I. Approach

Our detailed approach is constructed as follows:

A. Method Declaration

STCD is based on methods, it counts tokens in each method. So the very first thing to do is method declaration. Here we used AST Parser Tool to catch each method and get the information, which means, the *startLineNumber, endLineNumber, methodName, methodParameters, methodType, methodBody*. This costs extra time than needed and can be improved in future work.

B. Tokenization

Since we have divided the whole file into methods, next is to tokenize the method body and get the frequency of each token. This includes the comment and white space removal process.

C. Categorization

Now for each method, we have a list of tokens, these tokens fall into 9 categories: *methodParameter, methodType, tokenListNumber, tokenListType, tokenListKeyword, tokenListMarker, tokenListOperator, tokenListOther1, tokenListOther2*. The purpose of doing so is that these categories have different meanings. For example, two pieces of code both have 2 type names: "int", that provides no useful information, however, if two pieces of code both have 2 method parameter names: "drawVerticalLine", that means these pair are more likely the same. Later on these categories will be given different weights, depending on their importance.

A list of categorized tokens is shown in table I:

Token	Category	Freq	Token	Category	Freq
int	Type	1	)	Marker	6
for	Keyword	1	}	Marker	2
i	Other1	8	{	Marker	1
System	Other1	3	+	Operator	6
out	Other1	3	=	Operator	1
println	Other1	3	-	Operator	1
toBinary	Other1	1	¡	Operator	1
Integer	Other1	1	:	Other2	2
toBinaryString	Other1	1	5.0	Num	1
(	Marker	6	33.0	Num	1

TABLE I
A List of Token Frequency

D. Pretreatment of Variable Names

Among all the categories, variable name, i.e. *methodParameter* is special. Its similarity is caculated with bigram algorithm. If "drawVerticalLine" and "VerticalDrawLine" appear in two code fragments respectively, we don't want to take them as different. So two similar variable names will be compared by bigram similarity, if they are similar, they will be taken as the same and their frequencies will be compared later.

E. Similarity Calculation for Each Category

After the pretreatment of variable names, a similarity algorithm is applied to each category. A list of tokens with their frequencies can be taken as a vector, so the similarity for two vectors can be caculated, and we have 9 similarity numbers for each comparision.

F. Final Similarity Calculation and Result

Finally, the 9 similarity numbers are given different weights and the final similarity will be calculated. A threshold is introduced, if the final similarity is higher than the threshold, the result will be printed out in the following format: clone group number, similarity number: method name 1, start line number, end line number; method name 2, start line number, end line number.

G. Multilayer Perceptron

During the final step, different weights are given to different categories, however, picking 9 weights arbitrarily may not be reasonable. So we used machine learning method to improve the results. Provided training files with "ground truth", a multilayer perceptron(MLP) is introduced to achieve this goal.

MLP is an artificial neural network model that maps sets of input data onto a set of appropriate outputs. A single MLP contains three layers of nodes, and the nodes in one layer are fully connected with those in the next one. To train the network, this model utilizes back-propagation which is a supervised learning technique.

After actual testing, it turns out MLP does provide a better result.

A diagram describing the whole process is shown in Fig:1.

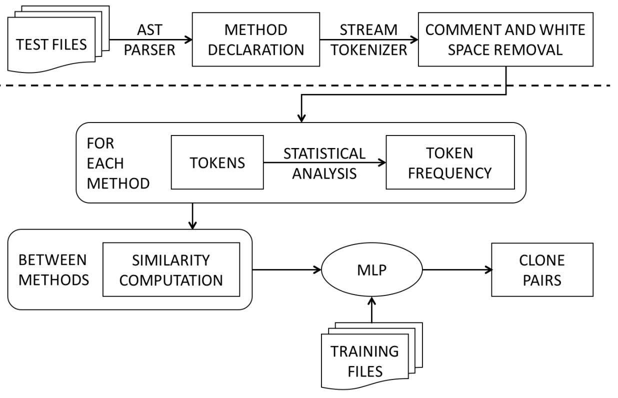

Fig. 1. Overall Project Framework