Semantic Similarity Search for Source Code Plagiarism Detection: An Exploratory Study

Fahad Ebrahim
Department of Computer Science
The University of Warwick
Coventry, UK
Fahad.Ebrahim@warwick.ac.uk

Mike Joy
Department of Computer Science
The University of Warwick
Coventry, UK
M.S.Joy@warwick.ac.uk

ABSTRACT

Source code plagiarism detection (SCPD) is a crucial challenge in computer science education that affects academic integrity. It can be considered as an Information Retrieval (IR) task. One of the IR approaches is the Semantic Similarity Search (S-3), which aims to retrieve related results, given a query. It can be applied to SCPD by obtaining the most similar pairs given a large collection of codes.

The paper presents an exploratory study that examines the utilisation of S-3 in the context of the SCPD task. So, given the source code reuse dataset (SOCO) written in Java/C++, the task is to retrieve the most similar (potentially plagiarised) pairs of codes. Technically, S-3 is based on vector search. So, embedding vectors generated by the major Code Pre-Trained Models (CodePTMs) were used as features of the conducted experiments. The accuracy of the S-3 approach exceeded the other SOCO-IR baselines in most of the CodePTMs without any training in terms of F1 score. The CodePTMs that incorporated multiple representations produced robust embeddings.

For improved accuracy metrics, several experiments were conducted to train the embedding models in both supervised and unsupervised manners. The results concluded that overall performance could improve slightly after supervised training due to the limited training set of the SOCO dataset. Unsupervised training tests had a negative impact on accuracy. The advantage of the S-3 is that it is lightweight and fast with the ability to produce excellent performance.

CCS CONCEPTS

• **Computing methodologies** → *Natural language processing*; *Unsupervised learning*; *Supervised learning by classification*; • **Information systems** → **Similarity measures**; **Language models**; • **Software and its engineering**;

KEYWORDS

Source Code Plagiarism Detection, Information Retrieval, Code Pre-Trained Models, Software Engineering, Computer Science Education, Code Similarity.

ITiCSE 2024, July 8–10, 2024, Milan, Italy
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0600-4/24/07
https://doi.org/10.1145/3649217.3653622

ACM Reference Format:
Fahad Ebrahim and Mike Joy. 2024. Semantic Similarity Search for Source Code Plagiarism Detection: An Exploratory Study. In *Proceedings of the 2024 Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024), July 8–10, 2024, Milan, Italy.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3649217.3653622

1 INTRODUCTION

Plagiarism is one of the critical issues in computer science education. Plagiarism in programming assignments can be denoted as Source Code Plagiarism [8] where a student uses someone's code without permission or knowledge. Detecting plagiarism achieves academic integrity and reduces the ethical and social consequences of committing plagiarism. Automation of the process would be time-efficient for instructors.

Source Code Plagiarism Detection (SCPD) can be viewed as a classification task where pairs of codes are classified whether they are plagiarised or not. SCPD can also be considered as a clustering task, where similar files can be grouped into clusters. Therefore, Machine Learning (ML) and Deep Learning (DL), whether in a supervised or unsupervised manner, can be utilised in the field of plagiarism detection. As the goal of SCPD is to identify the most similar files, it can be treated as an Information Retrieval (IR) task.

Natural Language Processing (NLP) is a subfield of Artificial Intelligence (AI) that seeks to make the machine understand natural written text as humans do [6]. To understand text, machines need to convert it into numbers or vectors, referred to as embeddings. Word embeddings are commonly used in NLP as a pre-processing step for different tasks to grab the context of a particular word. Vector embeddings are one of the representations of the source code [5].

The emergence of Pre-Trained Models (PTMs) greatly impacted the area of NLP and created a research field of their own. PTMs aim to train the models with large training data, which results in better generalisation. Then, it can be fine-tuned for smaller downstream tasks. There are several PTMs reported in the literature ([26],[35]). One of the common PTMs is Bidirectional Encoder Representations from Transformers (BERT) [9]. BERT aims to understand the contextual meaning of words. These can be tuned in the case of having new small training data, as it was previously trained on a large corpus of data. This is an advantage in SCPD, as finding large public data is challenging due to the legal and ethical aspects.

Then, domain-specific pre-trained models for Programming Languages (PLs) or Software Engineering (SE) have been developed for different tasks [27] as part of AI for Software Engineering (AI4SE).

AI4SE tasks can be divided into code understanding and code generation. There are various tasks related to code understanding, such as code summarisation, code completion, and code similarity detection. One of the first AI4SE models was CodeBERT [10], which is an extension of BERT for source code tasks. CodeBERT was mainly used for natural language code search and code documentation. These code models can be called CodePTMs. Robust embeddings can be extracted from these models and can serve as reliable features.

IR aims to obtain relevant results given a user request. One stage related to IR is search. Search can involve lexicals and/or semantics. Lexical search is limited to words, while semantic search considers meaning and context. Semantic search can be symmetric or asymmetric depending on the length of the text and the query. If they have a similar length, they would be symmetric, otherwise asymmetric. The task of SCPD can be represented as a symmetric similarity search where the most similar pairs of codes need to be retrieved. The IR process also involves similarity estimation, matching, and ranking.

This work is an exploratory study that inspects the effectiveness of Semantic Similarity Search (S-3) in the task of plagiarism detection. The paper explores the embeddings generated by code pre-trained models and uses them as vectors for the semantic similarity search to extract the most similar pairs of codes. The work also explores the impact of supervised and unsupervised training of the embeddings on accuracy metrics. The approach taken is simple, lightweight, and yet produces competitive accuracy metrics. All experiments are conducted on the public Source Code Reuse (SOCO) dataset [12] containing two programming languages (PLs), namely Java and C/C++.

The paper is organised as follows: section 2 covers the related work, section 3 presents the experimental study, section 4 discusses the results, and the paper ends with the conclusion, limitations, and future work.

2 RELATED WORK

2.1 SCPD surveys

There are several existing surveys on source code plagiarism detection in the literature, such as a 2019 systematic review [2] of plagiarism in programming assignments in an academic context, which covered methods such as assignment design, grading methods, available tools, and several other strategies. Furthermore, ethical aspects related to these types of plagiarism were discussed. A subsequent material survey [3] reviewed 32 articles in the field of plagiarism. They reviewed obfuscation methods, SCPD approaches, and explored available tools. The approaches were categorised on the basis of metrics, texts, tokens, graphs, and dynamics along with each similarity measurement. Another review paper [22] presented an overview of similarity detection techniques and categorised the techniques based on attribute counting, structure, and hybrid methods, and surveyed the available approaches based on these criteria. The study of [4] aimed to answer multiple questions related to SCPD. They explored techniques, PLs, and tools. A further survey [30] reviewed the definitions, obfuscation techniques, tools, algorithms, evaluations, and datasets of source code similarity detection through a systematic review. The most recent study [45] conducted a systematic review of the literature on source code similarity and considered SCPD and code cloning as applications. Tools, approaches, benchmarks, and datasets were discussed in depth.

2.2 SCPD Information Retrieval

Source code plagiarism detection approaches can be divided into different categories of approach. String-based approaches (or token- or text-based approaches) depend on treating the source code as plain text. Strings and text are synonyms, and they can be divided into multiple tokens. These approaches can be divided into N-grams, string-matching, and attribute-counting algorithms. An N-gram is a way of repeating n sequences in a text, and comparing these sequences can assist in plagiarism detection. String-matching algorithms aim to identify certain similar patterns that can indicate plagiarism. Attribute counting captures certain features of the source code by capturing the frequency of occurrences of various parts of the source code, such as identifiers and keywords.

A method related to attribute counting is Information Retrieval (IR). IR involves extracting a relevant piece of information from a set of documents. The authors in [21] explored different IR methods used in source code plagiarism. They initially created a dataset consisting of 467 files from 7 introductory Java tasks. They compared three methods: Vector Space Model (VSM), Latent Semantic Indexing (LSI), and Language Model (LM) on their dataset. The baseline was Running-Karp-Rabin Greedy-String-Tiling (RKRGST) [41]. They concluded that VSM was the technique that performed best compared to their baseline.

2.3 SOCO IR

Several works have used the SOCO dataset. Most of the papers focused on treating code reuse as classification, but as this paper is related to IR, only the works related to IR will be discussed and, therefore, compared to our work.

One of the IR approaches in SOCO was [13], which created a combination of a Java parser, an Abstract Syntax Tree (AST), and bag-of-words to be the input to a Language Model (LM) for retrieval on the Java set of the dataset. This work was named DCU.

Another work [14] used a method of two stages: IR and classification. Bag-of-words, LM, and AST were used to index candidates and then a classifier was applied to get the best pair of similar codes. They named the models LM, LM_AST, and FLM_AST based on the index type. This work experimented only with the Java subset of the dataset.

2.4 SCPD Software

Several softwares have been developed to assist with source code plagiarism detection. JPlag [34] is open-source software available for public use, and currently supports 11 PLs. Some of the supported languages are Java, C++, C, Python, and R. The comparison algorithm used is Greedy String Tiling [41], which basically compares substrings of codes with other strings. Another software is MOSS (Measure of Software Similarity). MOSS supports more than 24 PLs. MOSS employs a document fingerprinting algorithm known as Winnowing [39] where the document fingerprint is generated using the concept of hashing. The most recently developed software is Dolos [25]. Dolos utilises a hybrid approach containing tree-sitter parsers, a string-matching algorithm and indexing These parsers convert the code into an AST that captures the structure of the code. The fingerprint is generated using the concepts of hashing, k-grams and the Winnowing algorithm

2.5 Code Pre-Trained Models

There are different ways to represent source codes, as surveyed in [38]. One way to represent codes is through tokenisation and encoding to vectors. These vectors can be sparse, containing a large number of zeros, or dense, containing a compressed representation of the code. The pre-trained models have the ability to generate contextual dense vectors.

General PTMs can be divided into various categories. One category involves static models such as Word2vec [7], Glove [33] and FastText [20]. The other category involves contextual models such as BERT [9], RoBERTa [24], BART [23], T5 [36], and GPT-4 [32]. The capabilities of the latter category are higher with more robustness and semantic representations. In two articles [35],[26], lists of PTMs have been explored.

Then, domain-specific models for source code have been derived from the general NLP models. CodeBERT [10], GraphCodeBERT [17], and UniXcoder [16] follow the same architecture as BERT, PLBART [1] was derived from BART, CodeBERTa [42] was derived from RoBERTa, and CodeT5 [40] was derived from T5. These models were trained on huge source code corpora and have been used for various downstream tasks such as code summarisation, documentation, and bug detection. This work would focus on these six models. The following papers discuss the source code PTM [27–29, 43, 44, 46] and various tasks related to these models.

A comparison between the models selected for this work can be seen in Table 1 based on the number of programming languages (#PL), the input, whether it is PL or Natural Language (NL) or another representation, the number of parameters (in millions), the training dataset(s) and the architecture (Encoder or Encoder/Decoder). All these models have a maximum length of 512, 12 layers, 768-dimensional hidden states, and 12 attention heads, except CodeBERTa, which has 6 layers. All models except PLBART were trained mainly on the CodeSearchNET dataset, which is a large corpus of six programming languages [19] (Python, Java, Go, JavaScript, PHP, and Ruby). GraphCodeBERT considers the semantic structure of the code with a data flow graph. UniXcoder takes comments and AST into the representation of the source code and has two variations based on the number of PLs whether they are 6 or 9. PLBART considers the data flow graph along with the style in the overall representation of the code.

3 EXPERIMENTAL STUDY

3.1 Semantic Similarity Search

The Semantic Similarity Search approach in this work follows three stages. Firstly, the source codes were converted into embeddings extracted from the models. Secondly, the similarity between the embeddings was measured. Thirdly, the scores, along with the pairs, were ranked and obtained as seen in Figure 1. The embeddings were extracted with the sentence transformers [37] via mean pooling.

Measurement of semantic similarity was estimated using cosine similarity. It is one of the most widely used metric to estimate

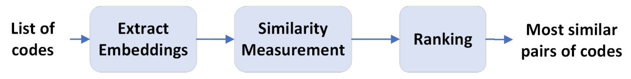

Figure 1: S-3 stages

similarity in NLP. It is the angle (θ) calculated by multiplying two vectors A and B and dividing them by the product of their norms according to Equation 1. Higher values mean more similarity between two vectors.

$$Cosine\ Similarity = cos(\theta) = \frac{A.B}{||A||.||B||} \tag{1}$$

Another similarity measurement that will be used later in the paper in supervised training is the Euclidean distance. Euclidean distance is the square root of the summation of the squared difference of two points in an M dimensional space according to Equation 2. A lower value of the distance means a shorter distance and greater similarity between a pair of files.

$$d = \sqrt{\sum_{i=1}^{M}(x_{2i} - x_{1i})^2} \tag{2}$$

For ranking and retrieval, the highest similarity scores were selected among all unique combinations of files for evaluation.

3.2 Dataset

There is a dearth of public source code plagiarism datasets due to ethical issues in data collection. Therefore, this work uses the public SOCO dataset that contains two sets: training and testing, two programming languages (Java and C++/C), and six scenarios per PL (A1, A2, B1, B2, C1, and C2). Details of the dataset can be seen in Table 2. The experiments did not include scenarios C1 and C2 in Java and C/C++, respectively, as there were no reuse cases. The ratio of instances where reuse occurs to instances where it does not is minimal, which makes the task challenging. Another challenge is the limited training data, as the number of examples is 259 and 79 in Java and C/C++, respectively. We denote the maximum number of similar codes in the same group by K. The value of K in the Java test set was equal to 2 in all scenarios. The value of K in the set of tests in C/C++ was 4 for A1 and A2 and 3 for the rest. We denote the number of reuse cases per scenario by N.

3.3 Evaluation

The evaluation of the results was based on the classification accuracy metrics. The metrics used were Precision (P), Recall (R), and the F1 score in equations 3, 4, and 5. The authors of the dataset [12] provided an evaluation script that was used to obtain the metrics values. Scores were ranked according to the F1 score.

$$Precision = P = \frac{TP}{TP + FP} \tag{3}$$

$$Recall = R = \frac{TP}{TP + FN} \tag{4}$$

$$f1score = 2 * \frac{P * R}{P + R} \tag{5}$$

Model	#PL	Input	#parameters	Training Dataset	Architecture
CodeBERT	6	NL+PL	125M	CodeSearchNET	Encoder (BERT)
GraphCodeBERT	6	NL+PL+Data flow	125M	CodeSearchNET	Encoder (BERT)
UniXcoder	6/9	NL+PL+Comment+AST	125M	CodeSearchNET + C4 + Stackoverflow	Encoder/Decoder
PLBART	7	NL+PL +Data flow+Style	140M	Github + Stackoverflow	Encoder (BART)
CodeT5	6	NL+PL	220M	CodeSearchNET + BigQuery	Encoder/Decoder (T5)
CodeBERTa	6	NL+PL	84M	CodeSearchNET	Encoder (RoBERTa)

Table 1: Code Pre-Trained Models

Lang/Set	**Train**		**Test**													
			A1		**A2**		**B1**		**B2**		**C1**		**C2**		**Overall**	
	Files	**Re-use**	**Files**	**Re use**	**Files**	**Re use**	**Files**	**Re use**	**Files**	**Re use**	**Files**	**Re use**	**Files**	**Re use**	**Files**	**Re use**
Java	259	84	3241	54	3093	47	3268	73	2266	34	124	0	88	14	12080	222
C++/C	79	26	5408	99	5195	86	4939	86	3873	43	335	8	145	0	19895	322

Table 2: SOCO Dataset details

3.4 Experimental Setup

Several experiments were conducted to answer the following questions:

RQ1: What is the effectiveness of source code plagiarism detection when utilising cosine similarity to compare vector representations from various pre-trained code models and selecting the pairs with the highest similarity?

RQ2: Could the performance of the task be enhanced through the application of supervised learning?

RQ3: What impact would unsupervised training with SimCSE have on the task's performance?

The following are the summarised steps of the S-3 approach used for the evaluation of the models without any training.

(1) Encode the codes into vector representations.
(2) Compare each code with all other codes and obtain the similarity score. Select each code as a query and get the top-K results.
(3) The output of the previous steps represents a table of code 1, code 2, and similarity score.
(4) Sort the table in descending order and retrieve the highest N pairs after removing duplicates.
(5) Use the evaluation script provided by the authors of the SOCO dataset to get the values of the classification accuracy metrics.

The goal of training the embeddings is to get similar codes closer in the vector space, while dissimilar codes farther apart in the vector space. The training set in the dataset is used for that purpose. Only the extracted embeddings of the models would be fine-tuned keeping the model weights intact.

For supervised training, the loss was the triplet loss [18] according to Equation 6. The components of the equation are the anchor A, which represents the reference point, P is a positive point that has the same class as A, N is a negative point, and α represents a margin. The distance metric (d) used here is the Euclidean distance. The aim of the triplet loss is to maximise the distance between the positive and negative points related to an anchor. There are different types of triplet loss. The one used in the experiments performed was the batch-all-triplet loss, which computes the loss of all possible triplets. Other configured parameters were the number of epochs equal to 1, and the batch size was 16.

$$\mathcal{L}(A, P, N) = \max\{d(A, P) - d(A, N) + \alpha, 0\} \tag{6}$$

One of the first unsupervised embedding training techniques was SimCSE [15]. SimCSE trains embeddings using contrastive learning. The loss was the Multiple Negative Ranking (MNR) referred to by InfoNCE [31] according to Equation 7. The notations used in the equation are: sim which represents a similarity function (cosine similarity) of the embeddings, and (τ) which represents a temperature. The numerator in the logarithm represents the positive points while the denominator represents the negative points. Positive pairs are created with data augmentation. The aim is to have the minimum loss by imposing a penalty on the model when the similarity between similar sentences is low and providing a reward when the similarity is high. The maximum function would ensure zero output in case of negative values. For the SimCSE experimental setup, the configurable parameters were the following: the number of epochs was 10, the batch size was 16, and the learning rate was 5×10^{-5}.

$$\mathcal{L}_{\text{SimCSE}}(x_i, x_j) = -\log \frac{\exp(\text{sim}(f(x_i), f(x_j))/\tau)}{\sum_{k=1}^{N} \exp(\text{sim}(f(x_i), f(x_k))/\tau)} \tag{7}$$

4 RESULTS

All results of the experiments can be seen in Table 3. The table represents the classification accuracy metrics for each dataset scenario and the overall metrics for all scenarios for both programming languages (Java and C/C++) using various CodePTMs in three different setups (without training, supervised training, and unsupervised training). It can be seen that all the metrics values are equal ($precision = recall = f1$) . This is due to the retrieval of

Tests/PL	Model	Accuracy Metrics (P/R/F1)											
		Java						C/C++					
		A1	A2	B1	B2	C2	Overall	A1	A2	B1	B2	C1	Overall
Without Training	**CodeBERT**	0.574	0.511	0.658	0.529	1	0.608	0.424	0.372	0.419	0.488	0.625	0.422
	GraphcodeBERT	0.630	0.574	0.685	0.559	1	0.649	**0.485**	**0.419**	**0.430**	**0.535**	**0.75**	**0.466**
	UniXcoder	0.611	0.596	0.781	0.765	1	0.712	**0.475**	**0.453**	**0.442**	**0.465**	**0.75**	**0.466**
	CodeBERTa	0.648	0.648	0.781	0.794	1	0.734	0.465	0.395	0.419	0.488	0.750	0.444
	CodeT5	0.648	0.596	0.740	0.706	1	0.698	0.434	0.407	0.442	0.488	0.750	0.444
	PLBART	**0.667**	**0.681**	**0.836**	**0.824**	**1**	**0.770**	0.434	0.349	0.442	0.442	0.750	0.422
Supervised Training	**CodeBERT**	0.574	0.532	0.658	0.500	1	0.608	0.434	0.384	0.407	0.465	0.750	0.425
	GraphcodeBERT	0.685	0.638	0.726	0.559	1	0.689	0.475	0.430	0.465	0.535	0.750	0.475
	UniXcoder	0.741	0.723	0.781	0.735	1	0.766	**0.525**	**0.500**	**0.442**	**0.512**	**0.750**	**0.500**
	CodeBERTa	0.704	0.660	0.767	0.794	1	0.748	0.475	0.419	0.419	0.488	0.750	0.453
	CodeT5	0.648	0.596	0.726	0.706	1	0.694	0.455	0.430	0.419	0.465	0.625	0.444
	PLBART	**0.685**	**0.745**	**0.808**	**0.824**	**1**	**0.779**	0.465	0.430	0.442	0.512	0.750	0.463
Unsupervised Training (SimCSE)	**CodeBERT**	0.556	0.511	0.671	0.529	1	0.608	0.313	0.302	0.360	0.442	0.250	0.339
	GraphcodeBERT	0.519	0.489	0.616	0.500	1	0.572	0.444	0.360	0.407	0.488	0.250	0.413
	UniXcoder	0.611	0.617	0.712	0.618	1	0.671	0.444	0.384	0.360	0.465	0.500	0.410
	CodeBERTa	0.648	0.617	0.795	0.735	1	0.725	0.455	0.372	0.407	0.512	0.750	0.435
	CodeT5	0.611	0.574	0.712	0.706	1	0.676	0.424	0.360	0.395	0.442	0.500	0.404
	PLBART	0.611	0.638	0.740	0.618	1	0.685	0.364	0.314	0.360	0.419	0.500	0.360

Table 3: S-3 SOCO All results

the exact N pairs of the plagiarised candidate pairs. Therefore, the number of output pairs is equal to the number of actual plagiarism cases.

For the results related to the use of the models' embeddings without any training, CodeBERT had the lowest metrics. For the other models, they produced excellent metrics. The highest performance in Java was with PLBART (0.77), while in C/C++ UniXcoder performed the best (0.466). CodeBERT was plainly trained on code with masking and, therefore, does not capture well the semantics of codes. The developers of PLBART and UniXcoder focused on creating robust embeddings for unseen data and tested the generalisation of the embeddings. One of the tasks that UniXcoder was tested on was the zero-shot search, where embeddings can be used for unseen data for code search. These are the reasons behind the high performance of these two models in the SCPD task. It can also be seen that the models that have more representation of the source code, such as PLBART and UniXcoder, generated robust contextual embeddings.

For supervised training, two models, namely CodeBERT and CodeT5, did not benefit from training in Java. Training of CodeBERT should be done using Masked Language Modelling (MLM), and training the embeddings would not yield better results. CodeT5 uses an encoder/decoder architecture, and training only the encoder part limits its capabilities. PLBART remained the best model for Java, yielding an accuracy of 0.779. The accuracy metrics were slightly improved with supervised training given the limited training set of the SOCO dataset. UniXcoder was the model that benefited the most from the training in C/C++ reaching an accuracy of 0.5. Other models also achieved higher accuracy metrics compared to using them out-of-the-box.

Again, unsupervised training had no effect on CodeBERT for the same reasons. However, it had a negative effect on the other models in Java and C/C++, reducing their performance. The main reason is that SimCSE focuses on positive pairs, which are limited in the training set. Furthermore, the high imbalance of the dataset affected the performance of SimCSE.

Several works, as mentioned previously, have used IR approaches for the SOCO dataset. There are two baselines: (1) JPlag and (2) the work of [11] that measured the cosine similarity of 3-grams of source codes. The two baseline results were reported in the original dataset paper [12]. Moreover, the work of DCU [13] created a language model based on parsing, AST and LM. Also, the work of [14] created three models LM (based on IR), LM_AST (indexing with AST) and FLM_AST (indexing with an IR field value and AST). This work would also compare MOSS and Dolos software using them with the same setup by selecting the highest N pairs in terms of similarity with the default parameters, as different setups with different parameters can lead to different results.

For the Java test set as seen in Figure 2, the maximum F1 score was 0.692 in the DCU. Without any training, all the models exceeded this score except CodeBERT and GraphCodeBERT. With the supervised training, the maximum score reached was 0.779 using PLBART-tuned embeddings.

For the C/C++ test set as seen in Figure 3, this work would be compared only to the baselines, MOSS, and Dolos. The maximum F1 score belonging to MOSS was 0.5. The maximum score reached with supervised training was 0.5 which is the highest score on a par with MOSS.

Therefore, this work's simple approach exceeds all these works in terms of the F1 Score in Java and results in the highest F1 Score in C/C++, along with MOSS.

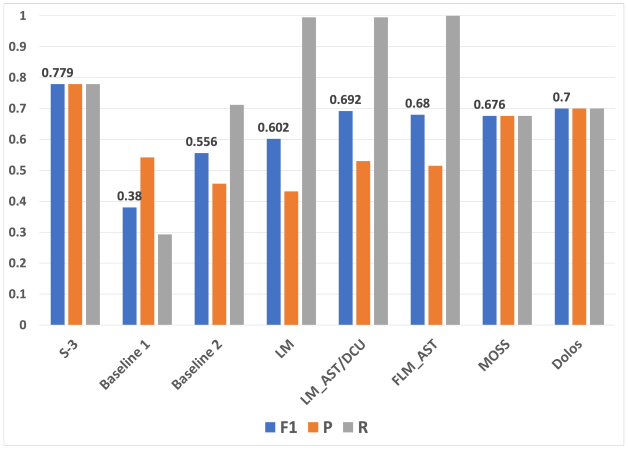

Figure 2: SOCO Java results

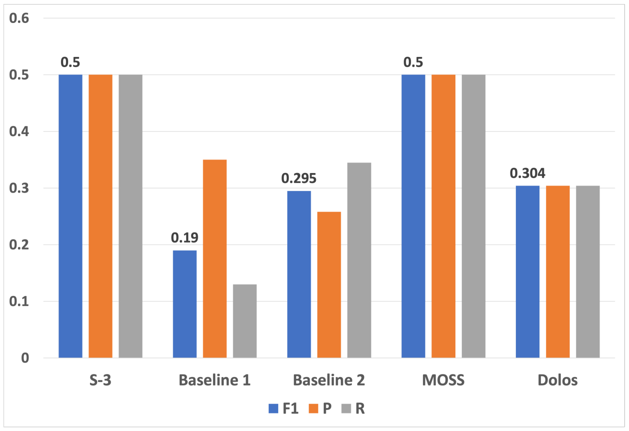

Figure 3: SOCO C/C++ results

After conducting various experiments with six code pre-trained models and three different scenarios (no training, supervised training, and unsupervised training), we would be able to answer the research questions raised in this work:

RQ1: What is the effectiveness of source code plagiarism detection when utilising cosine similarity to compare vector representations from various pre-trained code models and selecting the pairs with the highest similarity?

- The simple S-3 method achieved an excellent performance compared to the other approaches.
- CodeBERT had the lowest F1 score in both PLs.
- Models having multiple representations, such as PLBART and UniXcoder, produced better features.
- Although the models were not initially trained in C/C++, they provided excellent performance, especially UniXcoder and PLBART.
- The small CodeBERTa model achieved excellent accuracy, even with its smaller number of configurable parameters.

RQ2: Could the performance of the task be enhanced through the application of supervised learning?

- Supervised learning had no effect on CodeBERT and CodeT5 in both Java and C/C++.
- The other models had benefited from supervised learning, achieving better overall scores.
- The model that was highly affected by the training was UniXcoder.

RQ3: What impact would unsupervised training with SimCSE have on the task's performance?

- Unsupervised training with SimCSE had a negative impact on the performance on all models on both Java and C/C++ except CodeBERT which had no effect. This could be attributed to the fact that there were few positive pairs as the dataset was imbalanced.

5 CONCLUSION

Source code plagiarism is a common issue in the field of computer science education. Semantic search is about having a query and retrieving similar code files. In this work, given a collection of source codes, semantic similarity was applied to retrieve the most similar pairs of codes. This work started by exploring the robustness of the embeddings generated by Code Pre-Trained Models based on the source code reuse dataset (SOCO). The impact of supervised and unsupervised training training was investigated. The embeddings without any training exceeded the accuracy of plagiarism detection software (JPlag) and other information retrieval methods on the same dataset. Supervised training had a positive impact on accuracy, while unsupervised training with SimCSE had a negative impact on accuracy.

A limitation of semantic search is that it always returns results and scores, even if there are no relevant answers to a query. Another limitation is that the values of K and N are better known from the start. These limitations could be eliminated by using a similarity threshold, which can be an idea for future work. Moreover, there are different ways to train embedding models in an unsupervised manner. Therefore, more experiments could be conducted to check the effect of these techniques. Also some other techniques, like reranking and chunking, can also be explored. Moreover, instead of a similarity search with the calculation of cosine similarity, vector search would be an excellent application for SCPD and can be further investigated. This work focused on pre-ChatGPT datasets and is not compatible with detecting ChatGPT-generated code. A new research field of AI-generated text detection is currently active, and another field specific to code would be an interesting area for future work.

ACKNOWLEDGMENTS

The authors express their gratitude to the anonymous reviewers for providing constructive feedback that helped enhance the quality of the paper. Additionally, they would like to extend their thanks to Rien Maertens, one of the developers of the Dolos software, for his valuable assistance in providing the script required to benchmark the SOCO dataset.

REFERENCES

[1] Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Online, 2655–2668. https://doi.org/10.18653/v1/2021.naacl-main.211

[2] Ibrahim Albluwi. 2019. Plagiarism in programming assessments: a systematic review. *ACM Transactions on Computing Education (TOCE)* 20, 1 (2019), 1–28.

[3] Cîmpeanu Alexandra-Cristina and SL Alexandru Olteanu. 2022. Material Survey on Source Code Plagiarism Detection in Programming Courses. In *2022 International Conference on Advanced Learning Technologies (ICALT)*. IEEE, 387–389.

[4] Rodrigo C Aniceto, Maristela Holanda, Carla Castanho, and Dilma Da Silva. 2021. Source Code Plagiarism Detection in an Educational Context: A Literature Mapping. In *2021 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–9.

[5] Zimin Chen and Martin Monperrus. 2019. A literature study of embeddings on source code. *arXiv preprint arXiv:1904.03061* (2019).

[6] KR1442 Chowdhary. 2020. Natural language processing. *Fundamentals of artificial intelligence* (2020), 603–649.

[7] Kenneth Ward Church. 2017. Word2Vec. *Natural Language Engineering* 23, 1 (2017), 155–162.

[8] Georgina Cosma and Mike Joy. 2008. Towards a definition of source-code plagiarism. *IEEE Transactions on Education* 51, 2 (2008), 195–200.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[10] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics, Online, 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139

[11] Enrique Flores, Alberto Barrón-Cedeno, Paolo Rosso, and Lidia Moreno. 2011. Towards the detection of cross-language source code reuse. In *International Conference on Application of Natural Language to Information Systems*. Springer, 250–253.

[12] Enrique Flores, Paolo Rosso, Lidia Moreno, and Esaú Villatoro-Tello. 2014. On the detection of source code re-use. In *Proceedings of the Forum for Information Retrieval Evaluation*. 21–30.

[13] Debasis Ganguly and Gareth JF Jones. 2014. DCU@ FIRE-2014: an information retrieval approach for source code plagiarism detection. In *Proceedings of the Forum for Information Retrieval Evaluation*. 39–42.

[14] Debasis Ganguly, Gareth JF Jones, Aarón Ramírez-De-La-Cruz, Gabriela Ramírez-De-La-Rosa, and Esaú Villatoro-Tello. 2018. Retrieving and classifying instances of source code plagiarism. *Information Retrieval Journal* 21, 1 (2018), 1–23.

[15] Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple Contrastive Learning of Sentence Embeddings. In *Empirical Methods in Natural Language Processing (EMNLP)*.

[16] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. 2022. Unixcoder: Unified cross-modal pre-training for code representation. *arXiv preprint arXiv:2203.03850* (2022). https://doi.org/10.48550/arXiv.2203.03850

[17] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, et al. 2020. Graphcodebert: Pre-training code representations with data flow. *arXiv preprint arXiv:2009.08366* (2020).

[18] Alexander Hermans, Lucas Beyer, and Bastian Leibe. 2017. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737* (2017).

[19] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436* (2019).

[20] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651* (2016).

[21] Oscar Karnalim, Setia Budi, Hapnes Toba, and Mike Joy. 2019. Source Code Plagiarism Detection in Academia with Information Retrieval: Dataset and the Observation. *Informatics in Education* 18, 2 (2019), 321–344.

[22] Oscar Karnalim, William Chivers, et al. 2019. Similarity detection techniques for academic source code plagiarism and collusion: a review. In *2019 IEEE International Conference on Engineering, Technology and Education (TALE)*. IEEE, 1–8.

[23] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 7871–7880. https://doi.org/10.18653/v1/2020.acl-main.703

[24] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[25] Rien Maertens, Charlotte Van Petegem, Niko Strijbol, Toon Baeyens, Arne Carla Jacobs, Peter Dawyndt, and Bart Mesuere. 2022. Dolos: Language-agnostic plagiarism detection in source code. *Journal of Computer Assisted Learning* 38, 4 (2022), 1046–1061.

[26] Mourad Mars. 2022. From Word Embeddings to Pre-Trained Language Models: A State-of-the-Art Walkthrough. *Applied Sciences* 12, 17 (2022), 8805.

[27] Changan Niu, Chuanyi Li, Bin Luo, and Vincent Ng. 2022. Deep learning meets software engineering: A survey on pre-trained models of source code. *arXiv preprint arXiv:2205.11739* (2022).

[28] Changan Niu, Chuanyi Li, Vincent Ng, Dongxiao Chen, Jidong Ge, and Bin Luo. 2023. An Empirical Comparison of Pre-Trained Models of Source Code. *arXiv preprint arXiv:2302.04026* (2023).

[29] Changan Niu, Chuanyi Li, Vincent Ng, and Bin Luo. 2023. Comparing the Pretrained Models of Source Code by Re-pretraining Under a Unified Setup. *IEEE Transactions on Neural Networks and Learning Systems* (2023).

[30] Matija Novak, Mike Joy, and Dragutin Kermek. 2019. Source-code similarity detection and detection tools used in academia: a systematic review. *ACM Transactions on Computing Education (TOCE)* 19, 3 (2019), 1–37.

[31] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748* (2018).

[32] OpenAI. 2023. GPT-4 Technical Report. https://doi.org/10.48550/ARXIV.2303.08774

[33] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[34] Lutz Prechelt, Guido Malpohl, Michael Philippsen, et al. 2002. Finding plagiarisms among a set of programs with JPlag. *Journal of Universal Computer Science* 8, 11 (2002), 1016.

[35] Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. 2020. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences* 63, 10 (2020), 1872–1897.

[36] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 21, 1 (2020), 5485–5551.

[37] Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084* (2019). https://doi.org/10.48550/arXiv.1908.10084

[38] Hazem Peter Samoaa, Firas Bayram, Pasquale Salza, and Philipp Leitner. 2022. A systematic mapping study of source code representation for deep learning in software engineering. *IET Software* 16, 4 (2022), 351–385.

[39] Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. 2003. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 76–85.

[40] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859* (2021).

[41] Michael J Wise. 1993. String similarity via greedy string tiling and running Karp-Rabin matching. *Online Preprint, Dec* 119, 1 (1993), 1–17.

[42] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771* (2019). https://doi.org/10.48550/arXiv.1910.03771

[43] Man-Fai Wong, Shangxin Guo, Ching-Nam Hang, Siu-Wai Ho, and Chee-Wei Tan. 2023. Natural Language Generation and Understanding of Big Code for AI-Assisted Programming: A Review. *Entropy* 25, 6 (2023), 888.

[44] Frank F Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming*. 1–10.

[45] Morteza Zakeri-Nasrabadi, Saeed Parsa, Mohammad Ramezani, Chanchal Roy, and Masoud Ekhtiarzadeh. 2023. A systematic literature review on source code similarity measurement and clone detection: Techniques, applications, and challenges. *Journal of Systems and Software* (2023), 111796. https://doi.org/10.1016/j.jss.2023.111796

[46] Zhengran Zeng, Hanzhuo Tan, Haotian Zhang, Jing Li, Yuqun Zhang, and Lingming Zhang. 2022. An extensive study on pre-trained models for program understanding and generation. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 39–51.