identifying and recommending reconstructed clones, in: Proc. 2019 3rd Int. Conf. Manag. Eng. Softw. Eng. Serv. Sci., 2019, pp. 39–44.
[61] W. Wang, M.W. Godfrey, Recommending clones for refactoring using design, context, and history, in: Proc. - 30th Int. Conf. Softw. Maint. Evol. ICSME 2014, 2014, pp. 331–340, http://dx.doi.org/10.1109/ICSME.2014.55.
[62] R. Yue, Z. Gao, N. Meng, Y. Xiong, X. Wang, J.D. Morgenthaler, Automatic clone recommendation for refactoring based on the present and the past, in: Proc. - 2018 IEEE Int. Conf. Softw. Maint. Evol. ICSME 2018, 2018, pp. 115–126, http://dx.doi.org/10.1109/ICSME.2018.00021.
[63] J. Pati, B. Kumar, D. Manjhi, K.K. Shukla, A comparison among ARIMA, BP-NN, and MOGA-NN for software clone evolution prediction, IEEE Access 5 (2017) 11841–11851.
[64] F. Zhang, S.-C. Khoo, X. Su, Machine-learning aided analysis of clone evolution, Chin. J. Electron. 26 (2017) 1132–1138.
[65] F. Zhang, X. Su, W. Zhao, T. Wang, An empirical study of code clone clustering based on clone evolution, J. Harbin Inst. Technol. New Ser. 24 (2017).
[66] V. Arammongkolvichai, R. Koschke, C. Ragkhitwetsagul, M. Choetkiertikul, T. Sunetnanta, Improving clone detection precision using machine learning techniques, in: 2019 10th Int. Work. Empir. Softw. Eng. Pract., 2019, pp. 31–315.
[67] J. Svajlenko, C.K. Roy, A machine learning based approach for evaluating clone detection tools for a generalized and accurate precision, Int. J. Softw. Eng. Knowl. Eng. 26 (2016) 1399–1429.
[68] V. Saini, F. Farmahinifarahani, Y. Lu, D. Yang, P. Martins, H. Sajnani, P. Baldi, C.V. Lopes, Towards automating precision studies of clone detectors, in: 2019 IEEE/ACM 41st Int. Conf. Softw. Eng., 2019, pp. 49–59.
[69] X. Wang, Y. Dang, L. Zhang, D. Zhang, E. Lan, H. Mei, Can I clone this piece of code here? in: Proc. 27th IEEE/ACM Int. Conf. Autom. Softw. Eng., 2012, pp. 170–179, http://dx.doi.org/10.1145/2351676.2351701.
[70] X. Wang, Y. Dang, L. Zhang, D. Zhang, E. Lan, H. Mei, Predicting consistency-maintenance requirement of code clones at copy-and-paste time, IEEE Trans. Softw. Eng. 40 (2014) 773–794, http://dx.doi.org/10.1109/TSE.2014.2323972.
[71] S. Yan, L. Zhang, D. Liu, An empirical study on optimization of training dataset in harmfulness prediction of code clone using ensemble feature selection model, in: 2018 5th Int. Conf. Inf. Commun. Technol. Disaster Manag., 2018, pp. 1–4.
[72] F. Zhang, S. Khoo, X. Su, Predicting change consistency in a clone group, J. Syst. Softw. 134 (2017) 105–119.
[73] F. Zhang, S.C. Khoo, X. Su, Improving maintenance-consistency prediction during code clone creation, IEEE Access 8 (2020) 82085–82099, http://dx.doi.org/10.1109/ACCESS.2020.2990645.
[74] D. Liu, D. Liu, L. Zhang, M. Hou, C. Wang, The prediction of code clone quality based on bayesian network, Int. J. Softw. Eng. Appl. 10 (2016) 47–56, http://dx.doi.org/10.14257/ijseia.2016.10.4.05.
[75] D. Steidl, N. Göde, Feature-based detection of bugs in clones, in: 2013 7th Int. Work. Softw. Clones, IWSC 2013 - Proc., 2013, pp. 76–82, http://dx.doi.org/10.1109/IWSC.2013.6613047.
[76] A. Imazato, K. Hotta, Y. Higo, S. Kusumoto, Predicting risky clones based on machine learning, in: Int. Conf. Prod. Softw. Process Improv., 2014, pp. 294–297.
[77] J.R. Quinlan, C4. 5: Programs for Machine Learning, Elsevier, 2014.
[78] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, S. Khudanpur, Recurrent neural network based language model, in: Elev. Annu. Conf. Int. Speech Commun. Assoc., 2010.
[79] P. Wang, J. Svajlenko, Y. Wu, Y. Xu, C.K. Roy, CCAligner: a token based large-gap clone detector, in: Proc. 40th Int. Conf. Softw. Eng., 2018, pp. 1066–1077.
[80] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: Proc. 22nd Acm Sigkdd Int. Conf. Knowl. Discov. Data Min., 2016, pp. 785–794.
[81] L. Breiman, Random forests, Mach. Learn. 45 (2001) 5–32.
[82] J.J. Rodriguez, L.I. Kuncheva, C.J. Alonso, Rotation forest: A new classifier ensemble method, IEEE Trans. Pattern Anal. Mach. Intell. 28 (2006) 1619–1630.
[83] C.K. Roy, J.R. Cordy, NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization, in: 2008 16th IEEE Int. Conf. Progr. Compr., 2008, pp. 172–181.
[84] H. Sajnani, V. Saini, J. Svajlenko, C.K. Roy, C.V. Lopes, Sourcerercc: Scaling code clone detection to big-code, in: Proc. 38th Int. Conf. Softw. Eng., 2016, pp. 1157–1168.
[85] T. Kamiya, S. Kusumoto, K. Inoue, CCFinder: A multilinguistic token-based code clone detection system for large scale source code, IEEE Trans. Softw. Eng. 28 (2002) 654–670.
[86] L. Jiang, G. Misherghi, Z. Su, S. Glondu, Deckard: Scalable and accurate tree-based detection of code clones, in: 29th Int. Conf. Softw. Eng., 2007, pp. 96–105.
[87] N. Göde, R. Koschke, Incremental clone detection, in: 2009 13th Eur. Conf. Softw. Maint. Reengineering, 2009, pp. 219–228.
[88] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, C.-J. Lin, LIBLINEAR: A library for large linear classification, J. Mach. Learn. Res. 9 (2008) 1871–1874.
[89] D.W. Aha, D. Kibler, M.K. Albert, Instance-based learning algorithms, Mach. Learn. 6 (1991) 37–66.
[90] J. Friedman, T. Hastie, R. Tibshirani, et al., Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors), Ann. Statist. 28 (2000) 337–407.
[91] L. Breiman, Bagging predictors, Mach. Learn. 24 (1996) 123–140.
[92] T.G. Dietterich, Ensemble methods in machine learning, in: Int. Work. Mult. Classif. Syst., 2000, pp. 1–15.
[93] J. Svajlenko, J.F. Islam, I. Keivanloo, C.K. Roy, M.M. Mia, Towards a big data curated benchmark of inter-project code clones, in: 2014 IEEE Int. Conf. Softw. Maint. Evol., 2014, pp. 476–480.
[94] E. Duala-Ekoko, M.P. Robillard, Tracking code clones in evolving software, in: Proc. - Int. Conf. Softw. Eng., 2007, pp. 158–167, http://dx.doi.org/10.1109/ICSE.2007.90.
[95] F.A. Fontana, M. Zanoni, F. Zanoni, A duplicated code refactoring advisor, in: Int. Conf. Agil. Softw. Dev., 2015, pp. 3–14, http://dx.doi.org/10.1007/978-3-319-18612-2_1.
[96] D. Mazinanian, N. Tsantalis, R. Stein, Z. Valenta, JDeodorant: Clone refactoring, in: 38th Int. Conf. Softw. Eng. Companion, 2016, pp. 613–616, http://dx.doi.org/10.1145/2889160.2889168.
[97] S. Bazrafshan, R. Koschke, Effect of clone information on the performance of developers fixing cloned bugs, in: Proc. - 2014 14th IEEE Int. Work. Conf. Source Code Anal. Manip. SCAM 2014, 2014, pp. 1–10, http://dx.doi.org/10.1109/SCAM.2014.10.
[98] N. Tsantalis, D. Mazinanian, G.P. Krishnan, Assessing the refactorability of software clones, IEEE Trans. Softw. Eng. 41 (2015) 1055–1090, http://dx.doi.org/10.1109/TSE.2015.2448531.
[99] J.R. Quinlan, Induction of decision trees, Mach. Learn. 1 (1986) 81–106.
[100] Y. Freund, R.E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, J. Comput. System Sci. 55 (1997) 119–139.
[101] J. Platt, Sequential minimal optimization: A fast algorithm for training support vector machines, 1998.
[102] N. Friedman, D. Geiger, M. Goldszmidt, Bayesian network classifiers, Mach. Learn. 29 (1997) 131–163.
[103] O. Ivanciuc, Weka machine learning for predicting the phospholipidosis inducing potential, Curr. Top. Med. Chem. 8 (2008) 1691–1709, http://dx.doi.org/10.2174/156802608786786589.
[104] TensorFlow: An open-source software library for machine intelligence, 2021, https://www.tensorflow.org (accessed February 27, 2021).
[105] T. Mikolov, S. Kombrink, A. Deoras, L. Burget, J. Cernocky, Rnnlm-recurrent neural network language modeling toolkit, in: Proc. 2011 ASRU Work., 2011, pp. 196–201.
[106] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[107] Introduction to deep neural networks, 2021, https://deeplearning4j.org/ (accessed February 27, 2021).
[108] C.-C. Chang, C.-J. Lin, LIBSVM: a library for support vector machines, ACM Trans. Intell. Syst. Technol. 2 (2011) 1–27.
[109] V. Subramanian, Deep Learning with PyTorch: A Practical Approach to Building Neural Network Models using PyTorch, Packt Publishing Ltd, 2018.
[110] N. Ketkar, Introduction to pytorch, in: Deep Learn. with Python, Springer, 2017, pp. 195–208.
[111] P. Geurts, D. Ernst, L. Wehenkel, Extremely randomized trees, Mach. Learn. 63 (2006) 3–42.

[112] IJaDataset 2.0, 2021, https://sites.google.com/site/asegsecold/projects/seclone (accessed February 27, 2021).
[113] L. Mou, G. Li, L. Zhang, T. Wang, Z. Jin, Convolutional neural networks over tree structures for programming language processing, in: Proc. AAAI Conf. Artif. Intell., 2016.
[114] E. Juergens, F. Deissenboeck, B. Hummel, Clonedetective-a workbench for clone detection research, in: 2009 IEEE 31st Int. Conf. Softw. Eng., 2009, pp. 603–606.
[115] J.R. Cordy, C.K. Roy, The NiCad clone detector, in: 2011 IEEE 19th Int. Conf. Progr. Compr., 2011, pp. 219–220.
[116] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, L. Bier, Clone detection using abstract syntax trees, in: Proceedings. Int. Conf. Softw. Maint. (Cat. No. 98CB36272), 1998, pp. 368–377.
[117] J. Svajlenko, C.K. Roy, CloneWorks: a fast and flexible large-scale near-miss clone detection tool, in: ICSE (Companion), 2017, pp. 177–179.
[118] M.S. Uddin, C.K. Roy, K.A. Schneider, Simcad: An extensible and faster clone detection tool for large scale software systems, in: 2013 21st Int. Conf. Progr. Compr., 2013, pp. 236–238.
[119] Simian, Code clone detection tool, 2021, http://www.redhillconsulting.com.au/products/simian/ (accessed February 27, 2021).
[120] W. Toomey, Ctcompare: Code clone detection using hashed token sequences, in: 2012 6th Int. Work. Softw. Clones, 2012, pp. 92–93.
[121] T.K. Ho, The random subspace method for constructing decision forests, IEEE Trans. Pattern Anal. Mach. Intell. 20 (1998) 832–844.