International Journal Software Engineering and Computer Science (IJSECS)
5 (2), 2025, 599-611
Published Online August 2025 in IJSECS (http://www.journal.lembagakita.org/index.php/ijsecs)
P-ISSN: 2776-4869, E-ISSN: 2776-3242. DOI: https://doi.org/10.35870/ijsecs.v5i2.4431.

RESEARCH ARTICLE **Open Access**

Validating and Detecting User-Specific Code Clones: An AI Framework Leveraging Metric-Based Feature Vectors

Asfa Praveen *
Assistant Professor, Department of Computer Science, College of Engineering and Computer Science, Mustaqbal University, Buraidah, Al Qassim, Saudi Arabia.
Corresponding Email: asfa.prn@gmail.com.

Received: June 4, 2025; Accepted: June 30, 2025; Published: August 1, 2025.

Abstract: Like other verification aspects, code clone validation remains highly subjective and user-dependent. This research presents an AI-based approach utilizing fragment-specific metric-based feature vectors to identify and validate customized code clones. We derive classification feature vectors through appropriate code metrics, training various machine learning models for identifier classification. The resulting framework enables users to submit code clone pairs for cloud-based validation. Upon submission, the trained AI model analyzes pairs using their metric features, generating user-specific validation scores returned via a RESTful API. We describe the framework architecture encompassing metric extraction, model training, and cloud deployment. Experimental results demonstrate the framework's ability to adapt effectively to individual validation strategies, optimizing accuracy while significantly reducing inspection effort compared to non-customized clone detection systems. A prototype system demonstrates the feasibility of providing automatically computed AI-based validation scores integrated with existing validation tools.

Keywords: Feature Vectors; RESTful API; Artificial Intelligence; User-Specific Code; Clone Pairs.

1. Introduction

When building software, it is standard procedure to recycle segments of code that have been used in the past by copying and pasting the relevant portions or reusing them in some other way [1]. When this approach is applied, what are known as code clones are produced. Anytime sections of code are repeated, whether it be a copy and paste or altering minor details, code clones are created [2]. For the purpose of defining clones more precisely, the scientific community has accepted an arbitrary categorization consisting of four types:

1) Type-1 clones are those pieces of code that are exactly the same as another piece of code, possibly with some stylistic elements like comments and whitespaces or whitespace deviations.
2) Type-2 is the same as Type-1 with the only difference that in Type-2, the names and types of identifiers have been altered.
3) Type-3 rewrites the code that modifies the program by altering it at the statement level. These clones have their own unique characteristics.
4) Type-4 clones are created by scattering sections of code that execute identically but have different syntactic structures.

As shown in other studies, between 7-23% of the total source code of a given software application is redundantly copied and pasted [3]. It is feasible to intentionally make copies of code blocks for the sake of accelerating development, albeit with negative consequences. These so-called "clones," in turn, significantly contribute to software maintenance challenges which increase software sustaining costs [4]. Tt may be difficult to update in a manner that is uniform across the board when a large amount of code has been flexibly relocated within the system due to the high level of system complexity [5]. Whenever duplicated code is altered, these duplicated code segments undergo modification in a manner different from the original. Doing so causes new software problems in most cases. The developers' duplication of software logic in different sections of the framework through copy-pasting troublesome code contributes greatly to the software's underlying issues, which is a major contributor to software bugs. Discovering patterns of duplicate code using these techniques will significantly reduce the time required for maintaining software systems. Moreover, one can improve the design of the software system by leveraging commonalities among the discovered code clones, thereby enhancing the overall software system design [6][7].

Researchers have developed and suggested at least seventy different tools and methods for automating the process of clone detection as a result of the significant amount of attention that has been paid to this particular subject over the course of the past ten years. As noted earlier, "The computer program will strive to build an exhaustive catalog of all conceivable classes and pairs of code clones" 0. Owing to vast avenues through which these clones may evolve over time, it is practically impossible that a string-matching approach would succeed in detecting Type-2, Type-3, and Type-4 clone pairs. Code clone Type-1 does not change in any way. For instance, disguising the names of variables and procedure names can be done; small bits of code can be added or removed; and myriad syntactical alterations can be made to existing code clones. These changes can also be made to implement most features' implementations, turning them into something else entirely [9]. Due to all of these changing features evolving over time, the searching problem has become an order of magnitude harder to solve. To manage such intricate structures of the source code and detect all possible code clone pairings, the original source codes have to undergo extensive modifications. Modifications such as pretty-printing the code, normalizing the identifiers, as well as constructing syntax trees from the fragments of code, among other techniques, fall into this category [10].

2. Related Work

The repetition of clone detection processes often results in the identification of false positive clones. This happens due to the fact that the problem is usually difficult to solve, and some form of generalization or normalization is needed before the source code can be examined. In this case, the user arrives at the decision that the two code snippets are not the same, are not true clones for whatever reason, or are identical as far as these computer programs are concerned. Research indicates that the rule for identifying Type-3 and Type-4 clones is especially prone to subjectivity, and depends on the person being interviewed, or the software system being interrogated in relation to the cloning operation. This is a very relevant insight as it concerns Type-2 clones' criteria [11].

For instance, in a study carried out in [12], identical clone sets were given out to a variety of users so that those users could assess the accuracy of clone identification algorithms. According to the results of the research, there were noticeable differences in the ways in which various individuals verified the same clones (the number of decided genuine positive code clones for the same given clone sets ranged between 4.76% and 23.81%). Because of this, programmers are typically required to manually evaluate the results of a clone detection to determine whether or not they are a true clone before applying those results to certain scenarios. This is important because of the uncertainty whether the end product will be a true clone or not.

Every moderately complex software system inevitably encounters substantial delays every time there is a need for a human validation step. When attempting to detect code clones, the overwhelming number of proposed pairs makes it difficult for the programmers to uncover the genuine positive clones that they desire. This is due to the fact that the noise can obscure genuine positive clones. Some studies estimate that JDK 1.4.2 is over eight percent bloated with duplicate lines of code [13].

The line "Code clones are responsible for 15% of the total lines of code" implies that the remaining 85% may also contain some level of redundancy. The broad estimates that clone detection technologies provide point towards the time that will be absorbed manually verifying the data before it can be reliably worked with. In addition, the algorithms for clone identification that are employed by the tools often function in a broad way. This implies that they are loose from the specifics of a given system or the biases of a given user.

Hence, as long as the tool is able to produce true positive clones, the majority of clones concerning the projects that the engineers and programmers are focused on [14]. Code clone detection systems are not as helpful as they might be since it is a time-consuming effort to mine those code clones for significance from the report given by the software. The code clone detection systems lack logic which impacts their efficiency. The addition of dimensions to the problem drives up its complexity while simultaneously expanding its scope. Past investigations have demonstrated how subjective the technique is; therefore, any such copies which are discovered must undergo thorough scrutiny.

In regard to the user survey prepared for [15], there were 105 unique pairs of clones found together with a group of clerks who were tasked with validating them. The validation criteria for the same set of 105 newly identified clone pairs was markedly different across four uncoordinated manual workstations. For example, one user credited only five of the so-called random pairings without a discernible sequential logic. Others, however, cited twenty-four, twenty-three, and twenty-five random pairings respectively.

The research for software engineering problems toward solution strategies that incorporate computational intelligence. Early studies centered on software reliability, which included the analysis of program aging mechanisms that resulted in service outages [21], the formulation of methods for code clone analysis and identification as well as removal [22], and monolithic legacy systems in three-tiered analytic frameworks [16]. This focus on software quality later transformed into predictive maintenance, which seeks to understand "aging" by employing extreme learning machines accelerated by evolutionary computing [19]. More recent studies indicate a substantial shift toward transdisciplinary multi-ML integration. This includes applying ML to create a system structure [17], analyzing signals in [18], and employing techniques of ML and image processing for the automatic assessment of products [20]. Starting from solving problems of the internal software subsystem, the path illustrates a movement towards applying advanced computational algorithms like ML, evolutionary computing, and image processing to solve existing intricate problems in the healthcare diagnostic and safety management systems.

3. Proposed Model

In this research study, we investigated an issue that was rather close to user driven code clone classification. A user-based attribution for duplication is performed via a scope of copying determination, which is built upon a token sequence similarity analysis utilizing TF-IDF metric of term frequency-inverse document frequency (TF-IDF)-based vectors. For these texts, we will identify and compare the words that appear in the texts being compared most frequently. Users manually sift through the pairs of code clones detected by automatic systems and mark them as true or false positive clones as shown in figure 1 so as to train the system.

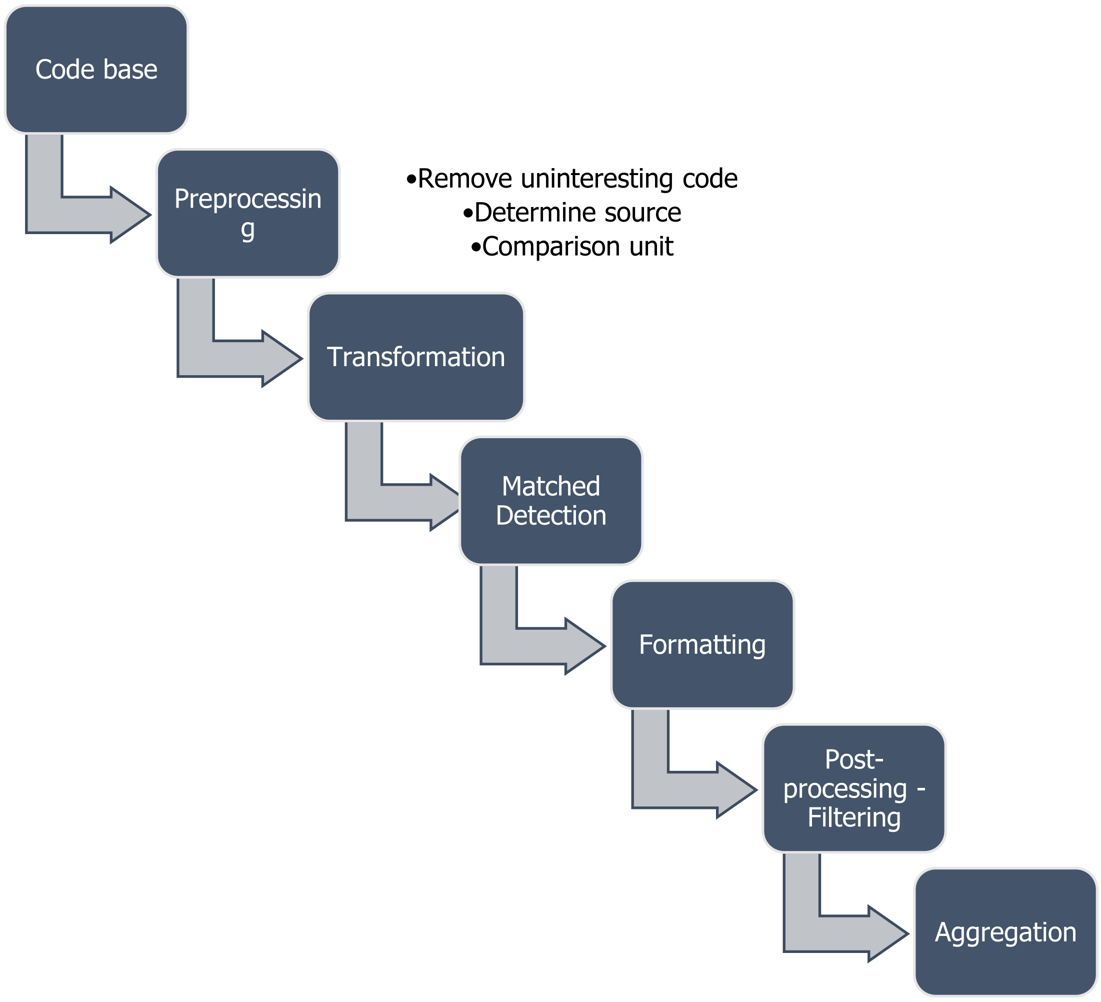

Figure 1. Proposed Model

When the clones are identified by numbers that fall within a specified range, the variable w(m), which is placed in the interval [0, 1], is used to signify the weight that has been allocated to each individual clone. This weight might be negative or positive. On the other hand, if the clones contain a Boolean marking, the result returned by w(m) will always be 1 regardless of the circumstances. In addition to this, the proposed system is able to repeatedly gather user feedback over the course of time to refresh the training sets MT and MF to enhance the model. This is done by analyzing the data generated by the model. The research also reveals that the validation accuracy greatly decreases as the target clone size increases, which is demonstrated by the research work. This is because the user-specific validation learning utilized by FICA is wholly dependent on token sequence. The clustering is the ultimate goal of the process of discovering new techniques, which will lead to the discovery of new ways. To determine which clone combinations, have the greatest likelihood of being successful, the next step that we are going to do is to compute the proper metrics that correspond to each of these strategies. As a technique for locating these duplicated actions, a collection of twelve distinct count metrics has been presented.

$$p(u) = dN^{-1} + (1-d) \sum_{v \in adj[u]} \frac{\cos\_sim(u,v)}{\sum_{z \in adj[u]} \cos\_sim(z,v)} p(v) \tag{1}$$

Where
N - codes,
d - damping factor,
adj[*x*] - nodes adjacent to a node *x*, and
p(*x*) - node centrality *x*.

Using TF-IDF to calculate Cosine Similarity for 'a' and 'b' within the document collection D, we get the value of CosSimD(a,b). After this, c, the score that measures the probability for an untagged clone set to belong to Mt or Mf, can be computed, as stated in the method and Equation 4. This score is based on the probability that an unmarked clone set will have the same genotype as a marked clone set. Count metrics are metrics whose values may be determined by counting the number of instances of a certain thing. Put differently, count metrics are those metrics that track the frequency of an event. The diversity of approaches to cloning is immense, each with its own prescribed count metric meld tailored to that approach's unique requirements.

$$\cos\_sim = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2 \sum_{i=1}^{n} (B_i)^2}} \tag{2}$$

Where
A_i - vector *A* component, and
B_i - vector *B* component.

3.1 Computation of Code Clone Metrics

Figure 2 presents various code clone metrics that are essential for analyzing and comparing code segments. These metrics provide quantitative measures to assess the similarity and complexity of code clones across different methods and functions.

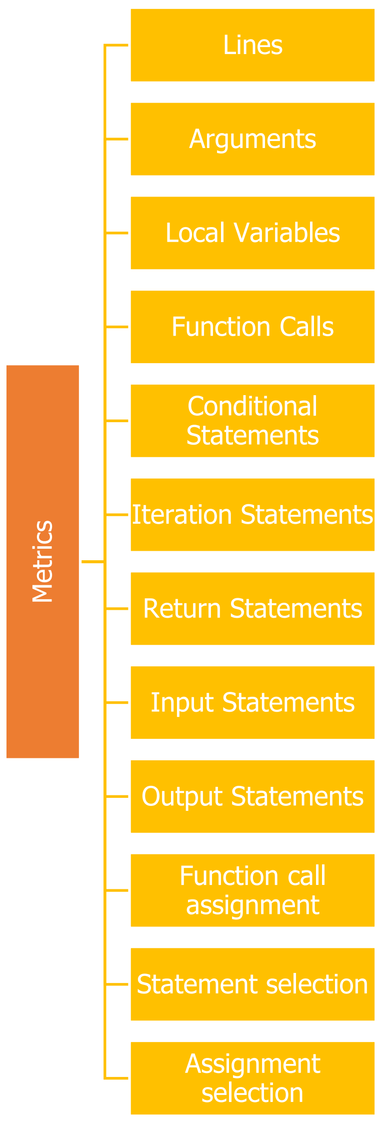

Figure 2. Code Clone Metrics

The following metrics are utilized in the computation of code clone characteristics:

1) No. of Lines: How many lines of actual code are in each method by counting the number of lines that appear between the symbols marking the beginning and conclusion of the function declaration, respectively.
2) No. of Arguments: This will give you the total number of lines of real code in each method. The number of arguments that a method is willing to take, as well as the data types of those arguments and their relative places on the call stack, are all things that are taken into consideration.
3) No. of Local Variables: This value, which is known as the Local Variable Count, is a representation of the total local variables that are defined within the function. This is the Count of Local Variables. At no time is the scope of the set variable which was used by the function Its global variables and activity level of the function variables taken into consideration.
4) No. of Function Calls: It is obtaining a rough estimation about the actual number of times the method under consideration calls other functions with the help of this information. It is a standard method for studying the logic that is included inside the source code of a program, and it provides a perspective from above of the methods that are defined in addition to those that are called.
5) Conditional Statements: The total number of conditional statements that are a part of the process. This includes the number of if, else if, and else statements, in addition to any other conditional statements that may be present. It is necessary since the semantics of the method as a whole are dependent on this one factor alone.
6) Iteration Statements: This variable provides the total amount of iteration statements that are contained within a method specification. When calculating this statistic, the number of times that statements define the prepositions while, do, and for is taken into consideration. These are also fairly significant in setting the standard way in which the method is carried out, therefore it crucial to pay attention to them.

7) Return Statements: The return statements that are associated with a method are displayed in this area. The count of all the method exits that are described in the specification is included in the value of the variable.
8) No. of Input Statements: In order to get the variable values, the user choices, and other information, the total number of input statements that were carried out during the procedure is tracked in a record that is maintained. These are of great service in determining the degree to which two distinct methods are comparable to one another and are a useful tool for making that determination.
9) No. of Output Statements: The same may be described for the number of output statements; in the same way that the number of input statements is useful for dissecting the fundamental components of the approach, so too can the number of output statements. It is of lesser concern to capture correct data for the provided buffers, console and other locations than it is to describe using basic output statements that achieve the purpose of message formatting for output and informative messages.
10) Function Call Assignments: This metric preserves a record of the number of variables that receive their values as a direct consequence of assigning the return value of a function to one of those variables, and it does so by counting the number of variables that receive their values. Their application is regarded as valuable due to the fact that they provide a separate classification of the variables and the values that those variables reflect.
11) Statement Selection: This metric is used to count the conditional operators, cases, and various other sorts of selection statements that are contained in each unique method. In addition to this, it tallies the overall number of selection statements. The branches will be generated as a result of the combination of this collection of statements and the conditional statements. After then, each of these branches will be evaluated, which will result in the procedure execution pattern becoming clear.
12) Assignment Statements: This indicator displays a count of the statements that were used to change the values of the method variables. These statements were used to assign new values to the variables. The phrase Number of Assignment Statements is used to refer to this particular metric. Statements may be constructed using elements such as basic assignments, mathematical expressions, unary operators, and a variety of other building blocks.

In addition to these twelve quantitative evaluations, there are also four more quantitative criteria to take into consideration. In the process of creating these metrics, features such as function calls, file reads, I/O activities, and the utilization of both reference and value parameters within function calls and global variables are taken into consideration. Check the code fragment starting with S. An account of each of the additional metrics considered is provided in the subsequent sections. The account is in the form of texts and there exists a plethora pertaining to the incident. Most importantly, focus on the fact that these measures have been computed using the provided, two functions in c statements.

$$C(\mathbf{S}) = \mathit{Fan\ Out}(\boldsymbol{S}) \tag{3}$$

Where
FAN OUT(***S*** *)* - individual function calls within S.

It is feasible to calculate it using the formula Complexity(S) = Globals(S) / (Fan Out(S) + 1), where Globals(S) represents the total number of global variable declarations that were used or updated in S. This can be done by dividing the result by 1. If the declaration of a variable is not located within the same code block in which the variable is being used, then that variable is referred to as being global. Mccabe(S) is determined by adding one to the total number of control decision statements that are present in S. The results of computing each of the 16 metrics that are connected with each approach are then saved in a database so that they may be later examined and extracted. It is a requirement that all of the metric values for a cloned method pair be the same for type 1, 2, and 4. This is a necessary condition. For the detection of -1, -2, and -4 clones, the only database records that are taken into consideration are those that provide metric values for method pairings that are matches.

3.2 Clone Detection

It is necessary to conduct a textual comparison on the code that has been formatted and normalized in order to ascertain whether or not the extracted pairs from the shortened set of methods are accurate. This is done so that the accuracy of the extracted pairs can be measured once they have been computed. Type-1 clones are those that, according to the definition, do nothing more than copy and paste the source code precisely as it is written. The methods that have a score that is an exact match with another method are referred to as type-1 copied techniques. This indicates that the entire number of lines in the method has exactly the same number of lines as the total number of lines that are comparable to one another. Clone pairs are pairs of processes that match up with one another in a textual comparison because they have the same computed metric values and are therefore deemed to be clones. Clone pairs are also known as clone pairs. It provides a breakdown and summary of the many identifying criteria that can be applied to clones.

Code segments that are structurally comparable to one another are indicative of type-2 cloned methods. These methods are distinguished by the fact that there are only minute differences in the identifiers, literals, types, white space, layout, and comments. This suggests that the textual comparison will be carried out on the program code that was produced by the template tool. A pair of procedures is assumed to be a clone if both it and its sibling have the same computed values for comparison. The comparison that the template provides considers, besides the type-1 procedures copied, also type-2 procedures. For that reason, it is important to eliminate each one of them individually. Also, it is possible to note the differences in the parameters when the altered code is compared to the original code.

The "type-3 clones" as they are termed, have some modifications made wherein statements can be added, deleted, or modified. Both the beginning and ending procedures from the created metrics boundaries can be assessed, however for this instance, only range values are considered. These modifications to the metrics incorporate accounting for variation. Because of this, two distinct ranges of metric values have been established which aid in the detection of type 3 clones. This is true because of the merges of the rules clones metric values ranges. Multiple methods have been defined in order to account for each different merge of ranges. These types of extracts are possible for metric ranges obtained from various definitional approaches. Range1 is defined as the range of operations which can be considered to be duplicates which are bound above by the mean and below by the actual evaluated metric range's value.

$$Range_1 = (Ac*100)/\ Av \tag{4}$$

Ac - Actual metric value for the present method
Av - Average metric value for the present method
It is only possible to consider clones of a method to be type-3 versions of that method if those clones have a value for Range1 that is higher than 90%. After that, in order to calculate Range2, we divide the total number of lines that are contained within a method by the number of lines that are comparable to the method that we are looking for. This allows us to determine the range of the method.

$$Range_2 = (N*100)/M \tag{5}$$

N – Total number of similar lines in the code
M – Total line of codes in a module

When determining whether or not a method pairing is qualified for a Type-3 clone declaration, it is necessary to determine whether or not both of the corresponding Range2 values in the template methods are more than 85%. The research has not yet determined a particular range for type-3 clones that can be regarded as conclusive. The difference between Range1 and Range2 is the same size for all three categories, which means it is equal to zero (1, 2, and 4). This threshold number for type-3 ranges was determined to be the ideal level as a result of extensive testing with values ranging from 85 to 100%. The production of clone programs of type 4 occurs when major chunks of the code share a great deal of semantic resemblance with one another. When it comes to these clones, the component that has been cloned is not always an exact duplicate of the one that was made initially. This is because cloning is not always perfect. Since they are both employed to carry out the same kind of reasoning, there is a potential that two different developers will write two pieces of code that are functionally identical to one another. Because of this, the meaning of the cloned pieces has been preserved, despite the fact that their syntactical and structural representations have been altered. For type 4, we first start with each of the ways that were discussed earlier and look at the differences and similarities using the results that were achieved through using the measurement system. In the case where the metric values that are produced out of the two different ways are subject to a comparison then they are compared to the methods which serve as templates. The comparison proves to be successful if both versions contain the same code as each other. Members of this group are unlike the rest of the population in every sense.

3.3 Post-processing

The prior step has provided us with clone pairs which are now at our disposal. Through the Clone Manager program, we receive both clone pairs and clone clusters as output. Clone Manager may be found here. As soon as all of the various cloning processes, which are also referred to as possible clone pairs, have been identified, the procedures are categorized into several distinct groups, with each group being assigned its own unique number. Clustering the clone pairs in which the members of the same cluster share many characteristics, whilst the members of other clusters share characteristics that are highly different from one another. There is one text file for each of the four distinct categories of clones, in which individual clones as well as pairings and groups of clones are documented. Under normal conditions, analysts expect that it will be between 0 and 1, although most likely it will be closer to 0. On the other hand, it is unfavorable because in Equations 6-8, the sum of squared errors (SSE) is more than the entire sum of squares. This is because there are not many data points that accurately represent this scenario; as a result, the underlying model or models learnt less about it and more about others. The reason for this is because there are not many data points that accurately describe this scenario.

$$R^2 = 1 - \frac{SSE}{TSS} \tag{6}$$

$$SSE = \sum_{i=1}^{N} \left(P_{m(i)} - P_{p(i)} \right)^2 \tag{7}$$

$$TSS = \sum_{i=1}^{N} \left(P_{m(i)} - P_{p} \right)^2 \tag{8}$$

Where,
$P_m(i)$ - i^{th} path loss,
$P_p(i)$ - predicted path loss,
P_p - average path loss, and
N - number of observations.

4. Result and Discussion

4.1 Results

When doing cloning analysis, the application of automated clone validation can be of assistance, and this is true for software systems of varied sizes. As a consequence of this, we aimed to put the system through its paces in a number of different scenarios, putting it through its paces with a variety of clone detection methods, user populations, and machine configurations in order to see how well it performed. We settled on Python version 2.7 as the language of choice for the server side after determining that this was the best way to get the job done. The Python programming language was used to develop the web application that was created with Flask, which is a microframework. On an individual basis, code clones that have been recognized by one of the various ways for detecting code clones can be retained on the system server and validated there. There are several methods for identifying code clones.

The duplicates of the code are shown to the users of the system several times so that they can perform an inspection by hand to look for any errors that may have been introduced. After recording each user's verdict for a given code clone pair, the server then does a comparison between the user verdict (either true positive or false negative) for that pair and the user's profile. This comparison determines whether the user's verdict was accurate or not. Because we intended to simplify scalability and distribute computing for large amounts of data, we made the decision to adopt CouchDB, which is a NoSQL database system. We anticipate that CouchDB will continue to be useful to us in the years to come.

The code clone pairs that are validated have already been shown in Tables 1-4. They form a part of the set of data that is available, and which can now be used to design a classification model as per the methodology described earlier. Currently, code clone pairs are undergoing preprocessing which will enable the extraction of features needed for constructing representation models. After data preparation, features of the clone using Java as a programming language were developed post-preprocessing of the data.

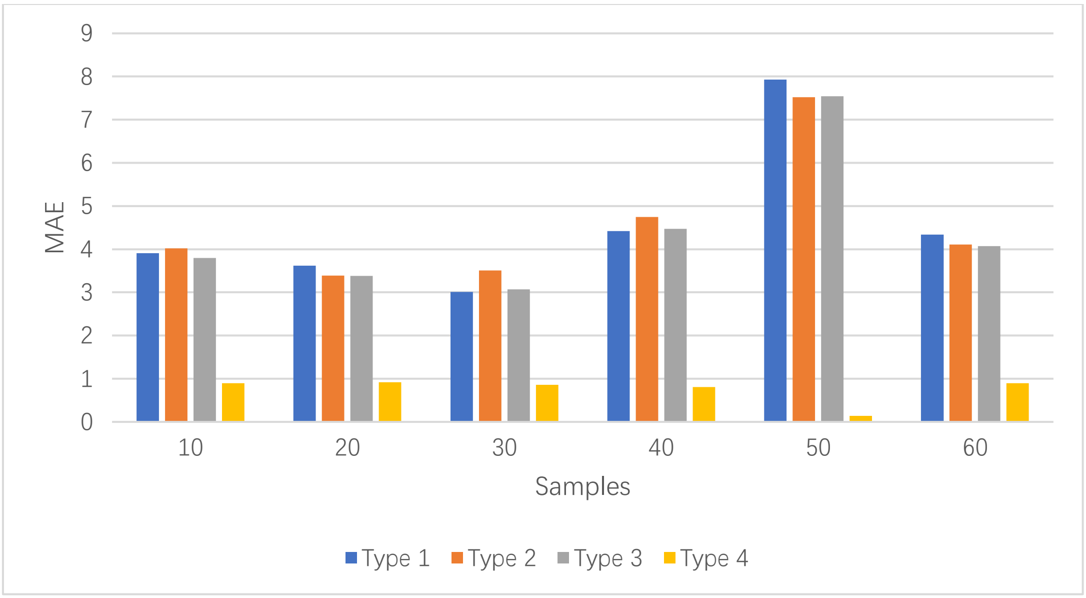

Figure 3. MAE of various Clone types

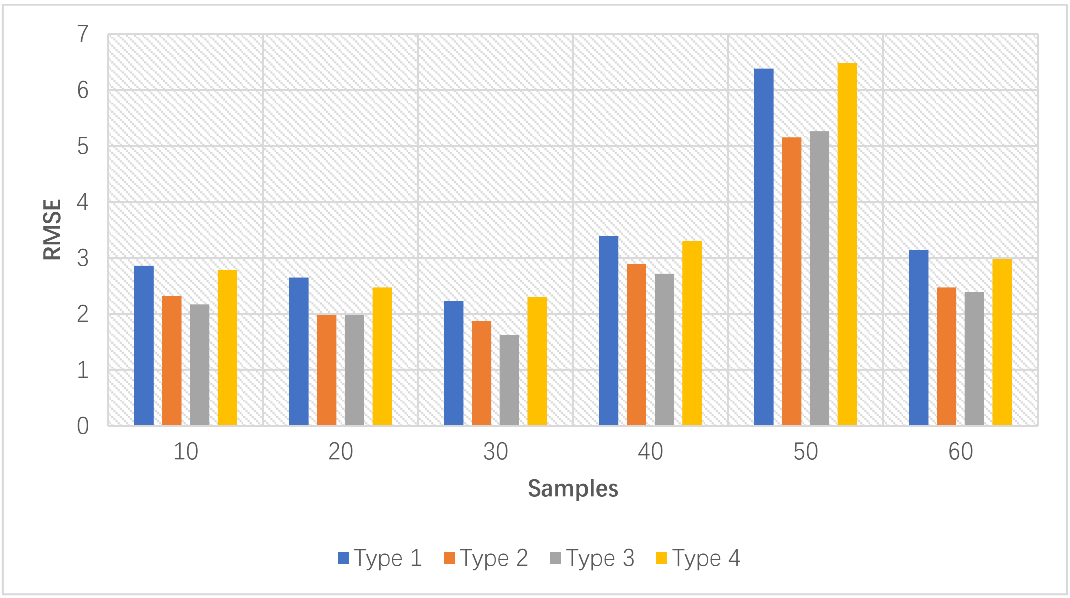

Figure 4. RMSE of various Clone types

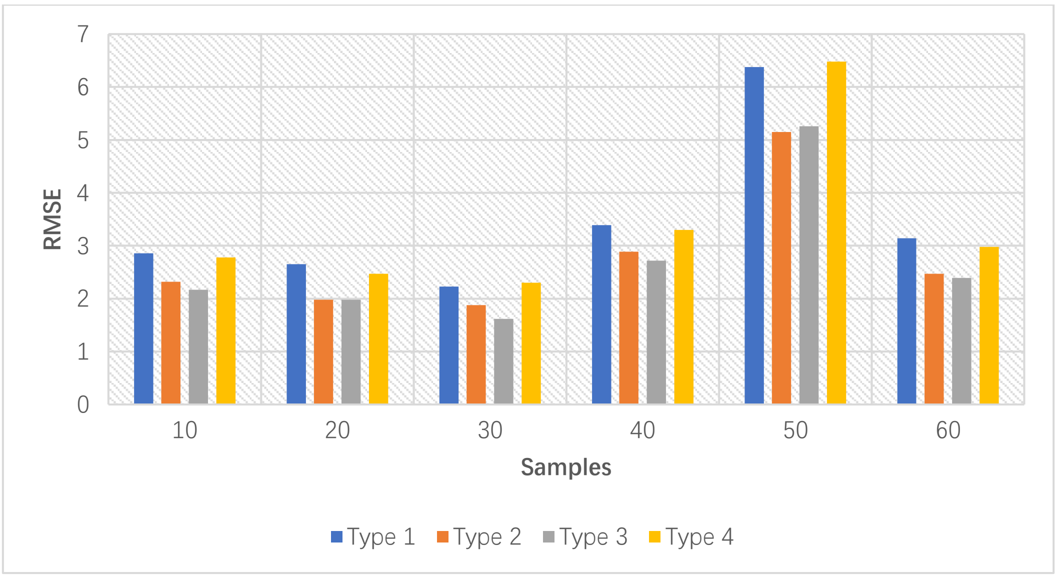

Figure 5. RMSE of various Clone types

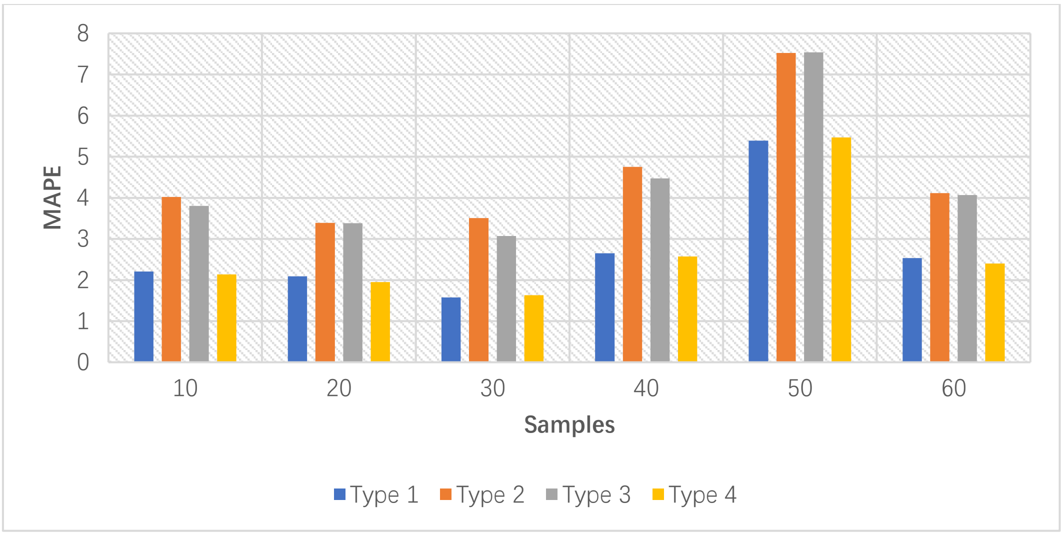

Figure 6. MAPE of various Clone types

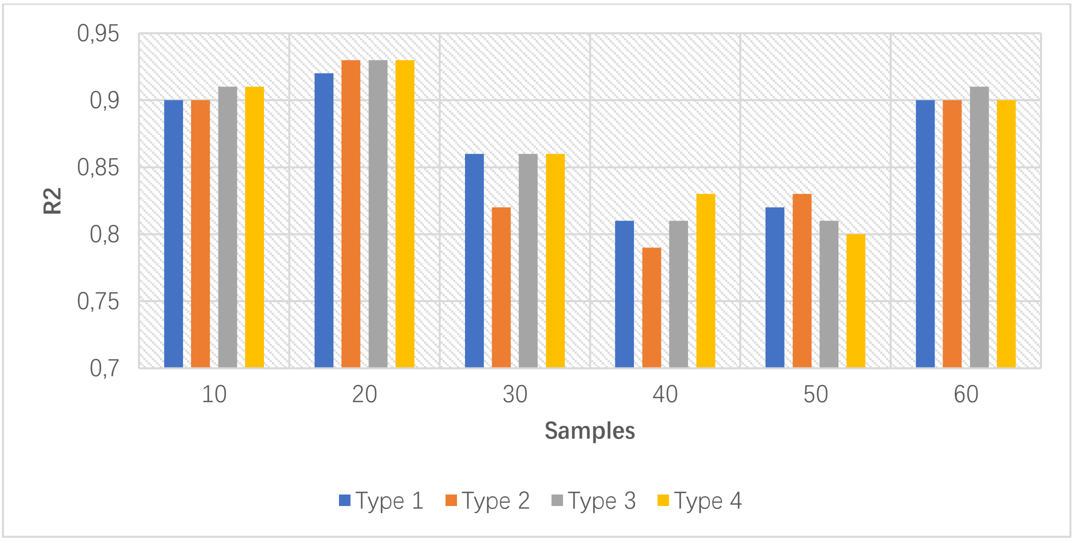

Figure 7. R2 of various Clone types

Table 1. MAE

Samples	Clone Pair	Clone cluster	Proposed AI
10	3.75	4.28	3.92
20	3.37	3.65	3.31
30	3.06	3.61	3.34
40	4.23	5.21	4.60
50	7.88	10.62	9.41
60	4.09	4.38	4.33

Table 2. RMSE

Samples	Clone Pair	Clone cluster	Proposed AI
10	2.90	3.10	2.90
20	2.49	2.66	2.44
30	2.46	2.54	2.46
40	3.54	3.98	3.55
50	6.38	8.79	7.82
60	3.04	3.19	3.16

Table 3. MAPE

Samples	Clone Pair	Clone cluster	Proposed AI
10	2.23	2.38	2.24
20	1.96	2.09	1.92
30	1.74	1.80	1.75
40	2.76	3.10	2.78
50	5.43	7.38	6.58
60	0.50	2.56	2.53

Table 4. R2

Samples	Clone Pair	Clone cluster	Proposed AI
10	0.90	0.89	0.90
20	0.93	0.92	0.93
30	0.84	0.82	0.84
40	0.80	0.75	0.80
50	0.13	0.43	0.11
60	0.90	0.89	0.90

4.2 Discussion

Requests for the validation of code clone pairs are sent to the server, which then makes use of the trained machine learning model for clone categorization prior to providing the validation score to the user in JSON format. On the server side, communication is accomplished through the use of RESTful API requests and answers. After the validation score has been computed, it can be fed into the appropriate code clone detection tools in order to facilitate the classification or comprehension of the clones, depending on the parameters of the tool. There are many different approaches to this. In practice, it can be done either before or after the validation score has been calculated according to your preference. Perhaps the modularly trained model might be able to utilize some local clone detection tools for local prediction. This would be preferable than relying largely on cloud deployment to make predictions. Because of this, there would be no longer be a requirement to rely entirely on cloud deployment.

5. Conclusion

In this paper, the feature vectors that were derived from the training dataset were incorporated into the training phase of the classification model. We tried out a few different ways of applying machine learning in order to achieve our goal of categorizing clones. Within the scope of this study, not only do we present in-depth findings, but we also conduct an in-depth analysis of how effectively the classifiers worked. When the training of the model is finished, it may be used to validate newly created code clone pairs for particular users. After the model has been trained, this is something that may occur. Users have the ability to submit code clone pairs for validation, and these pairs may then be uploaded to the cloud for storage and processing. After it attains that level, the learned model will be able to generate a prediction and subsequently return the validation score to the user endpoint associated with the specific forecast, delivering the response while en route to the user. After preparing the test code clone pairs, we also implemented a prototype aimed at the transfer of the validation score from the model to the user. This score was intended to prepare the user for the validation score. To meet the aim of the prototype, it was requisite to do this.

References

[1] Kodhai, E., & Kanmani, S. (2014). Method-level code clone detection through LWH (Light Weight Hybrid) approach. *Journal of Software Engineering Research and Development*, *2*(1), 1-29. https://doi.org/10.1186/s40411-014-0012-8

[2] Jia, Y., Binkley, D., Harman, M., Krinke, J., & Matsushita, M. (2009, March). KClone: A proposed approach to fast precise code clone detection. In *Third International Workshop on Detection of Software Clones (IWSC)* (Vol. 1).

[3] Walker, A., Cerny, T., & Song, E. (2020). Open-source tools and benchmarks for code-clone detection: Past, present, and future trends. *ACM SIGAPP Applied Computing Review*, *19*(4), 28-39. https://doi.org/10.1145/3381307.3381310

[4] Hu, Y., Zhang, Y., Li, J., & Gu, D. (2017, May). Binary code clone detection across architectures and compiling configurations. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)* (pp. 88-98). IEEE. https://doi.org/10.1109/ICPC.2017.22.

[5] Keivanloo, I., Zhang, F., & Zou, Y. (2015, March). Threshold-free code clone detection for a large-scale heterogeneous java repository. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* (pp. 201-210). IEEE. https://doi.org/10.1109/SANER.2015.7081830

[6] Kamiya, T. (2015, March). An execution-semantic and content-and-context-based code-clone detection and analysis. In *2015 IEEE 9th International Workshop on Software Clones (IWSC)* (pp. 1-7). IEEE. https://doi.org/10.1109/IWSC.2015.7069882.

[7] Murakami, H., Hotta, K., Higo, Y., Igaki, H., & Kusumoto, S. (2012, September). Folding repeated instructions for improving token-based code clone detection. In *2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation* (pp. 64-73). IEEE. https://doi.org/10.1109/SCAM.2012.21

[8] Xue, H., Venkataramani, G., & Lan, T. (2018, June). Clone-hunter: Accelerated bound checks elimination via binary code clone detection. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages* (pp. 11-19). https://doi.org/10.1145/3211346.3211347

[9] Tekchandani, R., Bhatia, R., & Singh, M. (2018). Semantic code clone detection for Internet of Things applications using reaching definition and liveness analysis. *The Journal of Supercomputing*, *74*(9), 4199-4226. https://doi.org/10.1007/s11227-016-1832-6

[10] Avetisyan, A., Kurmangaleev, S., Sargsyan, S., Arutunian, M., & Belevantsev, A. (2015, September). LLVM-based code clone detection framework. In *2015 Computer Science and Information Technologies (CSIT)* (pp. 100-104). IEEE. https://doi.org/10.1109/CSITechnol.2015.7358259

[11] Shobha, G., Rana, A., Kansal, V., & Tanwar, S. (2021). Code clone detection—a systematic review. *Emerging Technologies in Data Mining and Information Security: Proceedings of IEMIS 2020, Volume 2*, 645-655. https://doi.org/10.1007/978-981-33-4367-2_61.

[12] Roy, C. K., & Cordy, J. R. (2009, April). A mutation/injection-based automatic framework for evaluating code clone detection tools. In *2009 International Conference on Software Testing, Verification, and Validation Workshops* (pp. 157-166). IEEE. https://doi.org/10.1109/ICSTW.2009.18

[13] Ragkhitwetsagul, C., Krinke, J., & Marnette, B. (2018, March). A picture is worth a thousand words: Code clone detection based on image similarity. In *2018 IEEE 12th International Workshop on Software Clones (IWSC)* (pp. 44-50). IEEE. https://doi.org/10.1109/IWSC.2018.8327318

[14] Zhang, F., Khoo, S. C., & Su, X. (2020). Improving maintenance-consistency prediction during code clone creation. *IEEE Access*, *8*, 82085-82099. https://doi.org/10.1109/ACCESS.2020.2990645.

[15] Sargsyan, S., Kurmnagaleev, S., Belevantsev, A., Aslanyan, H., & Baloian, A. (2018). Scalable code clone detection tool based on semantic analysis. *Proceedings of the Institute for System Programming of the RAS*, *27*(1), 39-50. https://doi.org/10.15514/ISPRAS-2015-27(1)-3

[16] Praveen, A., Qamar, S., & Ahamad, S. (2015). Three levels analytical model for monolithic legacy program source code analysis. *Journal of Information Engineering and Applications*, *2224-5782*.

[17] Ahamad, S. (2022). System architecture for brain-computer interface based on machine learning and internet of things. *International Journal of Advanced Computer Science and Applications*, *13*(3).

[18] Gupta, D. N., Anand, R., Ahamad, S., Patil, T., Dhabliya, D., & Gupta, A. (2023, April). Phonocardiographic signal analysis for the detection of cardiovascular diseases. In *International Conference on Frontiers of Intelligent Computing: Theory and Applications* (pp. 529-538). Springer Nature Singapore. https://doi.org/10.1007/978-981-99-6706-3_47

[19] Ahamad, S. (2022). Evolutionary computing driven extreme learning machine for objected oriented software aging prediction. *International Journal of Computer Science & Network Security*, *22*(2), 232-240.

[20] Dhamodaran, S., Ahamad, S., Ramesh, J. V., Muthugurunathan, G., Manikandan, K., Pramanik, S., & Pandey, D. (2023). Food quality assessment using image processing technique. In *Handbook of Research on Thrust Technologies' Effect on Image Processing* (pp. 295-309). IGI Global. https://doi.org/10.4018/978-1-6684-8618-4.ch018

[21] Ahamad, S. (2016). Program aging and service crash. *International Journal of Computer Applications Technology and Research*, *5*(7).

[22] Bari, M. A., & Ahamad, D. S. (2011). Code cloning: The analysis, detection and removal. *International Journal of Computer Applications*, *20*(7), 34-38.