Augmenting Machine Learning with Information Retrieval to Recommend Real Cloned Code Methods for Code Completion

Augmenting Machine Learning with Information Retrieval to Recommend Real Cloned Code Methods for Code Completion

Muhammad Hammad
Eindhoven University of Technology
Netherlands
m.hammad@tue.nl

Önder Babur
Eindhoven University of Technology
Netherlands
o.babur@tue.nl

Hamid Abdul Basit
Prince Sultan University
Saudi Arabia
hbasit@psu.edu.sa

***Abstract*—Software developers frequently reuse source code from repositories as it saves development time and effort. Code clones accumulated in these repositories hence represent often repeated functionalities and are candidates for reuse in an exploratory or rapid development. In previous work, we introduced DeepClone, a deep neural network model trained by fine tuning GPT-2 model over the BigCloneBench dataset to predict code clone methods. The probabilistic nature of DeepClone output generation can lead to syntax and logic errors that requires manual editing of the output for final reuse. In this paper, we propose a novel approach of applying an information retrieval (IR) technique on top of DeepClone output to recommend real clone methods closely matching the predicted output. We have quantitatively evaluated our strategy, showing that the proposed approach significantly improves the quality of recommendation.**

***Index Terms*—language modeling, deep learning, code clone, code prediction, information retrieval, code search**

I. Introduction

Software developers need effective code search and reuse capability for rapid or exploratory development [1], as writing source code from scratch is an expensive activity. Often, programming of well-defined features amounts to a simple look-up in one's own or others' code in repositories. With the increasing volume of available source code repositories and online resources, it gets more probable to find useful code snippets [2]. Nevertheless, for identifying the relevant parts of the code for reuse [3], developers turn to ad-hoc code reuse [4] with manual searching and selective reading of the source code. It is an expensive and error-prone activity without effective support mechanisms like code snippet search, code predictions, code auto-completion and code generation, to assist them in writing code quickly and correctly. Language modeling is among the most popular methods to realize these features [5]–[7].

In previous work we proposed DeepClone, a deep neural network model trained by fine tuning GPT-2 over the BigCloneBench code clone dataset, for the purpose of clone method predictions. Despite having promising results from various evaluations, DeepClone had a shortcoming: the generated code snippets can contain syntax and logic errors due to the probabilistic nature of the language model, and the specific neural language generation technique applied (nucleus sampling [8]). This eventually requires manual modification making its reuse burdensome (see Table I and Table II for an example).

This problem is not specific to DeepClone, but rather an inherent problem of language models. In natural language generation, it is a well-known challenge to generate well-formed outputs [9], [10]. While recently there are significant advancements in neural language generation techniques, they still cannot match the quality of human authored content (e.g. programs or texts) [11]. They further possess certain problems at their core, notably, standard likelihood training and decoding leads to dull and repetitive outputs [8], and more training data and advanced sampling techniques does not seem to solve this issue entirely [12]. Token-level probabilities predicted by the language models also remain relatively poor [13]. However, the desired output might be a variation of another, previously observed sample [14], [15].

Language models (in our context) are fundamentally probabilistic models, which can generate multiple possible sequences of output (in our case clone methods) based on user context. The space of possible clone methods that could be generated grows exponentially with the length of the clone methods. By having V tokens in the vocabulary, there can be V^N possible clone methods of length N that could be generated. DeepClone model also has a similar problem as it generates clone methods that differ from the real clone methods, which can lead to various syntax and logic errors. For instance, in Table II, "destDir" identifier has been declared in the DeepClone output, but it has not been used anywhere. A fully trained language model can learn patterns in the code such as opening and closing brackets, but it cannot completely learn the logical flow of the code. This motivates our work here, where we seek to build a system that can recommend real clone methods on the basis of user context. Here, a real clone method is taken from a real project, contains the code of some particular functionality "as is", and has been manually validated by the curators of BigCloneBench. Our approach combines the DeepClone model output and information retrieval (IR) techniques to recommend real clone

methods.

Recommending code clones has various benefits. Code clones are useful for exploratory development, where the rapid development of a feature is required and the remedial unification of newly generated clone is not clearly justified [16]. Also, cloned code is expected to be more stable and poses less risk than new development. Hence, we believe that clone methods can be considered a useful component for neural code generation, as they can be used to capture the common code practices of developers, which can be offered as code prediction and completion to the developer.

In this work, we improve the re-usability aspect of code clones by recommending real clone methods using IR techniques to remove noise in the clone methods predicted by neural code generation. We believe that our approach can help in improving the quality of clone prediction on the basis of user context. In this paper, we have made the following contributions:

1) We present a novel approach for recommending real code clone methods by augmenting the DeepClone model output using an IR technique (TF-IDF) for retrieving most similar clone methods from a search corpus.
2) We have quantitatively evaluated our approach in terms of accuracy and effectiveness by calculating various metrices. The overall results show that the proposed approach significantly improves the quality of recommendation from DeepClone.

II. Methodology

We propose a methodology for recommending real clone methods on the basis of given code context by applying IR technique over the DeepClone output. Our starting point is the DeepClone model previously trained on the datasets BigCloneBench and IJaDataset. We have also developed a search corpus comprising of real clone methods from those datasets. Initially, independent of the search corpus, we attempt to predict clone methods consisting of subsequent token sequences starting from the start-of-clone tag $\langle$soc$\rangle$ until the end-of-clone tag $\langle$eoc$\rangle$ ([17]). Then, we apply an IR technique to retrieve real clone methods from the search corpus, which are most similar to the initially generated DeepClone prediction. Figure 1 displays a pictorial representation of our methodology to generate real clone methods (without showing the training steps of DeepClone model, which are shown in our previous paper [17]). In their raw form, we obtain DeepClone output, ground truth, and top-10 samples from the current methodology in an un-formatted style, where code tokens of a clone method are separated only by space characters. To make these outputs readable, we have formatted the code by using online tool [1] along with little manual editing. Table I and Table II contain the formatted output. We describe the details of our methodology in the following subsections.

A. Building the Search Corpus

We have built our search corpus from BigCloneBench and IJaDataset [18], [19], which we have also previously used to train DeepClone. BigCloneBench consists of over 8 million manually validated clone method pairs in IJaDataset 2.0 [20]- a large Java repository of 2.3 million source files (365 MLOC) from 25,000 open-source projects. BigCloneBench contains clones with both syntactic and semantic similarities, along with the references of starting and ending lines of method clones existing in the code repository. In forming this benchmark, methods that potentially implement a given common functionality were identified using pattern based heuristics. These methods were manually tagged as true or false positives of the target functionality by multiple judges. All true positives of a functionality were grouped as a clone class, where a clone class of size n contains $\frac{n(n-1)}{2}$ clone pairs. Currently, BigCloneBench contains clones corresponding to 43 distinct functionalities. Further details can be found in the relevant publications [18], [19].

We have performed several pre-processing steps to build our search corpus, which is similar to what we have followed in our previous work [17]. First, we have extracted the details of a total of 14,922 true positive clone methods, in which 11,920 are distinct clone methods (*Extraction*). Next, we have traced them in IJaDataset files, by following their references from the BigCloneBench dataset, and put them in our search corpus list by placing meta tokens $\langle$soc$\rangle$ at the start, and $\langle$eoc$\rangle$ at the end of each clone method (*Marking*). These meta tokens are also part of the DeepClone output, so inserting them in the search corpus clone method list helps in making a fair comparison. Afterwards, we have normalized each clone method code by removing whitespaces, extra lines, comments, as well as tokenizing (*Normalization and Tokenization*) by adapting Javalang[2] Python library, which contains a lexer and parser for Java 8 programming language. Finally, for each clone method, we have replaced integer, float, binary, and hexadecimal constant values with the $\langle$num_val$\rangle$ meta-token (*Replacement*). Similarly, we have replaced string and character values with $\langle$str_val$\rangle$. Again, this is just to ensure to have fair comparison as DeepClone output is also in this normalized format.

B. Clone Method Prediction by DeepClone

Several neural language generation methods can be used to predict token subsequences from clone methods based on the user context, such as beam search [21], sampling with temperature [22], top-k sampling [11] and nucleus sampling [8]. Each generation method has a specific decoding strategy to shape the probability distribution of language model, such as assigning higher probability to higher quality text. We have used the nucleus sampling method in DeepClone model as it outperforms other methods and is commonly considered the best strategy for generating large amounts of high quality text, comparable to human written text [8]. By having GPT-2

[1]https://www.tutorial spoint.com//online_java_formatter.htm

[2]https://github.com/c2nes/javalang

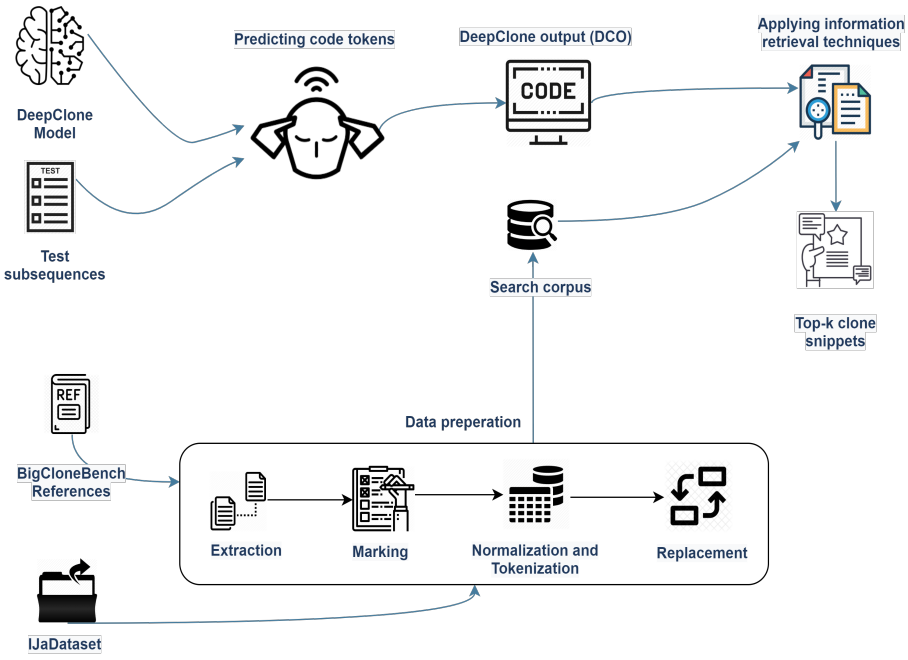

Fig. 1. Methodology of generating real code clones

fine-tuned for DeepClone model, along with nucleus sampling (threshold 0.95), we expect to generate a coherent set of code tokens for clone method predictions. Holtzman et al. [8] have also achieved coherent text generation results with similar settings.

We have performed small scale (100 context queries) experiment to perform next token subsequence prediction by choosing different subsequence sizes such as 10, 20, 30, 50, and 100. Among them, 20 sized token subsequences give us better results in terms of top-k accuracy and MRR. So, we have extracted subsequences of 20 tokens from the testing dataset (developed in previous study), and moved the sliding window one step ahead to obtain further subsequences. Out of a total of 28,197 subsequences containing 20 tokens each, we selected 735 subsequences containing the ⟨soc⟩ token, which indicates the start of a clone method. These subsequences are treated as input queries to generate DeepClone output. We passed these subsequences one by one to our DeepClone model, and we keep on predicting new tokens with nucleus sampling (threshold value 0.95) until the meta-token ⟨eoc⟩ (i.e. end of clone) appears. We use text generation script[3] of HuggingFace Transformer Library in this case. Note that certain parameters, such as the number of subsequences and size of tokens in each subsequence are chosen to perform a preliminary evaluation, which can be fine-tuned and optimized in a follow-up study. The focus of the paper here is to demonstrate the feasibility of our methodology for recommending real clones.

[3]https://github.com/huggingface/transformers/blob/master/examples/text-generation/run_generation.py

C. Retrieving Code Clones from the Search Corpus

The output from the previous step contains set of tokens of context along with the predicted tokens up till the ⟨eoc⟩ token. In this step, we extract only those tokens, which are between ⟨soc⟩ and ⟨eoc⟩ tokens (inclusive) from the DeepClone output (see DeepClone output step in Example Table I and Table II). We apply an IR technique to retrieve top-10 similar results with the generated DeepClone output from the search corpus. IR techniques, in general, are used to discover the significant documents in a large collection of documents, which match a user's query. Their main goal is to identify the significant information that satisfies the user information needs. An IR-based code retrieval method in particular usually extracts from a query a set of keywords and then search for the keywords in code repositories [23].

We have used TF-IDF word embedding based IR technique for retrieving the most similar real clone methods on the basis of DeepClone output. TF-IDF (Term Frequency-Inverse Document Frequency [24]) is often used in IR and text mining. A survey conducted in 2015 showed that 70% of text-based recommendation systems in digital libraries use TF–IDF [25]. Similarly, in the past many researchers have applied TF-IDF technique to retrieve code elements [26], [27]. TF-IDF is a weighting scheme that assigns each term in a document a weight based on its term frequency (TF) and inverse document frequency (IDF). In our context, TF-IDF is looking at the term overlap, i.e. the number of shared tokens between the two clone methods in question (and also how important/significant those tokens are in the clone methods). We use TF-IDF with unigrams as terms to transform clone methods into numeric

vectors, that can easily be compared by quickly calculating cosine similarities. If a term appears frequently in a clone method, that term is probably important in that method: term frequency is simply the number of times that a term appears in a method. However, if a term appears frequently in many clone methods, that term is probably less important generally. Inverse-document frequency is the logarithmically-scaled fraction of clone methods in the corpus in which the term appears. The terms with higher weight scores (high tf *and* idf) are considered to be more important. We first transform clone methods existing in the search corpus and DeepClone output into TF-IDF vectors using equation 1.

$$TF - IDF(i,j) = (1 + \log(TF(i,j)).\log(\frac{J}{DF(i)}) \quad (1)$$

where *TF (i, j)* is the count of occurrences of feature *i* in clone method *j*, and *DF (i)* is the number of clone methods in which feature *i* exists. *J* is the total number of clone methods. During retrieval, we create a normalized TF-IDF sparse vector from the DeepClone output as query, and then take its dot product with the feature matrix. Since all vectors are normalized, the result contains the cosine similarity between the feature vectors of the query and of every clone method. We then return the list of clone methods ranked by their cosine similarities.

III. Empirical Evaluation

To measure the naturalness of different clone methods, we use the perplexity score, while the quality of DeepClone output in terms of ground truth (GT) and top-k recommended clone methods is measured with ROUGE [28] score, which is a measure to compare machine generated text/code with human written text/code. Furthermore, we evaluate whether top-k recommended clone methods match with the ground truth or not. For this purpose, we calculate top-k accuracy and MRR for exact match and functionality type match with the ground truth. All these measures show that the proposed approach significantly helps in generating real clone methods matching the user context.

A. Perplexity

In previous work, it has been observed that n-gram language models can detect defects as they are less "natural" than correct code [29]. Similarly, Karampatsis et al. [30] have noted that defective lines of code have a higher cross-entropy (∼perplexity, to be explained later in this section) than their correct(ed) counterparts. Using the original DeepClone model, the predicted clone method is a buggy snippet, as we observe and can be expected from probabilistic language models. We measure and argue about perplexity scores for the original DeepClone output versus the real clone methods around these angles of naturalness and potential bug density. We expect original DeepClone output to have relatively high perplexity. Perplexity is used to measure the degree of accurately predicting sample data using a language model. At each point in a sequence, it gives an estimate of the average number of code tokens to select from [31]. Perplexity represents a probability distribution over a subsequence or even an entire dataset, and is widely used as a natural evaluation metric for language models. The formula for perplexity is presented in Equation 2:

$$P(L) = exp(-\frac{1}{M}\sum_{i}^{M}\log P(t_i|t_0 : t_{i-1})) \quad (2)$$

where $P(t_i|t_0 : t_{i-1})$ is the conditional probability assigned by the model to the token t at index i. By applying log of conditional probability, cross-entropy loss is calculated. M refers to the length of tokens. Hence, perplexity is an exponentiation of the average cross entropy loss from each token $[0, M]$. To assess how the original output of DeepClone model differs from the real clone code methods, we find the perplexity scores of the clone method predicted by DeepClone and top-k most similar retrieved clone methods. DeepClone output is potentially more noisy and less natural as compared to ground truth (GT) and top-10 recommended snippets in the samples. Table V depicts the mean perplexities of top-10 recommendations and DeepClone output. This clearly displays that DeepClone output has substantial *noise* in it (confirming our observations), some of which attributes to a high density of errors. On the other hand, the top-10 retrieved snippets have relatively low perplexities, which indicates that they are highly natural and less noisy as compared to DeepClone output. There are slight variations in the perplexity values of Top-10 samples, which can be attributed to various factors such as the type of functionality, the number of clone method snippet trained in the DeepClone model, and inner similarity among the clone methods' type. These factors have been discussed in detail in our previous work [17].

B. Exact Match Evaluation

We collect the top 10 recommended clone methods retrieved by our approach, and compute the top-k accuracy (the fraction of times the ground truth clone method appears in the top k recommended clone methods) for k ∈ [1, 10]. Moreover, we measure the Mean Reciprocal Rank (MRR) scores for the recommendations. A simplified description of MRR is that it averages top-k accuracy across various k. In this specific scenario k ∈ [1, 10] since the methodology output a list of top-10 recommended clone methods. The MRR is a rank-based evaluation metric, which produces a value between 0 and 1, where a value closer to 1 indicates a better clone method recommendation system. The reciprocal rank of a query response is the multiplicative inverse of the rank of the first correct answer, while MRR is the average of reciprocal ranks. Table VI shows the top-k accuracy as well as the MRR score. The results display that our methodology has the capability of identifying an exact match between ground truth and top-k recommend clone methods. Table I displays that top-1 recommended clone method exactly matches the ground truth.

TABLE I
AN EXAMPLE CONTAINING SCENARIOS SUCH AS EXACT MATCH, FUNCTIONALITY TYPE MATCH AND METHOD NAME MENTIONED IN THE CONTEXT

<table>
<tr><td>Context</td><td colspan="3">v[i][<num_val>];
}
<soc>
, public static Rectangle2D[][] transpose(Rectangle2D</td></tr>
<tr><td rowspan="2">Ground truth (GT)</td><td colspan="3"><soc> public static Rectangle2D[][] transpose(Rectangle2D[][] a) {
int m = a.length;
int n = a[<num_val>].length;
Rectangle2D[][] t = new Rectangle2D[n][m];
for (int i = <num_val> ; i < m; i++)
for (int j = <num_val> ; j < n; j++)
t[j][i] = a[i][j];
return t;
}
<eoc></td></tr>
<tr><td>Perplexity</td><td colspan="2">1.541</td></tr>
<tr><td rowspan="2">DeepClone output (DCO)</td><td colspan="3"><soc>
public static Rectangle2D[][] transpose(Rectangle2D[][] a, Rectangle2D[][] b) {
int[][] M = new int[a.length][a[<num_val>].length];
for (int i = <num_val> ; i < m.length; i++) {
for (int j = <num_val> ; j < a[i].length; j++) {
M[j][j] = b[i][j];
}
}
return M;
}
<eoc></td></tr>
<tr><td>Perplexity: 3.54</td><td>DCO vs GT</td><td>ROUGE-1: [P: 0.816, R:0.857, F: 0.836], ROUGE-2: [P: 0.676, R: 0.711, F: 0.693], ROUGE-L: [P: 0.862, R: 0.893, F: 0.877]</td></tr>
<tr><td rowspan="2">Top 1</td><td colspan="3"><soc> public static Rectangle2D[][] transpose(Rectangle2D[][] a) {
int m = a.length;
int n = a[<num_val>].length;
Rectangle2D[][] t = new Rectangle2D[n][m];
for (int i = <num_val> ; i < m; i++)
for (int j = <num_val> ; j < n; j++)
t[j][i] = a[i][j];
return t;
}
<eoc></td></tr>
<tr><td>Perplexity: 1.541</td><td>Top 1 vs DCO</td><td>ROUGE-1: [P: 0.816, R: 0.857, F: 0.836], ROUGE-2: [P: 0.676, R: 0.711, F: 0.693], ROUGE-L: [P: 0.862, R: 0.893, F: 0.877]</td></tr>
<tr><td rowspan="2">Top 2</td><td colspan="3"><soc>
public static byte[][] transpose(byte[][] m) {
byte[][] n = new byte[m[<num_val>].length][m.length];
for (int j = <num_val> ; j < m.length; j++)
for (int i = < num_val > ; i < m[<num_val>].length; i++)
n[i][j] = m[j][i];
return n;
}
<eoc></td></tr>
<tr><td>Perplexity: 1.63</td><td>Top 2 vs DCO</td><td>ROUGE-1 [P: 0.777, R: 0.86, F: 0.816], ROUGE-2: [P: 0.627, R: 0.696, F: 0.66], ROUGE-L [P: 0.828, R: 0.923, F: 0.873]</td></tr>
</table>

C. Functionality Type Evaluation

BigCloneBench contains references of multiple implementations (i.e. clones) of specific functionalities. It contains validated clone methods belonging to 43 different functionalities, for instance, ”copy file” functionality contains 3055 different implementations. Further details can be found from our previous paper [17], and BigCloneBench dataset [17]. Hence, it is possible to have recommended clone methods that do not exactly match the ground truth but match its functionality. For instance, Table II displays top-1 and top-2 clone methods belonging to the same functionality type as the ground truth (GT). So, both implementations can potentially satisfy the user's need. For this purpose, we extract the functionality id of the ground truth and recommended list of top-k clone methods against each context. We calculate top-k accuracy and MRR accordingly (see Table VI). The results indicate that we can recommend methods with the matching functionality (if not the exact implementation) with high accuracy, summarized with an MRR score of 0.738.

D. ROUGE Metrics

We have also evaluated our approach by using ROUGE (Recall-Oriented Understudy for Gisting Evaluation) [28]. It is designed to compare an automatically generated summary or translation against a set of reference summaries (typically human-generated). The ROUGE score (scores) allows us to measure the quality of text summarization by computing the frequency of overlapping n-grams between the produced summary and the reference one(s). Various forms of overlapping units, such as n-grams, word sequences, and word pairs, are counted between the auto-generated summary and the reference summaries. In our context, it helps us to automatically determine the quality of original DeepClone output by

TABLE II
AN EXAMPLE CONTAINING SCENARIOS SUCH AS FUNCTIONALITY TYPE MATCH AND METHOD NAME NOT MENTIONED IN THE CONTEXT

Context

```
substring(i + <num_val> ).toLowerCase();
}
return ext;
}
<soc>
public static void
```

Ground truth (GT)

```
<soc>
public static void copyFile(File in , File out) throws IOException {
    FileChannel inChannel = new FileInputStream( in ).getChannel();
    FileChannel outChannel = new FileOutputStream(out).getChannel();
    try {
        inChannel.transferTo( <num_val> , inChannel.size(), outChannel);
    } catch (IOException e) {
        throw e;
    } finally {
        if (inChannel != null) inChannel.close();
        if (outChannel != null) outChannel.close();
    }
}
<eoc>
```

Perplexity	1.21

DeepClone output (DCO)

```
<soc>
public static void copyStreams(File source, File dest, File destDir) throws IOException
{
    BufferedInputStream in = null;
    BufferedOutputStream out = null;
    try { in = new BufferedInputStream(new FileInputStream(source));
        out = new BufferedOutputStream(new FileOutputStream(dest));
        out.write(source.getAbsolutePath());
        int len = in .read();
        while (len > <num_val> ) {
            out.write(len);
        } in .close();
        out.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    } finally {
        if ( in != null) in .close();
        if (out != null) out.close();
    }
}
<eoc>
```

Perplexity	9.246	**DCO vs GT**	**ROUGE-1:** [P:0.511, R: 0.728, F: 0.601], **ROUGE-2:** [P:0.331, R:0.473, F: 0.389], **ROUGE-L:** [P:0.543, R:0.676, F: 0.602]

Top 1

```
<soc>
public static void copy(File source, File dest) throws Exception {
    FileInputStream in = new FileInputStream(source);
    FileOutputStream out = new FileOutputStream(dest);
    int c;
    while ((c = in .read()) != - <num_val> ) out.write(c); in .close();
    out.close();
}
<eoc>
```

Perplexity: 1.439	**Top 1 vs DCO**	**ROUGE-1:** [P: 0.443, R: 0.853, F: 0.583], **ROUGE-2:** [P: 0.331, R: 0.642, F: 0.437], **ROUGE-L:** [P: 0.565, R: 0.812,F: 0.667]

Top 2

```
<soc>
public static void copyFile(String source, String dest) throws IOException {
    FileChannel in = null, out = null;
    try { in = new FileInputStream(new File(source)).getChannel();
        out = new FileOutputStream(new File(dest)).getChannel();
        in.transferTo(<num_val> , in.size(), out);
    } finally {
        if ( in != null) in .close();
        if (out != null) out.close();
    }
}
<eoc>
```

Perplexity: 1.321	**Top 2 vs DCO**	**ROUGE-1** [P:0.618, R: 0.835 ,F: 0.711], **ROUGE-2** [P:0.454, R: 0.615, F: 0.522], **ROUGE-L:** [P: 0.522, R: 0.686, F: 0.593]

TABLE III
EMPIRICAL EVALUATION RESULTS BETWEEN DEEPCLONE OUTPUT (DCO) AND TOP 10 RECOMMENDED CLONES

	Top 1	Top (2-4)	Top (5-10)
ROUGE1			
Precision	0.666 ± 0.204	0.637 ± 0.197	0.612 ± 0.196
Recall	0.599 ± 0.214	0.587 ± 0.181	0.576 ± 0.181
F-measure	0.552 ± 0.157	0.559 ± 0.189	0.567 ± 0.182
ROUGE-2			
Precision	0.485 ± 0.225	0.457 ± 0.21	0.425 ± 0.203
Recall	0.403 ± 0.187	0.387 ± 0.168	0.374 ± 0.167
F-measure	0.362 ± 0.149	0.353 ± 0.173	0.361 ± 0.163
ROUGE-L			
Precision	0.631 ± 0.175	0.625 ± 0.17	0.601 ± 0.167
Recall	0.591 ± 0.177	0.58 ± 0.154	0.565 ± 0.148
F-measure	0.556 ± 0.136	0.549 ± 0.145	0.548 ± 0.151

comparing it with the ground truth and top-10 recommended clone methods. ROUGE doesn't try to assess how fluent the clone method is. It only tries to assess the adequacy, by simply counting how many n-grams in the DeepClone output matches the n-grams in the ground truth and top-k recommended clone methods. Because ROUGE is based only on token overlap, it can determine if the same general concepts are discussed between an automatic summary and a reference summary, but it cannot determine if the result is coherent or the clone method is semantically correct. High-order n-gram ROUGE measures try to judge fluency to some degree. In this paper, we evaluate the quality of DeepClone output with respect to the ground truth and recommended top-10 clone methods by calculating the scores of ROUGE-1, ROUGE-2, and ROUGE-L as most authors use them for automatic evaluation score besides human evaluation [32], [33]. We have calculated precision (P), recall (R), and F-measure (F) of ROUGE-1 ROUGE-2, and ROUGE-L between various combinations such as DeepClone output and ground truth (DCO vs GT), DeepClone output and top-10 recommended clone methods (top-1...10 vs DCO). ROUGE-1 refers to the overlap of unigrams between some reference output and the output to be evaluated. ROUGE-2, in turn, checks for bigrams instead of unigrams. The reason one would use ROUGE-1 over or in conjunction with ROUGE-2 (or other finer granularity ROUGE measures), is to also indicate fluency as a part of the evaluation. The intuition is that the prediction is more fluent if it more closely follows the word orderings of the reference snippet. Finally, ROUGE-L measures longest matching sequence of tokens between machine generated text/code and human produced one by using longest common subsequence (LCS). Using LCS has a distinguishing advantage in evaluation: it captures in-sequence (i.e. sentence level flow and word order) matches rather than strict consecutive matches.

DeepClone predicted clone method can be extremely long, capturing all tokens in the retrieved clone methods, but many of these tokens may be useless, making it unnecessarily verbose. This is where precision comes into play. It measures what portion of the DeepClone output was in fact relevant and desirable to be kept with respect to the reference output.

$$\text{Precision} = \frac{\text{\# of overlapping tokens}}{\text{total tokens in the predicted output}} \quad (3)$$

Recall in the context of ROUGE measures what portion of the reference output was successfully captured by the DeepClone output.

$$\text{Recall} = \frac{\text{\# of overlapping tokens}}{\text{total \# tokens in reference snippet}} \quad (4)$$

We also report the F-measure which provides a single score that balances both the concerns of precision and recall.

$$\text{F-Measure} = 2 * \frac{\text{Precision * Recall}}{\text{Precision + Recall}} \quad (5)$$

IV. RESULTS AND DISCUSSION

The approach we have proposed leads to promising results. We observe quite high mean perplexity scores and standard deviation of DeepClone output (11.624 ± 5.892), This indicates high noise and less natural code, which is a known problem of neural language generation [9], [10]. However, we notice quite low mean perplexity scores and standard deviation for the ground truth (2.047 ± 0.848) and top-10 recommended clone methods (range from 1.79 ± 0.716 till 1.907 ± 0.612) against the set of 735 input queries. These numbers show that our methodology can improve the original prediction by DeepClone, resulting in more natural snippets. We have further calculated top-k accuracy and MRR involving an exact match of the recommended clone methods with the ground truth. We achieve an accuracy of 39.3% in the top 10 recommended clone methods and MRR as 28.3% (see Table VI). In a fair share of the cases, we can find exactly the same clone method as in the ground truth. As BigCloneBench contains references to various implementations for each of the 43 functionalities, it is quite possible that the user is recommended a different snippet than the ground truth, yet implementing the same functionality. This alternative can also help the developer achieve their goal. To assess such cases, we have calculated the top-k accuracy and MRR taking alternative implementations into account. We achieve quite a high accuracy, notably 84.1% in the top 10 recommended clone methods, as well as 73.8% MRR in terms of functionality type match with the ground truth (see Table VI). This is a major improvement over the exact match scores and further reinforces the claims we make for our approach.

Furthermore, we have measured different ROUGE scores, i.e. ROUGE-1, ROUGE-2, and ROUGE-L, to evaluate the similarity (and the quality to a certain extent) of the DeepClone output to ground truth and top-10 recommended clone methods. After retrieving the top-10 recommendations, we have calculated the precision, recall and F-measure (as explained in Section III-D) for each ROUGE metric. Note that our approach ranks the recommendations with respect to their cosine similarity. Thus it is possible that when we compare DeepClone output with top-10 recommended clone methods using ROUGE, precision, recall and F-measure may not come

TABLE IV
EMPIRICAL EVALUATION RESULTS BETWEEN DEEPCLONE OUTPUT (DCO) AND GROUND TRUTH (GT)

	DCO vs GT
ROUGE-1	
Precision	0.667 ± 0.192
Recall	0.559 ± 0.226
F-measure	0.56 ± 0.185
ROUGE-2	
Precision	0.479 ± 0.217
Recall	0.398 ± 0.218
F-measure	0.4 ± 0.202
ROUGE-L	
Precision	0.652 ± 0.165
Recall	0.586 ± 0.183
F-measure	0.599 ± 0.153

TABLE V
PERPLEXITIES

	Perplexity
DCO	11.624 ± 5.892
GT	2.047 ± 0.848
1	1.79 ± 0.716
2	1.851 ± 0.726
3	1.795 ± 0.594
4	1.800 ± 0.491
5	1.874 ± 0.544
6	1.907 ± 0.612
7	1.887 ± 0.578
8	1.863 ± 0.505
9	1.855 ± 0.545
10	1.859 ± 0.578

TABLE VI
MRR AND TOP-K ACCURACIES

	Exact Match	Functionality Type Match
MRR	0.283	0.738
Top-1	0.233	0.692
Top-3	0.316	0.770
Top-5	0.355	0.797
Top-10	0.393	0.841

in a strict descending order given the different characteristics of each ROUGE metric (see Table III). Cosine similarity is a very common method for content-based evaluation, while ROUGE scores are used to evaluate machine generated against human produced content [28]. That is why we observe that F-measure for ROUGE-1 of Top (2-4) is slightly higher than Top-1 III.

In our qualitative investigation, we experienced two different scenarios based on the recommended output. The first one is the ideal scenario when one of the top-k recommended clone methods exactly match the ground truth. In the example given in Table I, "transpose" clone method implementation at top-1 exactly matches the ground truth. This scenario gives the best results. The second scenario is when none of the top-k recommended methods exactly match the ground truth but at least one of the top-k recommended clone method functionality type matches with functionality type of the ground truth. In Table II, although top-1 and top-2 recommended clone methods do not exactly match with the ground truth, they belong to the same functionality type "copy file". The main advantage of our methodology is that even if the recommended clone methods do not exactly match the ground truth, still they would be usually implementing the same functionality as the ground truth method, and might satisfy the user's need.

Similarly, there are two scenarios based on the input context. The first scenario is when the context contain the name of method. It is straightforward for the neural language technique to generate the predicted clone method following the given method name and current context. Table I gives an example of this scenario, where "transpose" method name is mentioned in the context and our approach recommends clone methods as top-1 and top-2, whose functionality type matches with the functionality type of the ground truth. The second scenario is based on the context that does not contain a method name. This can have two different output sub-scenarios. The first one is when the functionality type of the recommended clone method and the ground truth do not match. As we see in Table APPENDIX, the context does not have the full signature of the clone method. This makes the generated output by DeepClone using nucleus sampling deviate from the functionality type of the ground truth. Ground truth belongs to "copy file" functionality, while DeepClone output belongs to "delete directory" functionality type, which eventually leads to TF-IDF recommending clone methods as top-1 and top-2 on the basis of "delete directory". Such scenarios eventually result in low and largely deviating ROUGE scores between the DeepClone output and the ground truth (see Table III and the example in Table APPENDIX). There we observe that it also affects the other evaluation measures involving exact and functionality type matches. These clone methods eventually may not fulfil the desired goal of the user (see Table VI). So, it might be useful to guide the users to include the clone method name in the context for better results. The other output sub-scenario is when we manage to successfully generate DeepClone output whose functionality type matches with the ground truth. In Table II, "copy file" method name is not mentioned in the context, but the functionality type of the DeepClone output matches with the ground truth, which eventually helps TF-IDF to retrieve clone methods on the basis of DeepClone output. We notice that the total number of "copy file" clone methods used in DeepClone training are 2,454, which allows nucleus sampling to generate DeepClone output closer to ground truth in example II. Overall we believe our approach yields very promising results and can assist the developers by recommending real and accurate clone methods.

V. LIMITATIONS AND THREATS TO VALIDITY

The proposed approach is the first step towards recommending real code clone methods on the basis of user context. However, it has certain limitations as well. Despite the fact that the dataset used in this study is collected from a well-known clone code dataset (BigCloneBench), it does not necessarily mean the codebase used in this study represents Java language source code entirely (a threat to external validity in terms of generalizability). Another issue is that we have selected and used various parameter/threshold values and techniques with the goal of showcasing the feasibility of our approach. As an example, for generating predicted clone methods, we only used nucleus sampling with threshold value of 0.95 [8]. There are various other text generation methods such as beam search

[21], sampling with temperature [22], and top-k sampling [11], which can be explored for generating clone methods on the basis of user context. Similarly, threshold values can be tuned to get the best results. Based on experimentation, we determined certain parameters (e.g. 735 as the number of subsequences, 20 as the number of tokens per subsequence) aiming to demonstrate a preliminary evaluation of our methodology. However, by having different parameters, e.g. having subsequences of different sized tokens, and using the complete set of queries, we can have different results.

Another limitation involves the normalization step we have performed. We have replaced integer, float, binary, and hexadecimal constant values with the ⟨num_val⟩ meta-token. Similarly, we have replaced string and character values with ⟨str_val⟩. This reduces our vocabulary size, which leads to faster training of the model, but also reduces the vocabulary of the predictions. We nevertheless note that technique has been used by several researchers in the same manner for data preparation [30], [34]. Similarly, in order to have fair comparison between DeepClone output and clone methods available in search corpus, we have built a search corpus in the same format as we have used for DeepClone. This helps the TF-IDF technique to recommend clone methods accordingly.

In this study, we only apply TF-IDF, an IR technique to retrieve the most similar real clone methods, on the basis of the predicted clone method. However, there are other IR techniques such as GLOVE [35], and word2vec [36], which can be additionally explored. We leave it for future work to comparatively assess and optimize the parameters and techniques for our approach.

VI. Related Work

In this section, we present related work covering neural language generation techniques, and the role of recommendation systems in the field of code clones.

A. Neural Language Generation

To the best of our knowledge, no technique has been presented to improve the prediction of clone methods. However, many techniques have been introduced to improve the quality of generated program code and text. Hashimoto et al. [15] proposed an approach to predict python code tokens, first by retrieving a training example based on the input (e.g., natural language description) and then editing it to the desired output (e.g., code). Song et al. [14] proposed a novel ensemble of retrieval-based and generation-based dialog systems. They first obtained a candidate response on the basis of user utterance or query by applying IR technique from a large database. Then, they passed retrieved candidate and query to an RNN-based response generator, so that the neural model is aware of more information. The generated response is then fed back as a new candidate for post-reranking. Zhou el al. have proposed Lancer [37], a new context-aware code-to-code recommending tool leveraging a Library-Sensitive Language Model and a BERT model to recommend relevant code samples in real-time, by automatically analyzing the intention of the incomplete code. Lancer uses BERT model to complete an incomplete code sample, then it retrieves the relevant real code samples on the basis of Elastic search and rank them according to the deep semantic ranking scheme. The major difference with our methodology is that they used BERT model, whose intention is to complete the missing tokens in the incomplete code, while we are using fine-tuned GPT2 model in DeepClone, which is used to predict next tokens on the basis of context. DeepClone is fine-tuned on both non-clone and clone code patterns, but Lancer only used clone methods for training. Moreover, we can also generate correct DeepClone output when the context does not even contain a method name, a scenario that is apparently not covered by Lancer.

B. Code Clone Recommendation Systems

We could only find a single piece of work which suggests using clone methods for code recommendation [38]. However, they use a different similarity measure based on API calls for recommending relevant clone methods. Clones are generally considered to be harmful for a software system, and mainly researchers work on techniques for avoiding and eliminating clones [39]–[42]. Clone refactoring recommendation systems have been developed for this purpose. For instance, Yoshida et al. [39] proposed a proactive clone recommendation system for "Extract Method" refactoring, while Wang et al. [40] introduced an approach for automatically recommending clones for refactoring using a decision tree-based classifier.

VII. Conclusion and Future Work

In this work, we have proposed a recommendation system to suggest real code clones by using IR techniques. This system significantly improves the original prediction by DeepClone, a deep learning model we previously developed. We have have performed quantitative evaluation using a wide range of metrics, and qualitatively discussed additional scenarios, to support our claim. In the future, we plan to perform a comparative study by evaluating different IR techniques such as BERT; pretrained word embedding techniques such as word2vec [36] and GLOVE [35]; and code query formulation techniques [23], [43].

Acknowledgment

We acknowledge the contribution of Dr. Sohaib Khan (CEO at Hazen.ai) for providing us valuable feedback on methodology and empirical evaluation parts.

References

[1] C. Sadowski, K. T. Stolee, and S. Elbaum, "How developers search for code: a case study," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 191–201.

[2] M. Gabel and Z. Su, "A study of the uniqueness of source code," in *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, 2010, pp. 147–156.

[3] S. E. Sim, C. L. Clarke, and R. C. Holt, "Archetypal source code searches: A survey of software developers and maintainers," in *Proceedings. 6th International Workshop on Program Comprehension. IWPC'98 (Cat. No. 98TB100242)*. IEEE, 1998, pp. 180–187.

[4] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, "Do code clones matter?" in *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 2009, pp. 485–495.

[5] L. Mou, R. Men, G. Li, L. Zhang, and Z. Jin, "On end-to-end program generation from user intention by deep neural networks," *arXiv preprint arXiv:1510.07211*, 2015.

[6] T. T. Nguyen, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "A statistical semantic language model for source code," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 2013, pp. 532–542.

[7] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 2018, pp. 933–944.

[8] A. Holtzman, J. Buys, M. Forbes, and Y. Choi, "The curious case of neural text degeneration," *arXiv preprint arXiv:1904.09751*, 2019.

[9] J. Li, W. Monroe, T. Shi, S. Jean, A. Ritter, and D. Jurafsky, "Adversarial learning for neural dialogue generation," *arXiv preprint arXiv:1701.06547*, 2017.

[10] L. Shao, S. Gouws, D. Britz, A. Goldie, B. Strope, and R. Kurzweil, "Generating high-quality and informative conversation responses with sequence-to-sequence models," *arXiv preprint arXiv:1701.03185*, 2017.

[11] A. Fan, M. Lewis, and Y. Dauphin, "Hierarchical neural story generation," *arXiv preprint arXiv:1805.04833*, 2018.

[12] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

[13] S. Welleck, I. Kulikov, S. Roller, E. Dinan, K. Cho, and J. Weston, "Neural text generation with unlikelihood training," *arXiv preprint arXiv:1908.04319*, 2019.

[14] Y. Song, R. Yan, X. Li, D. Zhao, and M. Zhang, "Two are better than one: An ensemble of retrieval-and generation-based dialog systems," *arXiv preprint arXiv:1610.07149*, 2016.

[15] T. B. Hashimoto, K. Guu, Y. Oren, and P. S. Liang, "A retrieve-and-edit framework for predicting structured outputs," in *Advances in Neural Information Processing Systems*, 2018, pp. 10 052–10 062.

[16] C. J. Kapser and M. W. Godfrey, ""cloning considered harmful" considered harmful: patterns of cloning in software," *Empirical Software Engineering*, vol. 13, no. 6, p. 645, 2008.

[17] M. Hammad, O. Babur, H. A. Basit, and M. v. d. Brand, "Deepclone: Modeling clones to generate code predictions," *arXiv preprint arXiv:2007.11671*, 2020.

[18] J. Svajlenko, J. F. Islam, I. Keivanloo, C. K. Roy, and M. M. Mia, "Towards a big data curated benchmark of inter-project code clones," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 476–480.

[19] J. Svajlenko and C. K. Roy, "Bigcloneeval: A clone detection tool evaluation framework with bigclonebench," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2016, pp. 596–600.

[20] "Ambient software evoluton group, ijadataset 2.0," http://secold.org/projects/seclone, 2020 (accessed May 8, 2020).

[21] A. K. Vijayakumar, M. Cogswell, R. R. Selvaraju, Q. Sun, S. Lee, D. Crandall, and D. Batra, "Diverse beam search for improved description of complex scenes," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[22] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[23] L. Nie, H. Jiang, Z. Ren, Z. Sun, and X. Li, "Query expansion based on crowd knowledge for code search," *IEEE Transactions on Services Computing*, vol. 9, no. 5, pp. 771–783, 2016.

[24] M. Dillon, "Introduction to modern information retrieval: G. salton and m. mcgill. mcgraw-hill, new york (1983). xv+ 448 pp." 1983.

[25] J. Beel, B. Gipp, S. Langer, and C. Breitinger, "paper recommender systems: a literature survey," *International Journal on Digital Libraries*, vol. 17, no. 4, pp. 305–338, 2016.

[26] K. Kim, D. Kim, T. F. Bissyandé, E. Choi, L. Li, J. Klein, and Y. L. Traon, "Facoy: a code-to-code search engine," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 946–957.

[27] S. Luan, D. Yang, C. Barnaby, K. Sen, and S. Chandra, "Aroma: Code recommendation via structural code search," *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 1–28, 2019.

[28] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[29] B. Ray, V. Hellendoorn, S. Godhane, Z. Tu, A. Bacchelli, and P. Devanbu, "On the" naturalness" of buggy code," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 2016, pp. 428–439.

[30] R.-M. Karampatsis, H. Babii, R. Robbes, C. Sutton, and A. Janes, "Big code!= big vocabulary: Open-vocabulary models for source code," 2020.

[31] M. Allamanis and C. Sutton, "Mining source code repositories at massive scale using language modeling," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 207–216.

[32] A. Moeed, Y. An, G. Hagerer, and G. Groh, "Evaluation metrics for headline generation using deep pre-trained embeddings," in *Proceedings of The 12th Language Resources and Evaluation Conference*, 2020, pp. 1796–1802.

[33] R. Paulus, C. Xiong, and R. Socher, "A deep reinforced model for abstractive summarization," *arXiv preprint arXiv:1705.04304*, 2017.

[34] M. White, C. Vendome, M. Linares-Vásquez, and D. Poshyvanyk, "Toward deep learning software repositories," in *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE Press, 2015, pp. 334–345.

[35] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[36] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[37] S. Zhou, B. Shen, and H. Zhong, "Lancer: Your code tell me what you need," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 1202–1205.

[38] S. Abid, "Recommending related functions from api usage-based function clone structures," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 1193–1195.

[39] N. Yoshida, S. Numata, E. Choiz, and K. Inoue, "Proactive clone recommendation system for extract method refactoring," in *2019 IEEE/ACM 3rd International Workshop on Refactoring (IWoR)*. IEEE, 2019, pp. 67–70.

[40] W. Wang and M. W. Godfrey, "Recommending clones for refactoring using design, context, and history," in *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 2014, pp. 331–340.

[41] H. A. Basit, M. Hammad, S. Jarzabek, and R. Koschke, "What do we need to know about clones? deriving information needs from user goals," in *2015 IEEE 9th International Workshop on Software Clones (IWSC)*. IEEE, 2015, pp. 51–57.

[42] M. Hammad, H. A. Basit, S. Jarzabek, and R. Koschke, "A systematic mapping study of clone visualization," *Computer Science Review*, vol. 37, p. 100266, 2020.

[43] M. Lu, X. Sun, S. Wang, D. Lo, and Y. Duan, "Query expansion via wordnet for effective code search," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 545–549.

TABLE APPENDIX

AN EXAMPLE CONTAINING SCENARIOS SUCH AS FUNCTIONALITY TYPE NOT MATCHED AND METHOD NAME NOT MENTIONED IN THE CONTEXT

Context

```
MAX_IO_CHUNK_SIZE = <num_val> * <num_val> * <num_val>;
private static final long LASTMODIFIED_DIFF_MILLIS = <num_val> ;
<soc> public static boolean
```

Ground truth (GT)

```
<soc> public static boolean copy(File from, File to, Override override) throws IOException {
    FileInputStream in = null;
    FileOutputStream out = null;
    FileChannel srcChannel = null;
    FileChannel destChannel = null;
    if (override == null) override = Override.NEWER;
    switch (override) {
        case NEVER:
            if (to.isFile()) return <num_val>;
            break;
        case NEWER:
            if (to.isFile() && (from.lastModified() - LASTMODIFIED_DIFF_MILLIS) < to.lastModified
()) return <num_val> ;
            break;
    }
    to.getParentFile().mkdirs();
    try { in = new FileInputStream(from);
        out = new FileOutputStream(to);
        srcChannel = in .getChannel();
        destChannel = out.getChannel();
        long position = <num_val> ;
        long count = srcChannel.size();
        while (position < count) {
            long chunk = Math.min(MAX_IO_CHUNK_SIZE, count - position);
            position += destChannel.transferFrom(srcChannel, position, chunk);
        }
        to.setLastModified(from.lastModified());
        return <num_val > ;
    } finally {
        CommonUtils.close(srcChannel);
        CommonUtils.close(destChannel);
        CommonUtils.close(out);
        CommonUtils.close(in);
    }
} <eoc>
```

Perplexity 3.109

DeepClone output (DCO)

```
<soc>
public static boolean deleteDirectory(File f) {
    try {
        final File[] fFiles = f.listFiles();
        for (int i = <num_val> ; i < fFiles.length; i++) {
            if (fFiles[i].isDirectory()) deleteDirectory(fFiles[i]);
        }
    } catch (Exception e) {
        System.out.println(<str_val> +f + <str_val> +i + <str_val>);
        return <num_val> ;
    }
    return <num_val> ;
}
<eoc>
```

Perplexity 6.137 | **DCO vs GT** | **ROUGE-1:** [P:0.551, R: 0.216, F: 0.31], **ROUGE-2:** [P:0.295, R:0.115, F: 0.166], **ROUGE-L:** [P: 0.474, R:0.281, F: 0.353]

Top 1

```
<soc> private void deleteDirectory(File dir) {
    File[] a = dir.listFiles();
    if (a != null) {
        for (File f: a) {
            if (f.isDirectory()) {
                deleteDirectory(f);
            } else {
                f.delete();
            }
        }
    }
    dir.delete();
} <eoc>
```

Perplexity: 1.887 | **Top 1 vs DCO** | **ROUGE-1:** [P: 0.494, R: 0.677, F: 0.571], **ROUGE-2:** [P: 0.295, R: 0.406, F: 0.342], **ROUGE-L:** [P: 0.447, R:0.654,F: 0.531]

Top 2

```
<soc> public static void deleteDirectory(File dir) {
    File[] files = dir.listFiles();
    for (File f: files) {
        if (f.isDirectory()) {
            deleteDirectory(f);
        } else f.delete();
    }
    dir.delete();
} <eoc>
```

Perplexity:1.831 | **Top 2 vs DCO** | **ROUGE-1** [P:0.494, R: 0.786 ,F: 0.607], **ROUGE-2** [P:0.318, R: 0.509, F: 0.392], **ROUGE-L:** [P: 0.5, R: 0.76, F: 0.603]