Received November 27, 2021, accepted January 5, 2022, date of publication January 21, 2022, date of current version February 2, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3145686

Clone-Seeker: Effective Code Clone Search Using Annotations

MUHAMMAD HAMMAD[1], ÖNDER BABUR[1,2], HAMID ABDUL BASIT[3], AND MARK VAN DEN BRAND[1]

[1]Department of Mathematics and Computer Science, Eindhoven University of Technology, 5612 AZ Eindhoven, The Netherlands
[2]Information Technology Group, Wageningen University and Research, 6708 PB Wageningen, The Netherlands
[3]Department of Software Engineering, Prince Sultan University, Riyadh 12435, Saudi Arabia

Corresponding author: Muhammad Hammad (m.hammad@tue.nl)

This work was supported by Prince Sultan University (by paying the article processing charges (APC) for this publication).

ABSTRACT Source code search plays an important role in software development, e.g. for exploratory development or opportunistic reuse of existing code from a code base. Often, exploration of different implementations with the same functionality is needed for tasks like automated software transplantation, software diversification, and software repair. Code clones, which are syntactically or semantically similar code fragments, are perfect candidates for such tasks. Searching for code clones involves a given search query to retrieve the relevant code fragments. We propose a novel approach called Clone-Seeker that focuses on utilizing clone class features in retrieving code clones. For this purpose, we generate metadata for each code clone in the form of a natural language document. The metadata includes a pre-processed list of identifiers from the code clones augmented with a list of keywords indicating the semantics of the code clone. This keyword list can be extracted from a manually annotated general description of the clone class, or automatically generated from the source code of the entire clone class. This approach helps developers to perform code clone search based on a search query written either as source code terms, or as natural language. With various experiments, we show that (1) Clone-Seeker is effective in finding clones from BigCloneBench dataset by applying code queries and natural language queries; 2) Clone-Seeker has a higher recall when searching for semantic code clones (i.e., Type-4) in BigCloneBench than the state-of-the-art; 3) Clone-Seeker is a generalized technique as it is effective in finding clones in Project CodeNet dataset by applying code queries and natural language queries. 4) Clone-Seeker with manual annotation outperforms other variants in finding clones on the basis of natural language queries.

INDEX TERMS Annotation, code clone, code clone search, keyword extraction, information retrieval.

I. INTRODUCTION

Software plays a central role in society, touching billions of lives on a daily basis. Writing and maintaining source code is a core activity for software developers, who aim to provide reliable and functional software [1]. One of the challenges a software developer faces when writing new code is to find out how to implement a certain functionality (e.g., how to implement quick sort) [2]. The implementation of such functionality might already be realized by other developers and can be reused rather than written from scratch. Over the years, a huge number of open source and industrial software systems have been developed and the source code of these systems is typically stored in source code repositories such as GitHub. This source code can be treated as an important reusable asset for developers [3].

Many software development and maintenance tasks rely on effective code search [4] to find the source code related to a specific functionality. In modern software development, developers often refer to web search engines in order to search for code examples from large amounts of online resources such as GitHub, online tutorials, technology blogs, API documents, social media posts, etc. [5], [6]. Indeed, code search is an integral part of software development; developers spend up to 19% of their development time on code search [7]. Similarly, studies have even revealed that more than 60% of developers search for source code examples every day [8], [9].

Source code examples or code fragments can help developers understand how others addressed the similar problems [10]–[15] and can serve as a basis for writing new

The associate editor coordinating the review of this manuscript and approving it for publication was Giuseppe Destefanis.

programs [16], [17]. These code examples can accelerate the development process [18] and increase the product quality [19]. A working code example can be considered for both learning and pragmatic reuse. Such working code examples can spawn a wide range of applications, varying from API usage (e.g., how to use the JFreeChart library[1] to display a chart in Java) to basic algorithmic problems (e.g., how to implement quick sort). An ideal working code example should be concise, complete, self-contained, and easy to understand and reuse. Code clones can be considered as the ideal code examples as they are more stable and possess less risk than new development [20]–[22].

Code clones are usually categorized as Type-1, Type-2, Type-3 and Type-4 [23]–[26] clones, based on their level of similarity with each other. Type-1 clones are same code fragments, except for dissimilarities in comments, layout, and whitespace. Type-2 clones refer to same code fragments, except for dissimilarities in literal values and identifier names, in addition to Type-1 clone dissimilarities. Type-3 clones are syntactically similar code fragments with differences in their statements. Such fragments can contain additions, modifications and/or removals to their statements with respect to each other, in addition to the changes allowed for Type-1/-2 clones. Type-4 clones are also known as semantic clones, which have similar functional behavior, even if the syntax of the code is different. Type-3 and Type-4 clones are further categorized into four sub-categories based on their syntactic similarity: Very Strongly Type 3 (VST3) with a similarity in range of 90% (inclusive) to 100%, Strongly Type 3 (ST3): 70–90%, Moderately Type 3 (MT3): 50–70%, and Weakly Type 3/Type 4 (WT3/4):0-50% in BigCloneBench [27], [28]. Developers often need to search for these code clones to improve their code [2] for addressing software engineering challenges such as software diversification [29], software repair [30], and even automated software transplantation [31]. According to Kapser and Godfrey [20], code clones are helpful in exploratory development, where one wishes to rapidly develop a new feature using a clone-and-own approach, and not necessarily unify (refactor, parameterize, etc.) the existing clones. Hence, such a cloning approach allows flexibility and increased productivity in an opportunistic programming scenario.

However, searching for code clones (or code fragments in general) is challenging, because there is only a small chance (10-15%) that developers would guess the exact words used in the code [32] and use them in their query. Similarly, source code is structured non-linearly; this makes it difficult to be read in a linear fashion like normal text, and be searched effectively. There are several other factors which depends on the effectiveness of code search such as quantity of data, quality of the indices [2], [33], search technique, query, and metadata (also known as *natural language document*). Often it is difficult for a developer to formulate an accurate query to express what is really in her/his mind, especially when the maintainer and the original developer are not the same person. When a query performs poorly, it has to be reformulated. But the words used in a query may be different from those that have similar semantics in the source code, i.e., the synonyms, which will affect the accuracy of code search results. For example, in Java, the programming concept "array" does not match with its syntactic representation of "[ ]." Code search engines can more effectively assist developers over the search query, if such semantic mappings exist. Similarly, there are two types of code search engines. One is known as code-to-code search engine, which accepts code fragments from users, and recommends syntactically or semantically similar code fragments found in a target code base [2]. The other one is known as natural-language-to-code search engine, which accepts natural language terms and recommends code fragments from a target code base that closely match those terms. Implementing any of these search engines can be challenging, and recently a number of techniques have been proposed to address the weaknesses of existing code search techniques [34]–[36].

We found that the accuracy of existing code search tools are often unsatisfactory in retrieving Type-4 (semantic) clones, with the recall for Type-4 clones reported by the state-of-the-art clone detectors and clone search approaches is as low as 17% [37]. In this work, we tackle the problem of effective code-to-code search, particularly for Type-4 clone methods. We evaluate the effectiveness of our technique by using natural language queries as well. We propose a number of different approaches to represent metadata for each clone method to enable accurate and efficient retrieval. Our experiments show that the accurate representation of clone methods in terms of metadata increases the effectiveness of code search. Specifically, we have made the following contribution.

1) We present a novel approach called Clone-Seeker, which builds a natural language document of each clone method by combining important keywords of each clone method with keywords extracted from a general description of the clone class, annotated either manually or automatically to assist developers in performing code-to-code search and natural language query search.
2) We have performed extensive empirical evaluation of our approach to assess the accuracy on two publicly available datasets, BigCloneBench [27], [28] and Project CodeNet [38].

II. RELATED WORK

In this section, we present related work covering different search techniques applied in the field of software engineering. Moreover, we also discuss different type of available clone datasets.

A. CODE SEARCH TECHNIQUES

To the best of our knowledge, no previous technique has explored the effect of utilizing clone class features in

[1]https://www.jfree.org/jfreechart/

searching for clone methods. However, there are several approaches in the literature focusing on different aspects of effective code search such as query refinement, quality of indices and search technique. We discuss these approaches in this section.

Several techniques for code search have been devised, which focus on query refinement and query expansion. For example, Hill *et al.* [34] reformulate queries with natural language phrasal representations of method signatures. Haiduc *et al.* [36] proposed to reformulate queries based on machine learning. They trained a machine learning model that automatically recommends a reformulation strategy based on the query properties. Lu *et al.* [39] proposed to extend a query with synonyms generated from WordNet. However, Sridhara *et al.* [40] observed that automatically expanding a query with inappropriate synonyms may produce even worse results than not expanding the query.

There is also a substantial volume of work that takes into account code characteristics for code search. For example, Mcmillan *et al.* [41] proposed Portfolio, a code search engine that combines keyword matching with PageRank to return a chain of functions. Lv *et al.* [3] proposed CodeHow, a code search tool that incorporates an extended Boolean model and API matching. Ponzanelli *et al.* [42] introduced an approach that automatically retrieves pertinent discussions from Stack Overflow, given a context in the IDE. Li *et al.* [43] presented RACS, a code search framework for JavaScript that considers relationships (e.g., sequencing, condition, and callback relationships) among the invoked API methods.

Our approach is quite similar to the idea of Software Bertillonage [44], which is a signature based matching technique. It focuses on the reduction of search space, when trying to locate a software entity within bytecode. Our technique works in a similar way, as it also reduces the search space by representing each clone method with metadata. However, our technique helps developers to perform code clone search based on a search query written either as source code terms, or as natural language instead of locating an entity within bytecode. In other related work, researchers proposed representing code snippet in the form of metadata or natural language document or signature and applied different search techniques such as information retrieval, neural networks, and joint models. For example, Sachdev *et al.* [45] investigated the use of natural language processing and information retrieval techniques to carry out natural language search directly over source code. They represented code snippet in the form of natural language document to make the retrieval of code effective. Kim *et al.* [2] proposed FACOY, which is an effective code-to-code search engine for finding semantically similar code fragments in large code bases. FACOY took a code snippet as the input query and retrieves semantically similar code snippets from the corpus. They conducted their study on a comprehensive dataset collected from GitHub and StackOverflow websites. They replaced GitHub-based code index with IJaDataset to make FaCoY search only code fragments in the specified dataset. Bajracharya *et al.* [46] presented a framework, named as Sourcerer, for performing code search over open–source projects available on the Internet. Sourcerer worked by extracting keywords and fine–grained structural features from source code, and searching for similar code using the text search engine Apache Lucene.[2] Ragkhitwetsagul and Krinke [37] incorporated a multi-representation technique named as Siamese, corresponding to four clone types, to represent an indexed corpus of code. They improved the query quality by leveraging the knowledge of token frequency in the codebase, and finally re-rank the searched candidate code based on the TF-IDF weighting method. Ahou *et al.* [47] proposed Lancer, a context-aware code to-code recommending tool. Lancer used a Library-Sensitive Language Model and a BERT model to recommend relevant code samples in real-time based on the incomplete code. Various other search methodologies [1], [48], [49] existed, which jointly embedded code snippets and natural language descriptions into a high-dimensional vector space, in such a way that code snippet and its corresponding description had a similar vector representation.

B. CLONE DATASETS

There are several popular clone related datasets, which can be used for clone search. These datasets include BigCloneBench (BCB) [27], [28], Project CodeNet (PCN) [38], SeSaMe [50], Pedagogical programming Open Judge (POJ-104) [51] and Google Code Jam(GCJ) [52]. BigCloneBench dataset contains references of clone methods belonging to different functionality types that exist in IJaDataset. SeSaMa contains clone method pairs, which are mined from 11 open source repositories. PCN, POJ-104 and GCJ are mined through online judge websites. These datasets consist of several problems, where each problem has multiple solutions submitted by students. Each solution is contained in a single file, and all files belonging to the same problem are syntactically or semantically equivalent to each other as they are attempting to solve the same problem. These datasets do not contain non-clone parts, because the whole file is considered to be a clone of the other files. However, in BigCloneBench and SeSama, clones are at the method level only.

III. OUR METHODOLOGY FOR CODE CLONE SEARCH

In this section, we outline our methodology for Clone-Seeker,[3] which focuses on utilizing clone class features in retrieving code clones, given a search query. The effectiveness of code clone search relies on multiple factors such as search technique, quality of the search query, and its relationship to the text contained in the software artifacts (metadata, i.e. natural language document). We believe that utilizing clone class features should also be considered as an important factor, which can help in performing effective code clone search. For this purpose, we first apply pre-processing steps

[2]https://lucene.apache.org/

[3]Code and dataset is accessible from link https://www.win.tue.nl/∼mhammad/Clone-Seeker/cloneseeker.html

to extract identifiers from each clone method. Afterwards, we present two ways to annotate clone classes with keywords: manual and automatic. Finally, we augment the annotated keywords of clone classes with clone method identifiers to build a natural language document of each clone method. Figure 1 displays a pictorial representation of our methodology. The top portion displays how our search corpus is built from a dataset, whereas the bottom portion displays how the search results are retrieved from Clone-Seeker. We elaborate the details of our approach in the following sections.

A. DATASET SELECTION

We build our search corpus from IJaDataset clone methods referenced by BigCloneBench [27], [28]. BigCloneBench is the largest clone benchmark dataset, with over 8 million manually validated clone method pairs in IJaDataset 2.0.[4] IJaDataset, in turn, is a large Java repository with 2.3 million source files (365 MLOC) from 25,000 open-source projects. BigCloneBench contains references to both syntactic and semantic method clones, and keeps the references of starting and ending lines for those clones. In forming this benchmark, the authors used pattern-based heuristics for identifying the methods that potentially implement a given common functionality. These methods were manually annotated as true or false positives of the given functionality by the judges. All true positives of a functionality were grouped as a clone class, such that a clone class of size n contains $\frac{n(n-1)}{2}$ clone pairs. Currently, BigCloneBench contains clones corresponding to 43 distinct functionalities (i.e. clone classes).

B. IDENTIFIER EXTRACTION

Identifiers are an important source of domain information and can often serve as a starting point in many program comprehension tasks [39], [53]. We follow similar pre-processing steps to extract identifiers from each clone method as defined by Lu *et al.* [39] (see Step A1 in Figure 1). First, we extract starting and ending line references of a total of 14,922 true positive clone methods (*Extraction*). Next, we trace them in the IJaDataset files, by following their references from the BigCloneBench dataset, and put them in our search corpus list (*Tracing*). Afterwards, we normalize each clone method code by removing the Java reserved keywords, constant values, whitespaces, extra lines, comments, as well as perform tokenization (*Normalization*). We use the Javalang[5] Python library for tokenization, which contains a lexer and parser for the Java 8 programming language. We do not use comments as they have been reported to be non-reliable and inconsistent source for extracting natural language document [54], [55]. Similarly, software projects can be poorly documented. We are unable to know the extent to which each comment accurately describes its associated clone method. For example, a number of comments can be outdated with regard to the code that they describe. Moreover, we found some comments of clone methods in IJaDataset written in other languages, whereas our methodology focuses only on English queries. After tokenization, we perform snake case and camel case splitting of identifiers to separate words (*Splitting*). We also eliminate single characters (*Single Character Elimination*) and apply stemming (*Stemming*). Stemming helps in mapping the related words to the same stem or root word, which can help in retrieving semantic code clones effectively. This finally produces a flat list of keywords representing each clone method.

C. ANNOTATING CODE CLONES

Data annotation is a process of labeling the data available in various formats like text, video or images [56]. It is expected to play a major role in enhancing product recommendations, relevant search engine results, computer vision, speech recognition, chatbots, and more. We apply annotation techniques to best describe the core functionality of each clone class in the BigCloneBench dataset. The purpose of annotating clone classes is to assist in retrieving clone methods, when a code-to-code search or a natural language query is applied. Annotation can be performed manually or automatically (Step A2 in Figure 1), the details of which are described as follows.

1) METHOD 1: MANUAL ANNOTATION

Manual annotation is a process of labeling or annotating any data by humans. The approach is popular owing to its benefits such as accuracy, high level of integrity, need for minimal administration of data annotation efforts, and a higher chance of discovering intriguing insights pertaining to the data as compared to automatic annotation techniques, which can be later integrated into an algorithm. However, manual annotation is more expensive and time-consuming. Nowadays, there are many key players offering exclusive products and services in the market,[6] to manually annotate a huge bulk of data.

BigCloneBench dataset is a manually created and validated dataset, which also contains annotations, i.e. the description of each clone class in natural language terms. To use in this paper, we manually changed the description slightly by removing bullet points, correcting spelling mistakes, and making the sentence structure more readable (Step A3b in Figure 1). Table 1 displays the resulting list of clone class descriptions used in our experiments. Afterwards, we performed several pre-processing steps such as removing spaces, single characters, and stop-words, and performed stemming in order to get a flat list of words (*Word Extraction*) per clone class.

2) METHOD 2: AUTOMATIC ANNOTATION

The second approach that we adopt to annotate clone classes involves automatically extracting keywords. Keywords are a subset of words or phrases from a document that can describe concepts or topics covered in the document [57], [58]. They are commonly used to annotate articles or other documents,

[4]https://sites.google.com/site/asegsecold/projects/seclone
[5]https://github.com/c2nes/javalang

[6]https://aimultiple.com/data-annotation-service

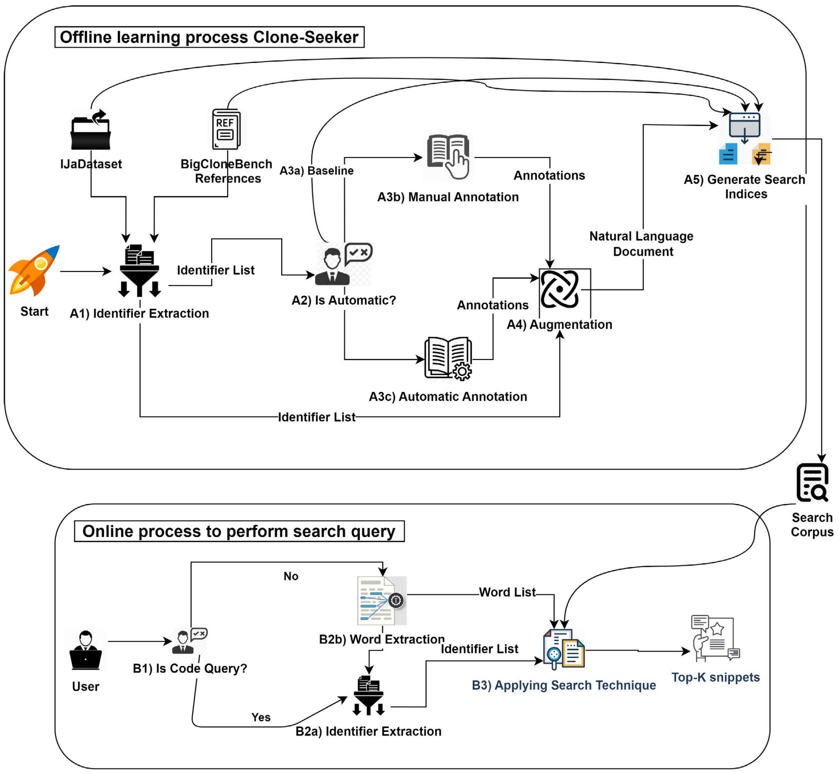

FIGURE 1. Clone-seeker methodology and applying search query.

and are essential for the categorization and fast retrieval of such items in digital libraries [59].

Automatic keyword extraction (also known as keyword detection or keyword analysis) is the process of selecting words and phrases from a text document that can best describe the core sentiment of the document without any human intervention depending on the model [57], [58]. It is a text analysis technique that automatically extracts the most used and most important (with respect to certain criteria, to be elaborated later in this section) words and expressions from text. It helps to summarize the content of texts, recognize the main topics discussed, and automatically create a compressed version of a text that provides useful information for the users.

Keyword extraction simplifies the task of finding relevant words and phrases within unstructured text. There are various methodologies to extract keywords such as word frequency, word degree, TF-IDF and RAKE [60]. We follow a simple approach known as word frequency or naive counting [61], in which we identify the list of words that repeat the most within a set of natural language documents of clone methods. Various researchers [62]–[66] in the past have adopted this methodology for tagging. It is considered to be the most effective starting point for understanding a text [67]. We have adopted this method as a proof of concept for our approach (Step A3c in Figure 1). It can be useful for identifying recurrent terms in a set of clone methods belonging to a clone class, and eventually finding out the most common words in the whole clone class. More advanced approaches are to be investigated in future work.

D. AUGMENTATION OF IDENTIFIERS WITH ANNOTATED KEYWORDS

We augment (Step A4 in Figure 1) the list of words from the identifier extraction process (Section III-B) with the ones extracted from the clone class annotations (Section III-C) and build a natural language document (i.e. metadata) mapped to each clone method (Step A5 in Figure 1). We hypothesize that the augmentation will help in retrieving clone methods effectively when a search query is applied. For further illustration, we explain how the natural language document of a "Copy File" clone method has been built in Table 2, by merging the identifier keywords with the annotation words. Annotated words are highlighted with blue color.

E. FORMULATING A SEARCH QUERY

There are two ways to formulate a search query; one is to enter some code fragment as a query, and the other is to write keywords in natural language to find relevant code fragments. Our approach can help in retrieving clone methods in both these forms. For the first case, i.e. code-to-code search, we perform the pre-processing steps on the search query in the form of a code method, as mentioned in *Identifier Extraction* (see Section III-B and Step B2a in Figure 1). In the second case, we perform the pre-processing steps for *Word*

TABLE 1. Manual annotation of clone classes for BigCloneBench dataset.

Id	Description
2	Download file from http link by using Web
3	Generate secure hash in binary format and convert it into a string representation
4	Copy file from source to destination
5	Retrieve a zip archive (from disk, internet, etc). Open it, and iterate through its entries. Decompress the files to disk, by preserving their directory structure.
6	Connect to the FTP server by log in with user name and password
7	Sorts an array of values using bubble sort
8	Setup SGV by creating ScrollingGraphicalViewer object and inject model into it
9	Setup SGV event handler by creating ScrollingGraphicalViewer object and add listener
10	Execute a database update and rollback perhaps conditionally
11	Initialize Java eclipse project by creating it in a workspace. Then, set the project's nature to Java (may also set other natures, like plugin). Afterwards, set its class path. Finally, set its output path as bin directory or folder.
12	Takes an integer and returns its prime factors
13	Shufflle array by placing elements randomly such as by using Fisher-Yates algorithm
14	Finds the position of a specified input value (search key) within an array by key value using the binary search algorithm. True positive may use any linear data-structure (array, list, etc). Must use the binary search algorithm. Must return the index or an error code.
15	Load custom font by using URL or file etc. Then, create font and register it with graphics environment.
17	Create encryption key files by first generating a pair of encryption keys such as public and private. Then, these keys are written to file or files in some format.
18	Takes a sound source and plays it
19	Take screenshot. Save screenshot to file.
20	Calculates either the ith fibonacci sequence, or the fibonacci sequence up to (or including) the ith term.
21	Build a message such as MessageBuilder and send it over XMPP service.
22	Takes some data, encrypts it, and writes it to a file
23	Resizes (shrink or expand) an array
24	Open URL in system default web browser.
25	Open file in desktop application by firstly check if desktop AWT is supported on the platform (isDesktopSupported). Then, open the file by using default desktop application.
26	Find Greatest Common Denominator (GCD)
27	Uses reflection to access a method and calls or invokes that method
28	Parse XML to DOM by first creating or retrieving DocumentBuilderFactory. Then, configure that factory and use it to create new document builder. Finally, use that document builder to parse the XML data and return or use the DOM.
29	Convert date string format by parsing source date string and format it by using a destination format.
30	Create zip archieve containing one or more files
31	File selection dialog. Use a file chooser to select one or more files or directories.
32	Send email using Java mail. Receives a session (either creates session in snippet, or its created elsewhere). Creates the mime message and configures it with at least some standard data. Sends the email.
33	Receives some file. Produces a CRC32 checksum of the file.
34	Execute external process and do something with the output (stdout or sterr or both).
35	Instantiate using reflection. Get class object. Use reflection to find the constructor. Use the constructor to instantiate the object.
36	Connect to database. Create a database connection.
37	Load file into byte array. The entire file is loaded into a single byte array.
38	Get the MAC address of a network device. Convert byte MAC address into standard format.
39	Delete a folder, and all the files and folders it contains, recursively down the file heirarchy.
40	Parse the elements of a csv file. Where each line is an entry, and the elements of an entry are separated by some delimiter on that line.
41	Transpose a matrix (2d array).
42	Use a regular expression to extract matches of a pattern in text.
43	Copy directory and its contents.
44	Test if a string is a palindrome.
45	Writes a pdf file.

Extraction, in order to get a flat list of words (Step B2b in Figure 1).

F. SEARCH METHODOLOGY

Information Retrieval (IR) techniques have been successfully applied to address various software engineering tasks, including concept/feature/concern location, impact analysis, code retrieval and reuse, bug triage, refactoring and restructuring, reverse engineering, and defect prediction [68]. IR techniques, in general, are used to discover the significant documents in a large collection of documents, which match the user's query. They mainly aim for identifying the information relevant to the user requirements in a given scenario. An IR-based code retrieval method in particular usually extracts a set of keywords from a query and then searches for the keywords in code repositories [69].

We apply an IR technique (Step B3 in Figure 1) to retrieve the top-10 results from the search corpus that match the natural language query. The selected IR technique is based on Term Frequency-Inverse Document Frequency (TF-IDF) word embeddings for retrieving the clone methods most similar to the query [70]. TF-IDF is a technique often used in IR and text mining. According to a survey in 2015, 70% of text-based recommendation systems for digital libraries use TF–IDF [71]. TF-IDF uses a weighting scheme which assigns each term in a document a weight corresponding to

TABLE 2. Building the natural language document (NLD) of the "Copy File" clone method with annotation techniques.

Clone Method	`public static void copyFile(File src, File dest) throws IOException {`
	`    FileInputStream fis = new FileInputStream(src);`
	`    FileOutputStream fos = new FileOutputStream(dest);`
	`    java.nio.channels.FileChannel channelSrc = fis.getChannel();`
	`    java.nio.channels.FileChannel channelDest = fos.getChannel();`
	`    channelSrc.transferTo(0, channelSrc.size(), channelDest);`
	`    fis.close();`
	`    fos.close();`
	`}`

Identifier Extraction	copi file src dest io except input stream fis output fos java nio channel get transfer to size close
Manual Annotation	Copy a file from source to destination
Manual Annotation (after pre-processing)	copi file sourc destin
NLD (Manual Annotation)	copi file sourc destin src dest io except input stream fis output fos java nio channel get transfer to size close
Automatic Annotation (after pre-processing)	except file close stream input
NLD (Automatic Annotation)	except file close stream input copi src dest io fis output fos java nio channel get transfer to size

its term frequency and inverse document frequency. In our context, TF-IDF looks 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. These vectors can in turn easily be compared by quickly calculating their cosine similarities. If a term appears frequently in a clone method's natural language document, that term is likely important in that method. The frequency of a term is simply the number of times that a term appears in a clone method. However, if a term appears frequently in many clone methods' natural language document, that term is likely less important generally. To factor this, we use the IDF measure. IDF 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 the pre-processed query into TF-IDF vectors using the formula in 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. For the retrieval, we generate a normalized TF-IDF sparse vector from a given query, and then take its dot product with the feature matrix. Given that all vectors are normalized, the result yields the cosine similarity between the query vector of the query and of every clone method. Afterwards, we return the list of all the clone methods, ranked by their cosine similarities to the query vector.

IV. EMPIRICAL EVALUATION

We present empirical results in this section to validate the effectiveness of our approach. We evaluate Clone-Seeker by formulating the search query in two ways: by source code and by natural language terms. For the first case, we evaluate the effectiveness of our proposed approach by retrieving code clones when a code query is applied. For the second case, we evaluate the effectiveness using natural language queries. We describe the design of different assessment scenarios for Clone-Seeker and report on the evaluation results. Our primary evaluation is based on the BigCloneBench dataset. However, we also perform experiments on the Project CodeNet dataset, to evaluate the generalizability of our approach. We perform statistical analysis of our empirical results in terms of Wilcoxon rank sum test. It is used to test that a distribution is symmetric about some hypothesized value by utilizing magnitudes of the differences [72]. We compare our experimental result of code to code search for BigCloneBench dataset with previous techniques. For other experiments, there is no related work to directly compare our results against. We performed the pre-processing steps on a Toshiba laptop with Windows 10, 2.6 GHz Intel Core i5 CPU and 16 GB RAM, and the empirical evaluation on SurfSara and TU/e HPC Cluster. It took 18 minutes and 15 seconds to complete the pre-processing steps for all the variants of Clone-Seeker for BigCloneBench dataset. Similarly, it took 10 minutes and 43 seconds to complete the pre-processing steps for all the variants of Clone-Seeker for Project CodeNet dataset. Moreover, we also measured the average time taken by each query on the same laptop, in retrieving code clones for all Clone-Seeker variants on BigCloneBench (BCB) and Project CodeNet (PCN) datasets (see Table 3). The time differed in terms of the number of tokens exist in a search corpus. The results in Table 3 shows that the query usually took more time in retrieving code clones for Clone-Seeker (Manual) strategy as compared to others. Specifically, our experiments aim to address the following research questions:

1) RQ1: What is the effectiveness of Clone-Seeker in finding clones among BigCloneBench dataset, based on code-to-code search?
2) RQ2: How relevant are code examples found by Clone-Seeker compared to other code-to-code search engines?

TABLE 3. Average time taken for retrieval for all clone-seeker variants on the basis of code-to-code and natural-language-to-code search.

Dataset	Query type	Clone-Seeker variants		
		Baseline	**Manual**	**Automatic**
BCB	Natural-language-to-code	1.32 sec	1.59 sec	1.43 sec
	Code-to-code	1.11 sec	1.17 sec	1.14 sec
PCN	Natural-language-to-code	0.81 sec	0.86 sec	0.85 sec
	Code-to-code	0.75 sec	0.91 sec	0.79 sec

3) RQ3: What is the effectiveness of Clone-Seeker in finding clones among BigCloneBench dataset, based on natural language queries to code search?
4) RQ4: Does Clone-Seeker generalize to other datasets in finding clones using code query?
5) RQ5: Does Clone-Seeker generalize to other datasets in finding clones using natural language query?

A. RQ1: FINDING CODE CLONES USING CODE-TO-CODE SEARCH IN BigCloneBench DATASET

1) EXPERIMENT DESIGN

In this section, we evaluate the performance of Clone-Seeker against BigCloneBench dataset in a scenario for searching by code. Given that BigCloneBench consists of clone classes each having a set of clone methods (i.e. references to Java methods in IJaDataSet), we process each individual clone method to be used in a code-to-code query. For each clone method c_{input}, we first extract the identifiers from the source code following the steps mentioned in Section III-B. Using this flat list of identifiers as the query to Clone-Seeker, we search for the most similar natural language documents as developed in Section III-D. These natural language documents links to resulting code methods ($c_{res}^{k..l}$) in IJaDataset. Since the most similar code methods are already labelled and referenced in BigCloneBench ($c_{ref}^{m..n}$) in the corresponding clone class, we check whether the pairs (c_{input}, c_{res}^{i}) coincide with any (c_{input}, c_{ref}^{j}). Afterwards, we investigate the top-10 retrieved results and calculate the precision, which is the number of ground-truth answers hit on average in the Top@k returned for a query using a Precision@k metric defined as follows:

$$Precision@k = \frac{1}{|Q|}\sum_{|Q|}^{i=1}\frac{|relevant_{i,k}|}{k} \tag{2}$$

where $relevant_{i,k}$ represents the relevant search results for query i in the top k returned results, and Q is a set of queries. Precision shows the relevance of the returned results to the queries with respect to the ground-truth answers; the higher the value, the more relevant the results are. Moreover, we measure the Mean Reciprocal Rank (MRR) scores [2], [73], which is the average of the reciprocal ranks of results of a set of queries Q. The reciprocal rank of a query is the inverse of the rank of the first hit result. It is used, when user is interested in considering only the first hit. MRR is calculated by using the following formula:

$$MRR = \frac{1}{|Q|}\sum_{|Q|}^{i=1}\frac{1}{rank_i} \tag{3}$$

where $rank_i$ refers to the rank position of the first relevant result for the i-th test input method. The higher the MRR value is, the better the code search performs. Table 4 shows the evaluation results in terms of MRR and precision of different approaches for each query in the benchmark.

We evaluate three different approaches as the annotation strategy: (1) Manual Annotation and (2) Automatic Annotation, as described in Section III-C, as well as (3) Baseline. In Baseline, we do not provide any annotation (Step A3a in Figure 1), and just consider a flat list of pre-processed keywords as described in Section III-B. We aim for assessing the impact of adding annotations on the accuracy of code-to-code search. In order to annotate each clone class automatically with a set of words, we retrieve the top-k most recurrent words list by applying different threshold values for k such as 5, 10, 15, and 20. Then, we calculate MRR and Precision after performing the augmentation step (see Section III-D). We notice that with threshold $t = 5$, precision performs best among other values. Therefore we choose the top-5 most recurrent keywords for our remaining experiments related to BigCloneBench dataset.

2) RESULTS

The overall approach we propose leads to reasonable results in retrieving clone methods against specified pre-processed clone code queries. Table 4 presents the results in terms of MRR, Precision@1, Precision@3, Precision@5, and Precision@10. We notice that by adding more meta tokens as part of manual or automatic annotation in the natural language document has incurred minor differences as compared to baseline on the performance of code-to-code search. We have performed Wilcoxon rank sum test to statistically compare **MRR** and **Precision** values of different variants of Clone-Seeker. The larger the p-value from the significance level ($\alpha = 0.05$), indicates the higher the chance is to accept the null-hypothesis (H_0) and reject the alternate-hypothesis (H_1), and vice-versa.

We compare Clone-Seeker (Automatic) with Clone-Seeker (Manual), which indicates that Clone-Seeker (Manual) is not significantly different than Clone-Seeker (Automatic) (H_0:$C_{Manual} = C_{Automatic}$, H_1:$C_{Manual} \neq C_{Automatic}$, $p = 0.18$). Similarly, when the same test was performed to compare Clone-Seeker (Baseline) with Clone-Seeker (Manual), the difference was again found to be not significantly different (H_0:$C_{Baseline} = C_{Manual}$, H_1:$C_{Baseline} \neq C_{Manual}$, $p = 0.460$). We further notice that Clone-Seeker (Automatic) does not achieve significantly better scores as compared to Clone-Seeker (Baseline) (H_0:$C_{Automatic} = C_{Baseline}$, H_1:$C_{Automatic} \neq C_{Baseline}$, $p = 0.06$). This points out that there is still room to explore different annotation techniques, which can significantly outperform baseline.

TABLE 4. MRR and precision of finding code clones using code-to-code search in BigCloneBench dataset.

	MRR	P@1	P@3	P@5	P@10
Baseline	0.986	0.984	0.940	0.920	0.895
Manual	0.991	0.984	0.935	0.915	0.888
Automatic$_{(t=5)}$	0.987	0.985	0.942	0.923	0.898
Automatic$_{(t=10)}$	0.991	0.985	0.941	0.922	0.897
Automatic$_{(t=15)}$	0.991	0.984	0.939	0.918	0.893
Automatic$_{(t=20)}$	0.991	0.984	0.938	0.917	0.893

More annotation strategies need to be explored, which can considerably perform better than Clone-Seeker (Baseline) in terms of code query is applied.

B. RQ2: COMPARISON WITH CODE SEARCH ENGINES

In this section, we demonstrate the effectiveness of Clone-Seeker by comparing its performance with other code search engines on the basis of recall.

1) EXPERIMENT DESIGN

We evaluate Clone-Seeker against BigCloneBench dataset in a scenario for searching by code. We compare recall of our search approach with state-of-the-art clone detector and search approaches presented by [2], [37].

In BigCloneBench, clone pairs are assigned a type based on the criteria in [74]. Type-1 and Type-2 clone pairs are classified according to the classical definitions given in [2], [37]. Moreover, Type-3 and Type-4 clones are divided into four sub-categories according to their syntactical similarity: Very Strongly Type 3 (VST3), Strongly Type 3 (ST3), Moderately Type 3 (MT3), and Weakly Type 3/Type 4 (WT3/4). Each clone pair (unless it is Type 1 or 2) is identified as one of the four types if its similarity score falls into a specific range; VST3: [90%, 100%), ST3: [70%, 90%), MT3: [50%, 70%), and WT3/4: [0%, 50%). We compute the recall of different variants of Clone-Seeker following the definition proposed in the original benchmark of BigCloneBench [27].

$$Recall = \frac{D \cap B_{tc}}{B_{tc}} \tag{4}$$

where B_{tc} is the set of all true clone pairs in BigCloneBench, and D is the set of clone pairs found by Clone-Seeker. We compare the performance of Clone-Seeker with the other approaches reported in [2], [37]. For fairness in comparison, we choose the same configuration of retrieving the top-900 natural language documents, which are mapped to their associated clone methods. Similarly, we choose clone methods having at least 6 lines and 50 tokens in length, a standard measure to consider clones for benchmarking [2].

2) RESULTS

Table 5 depicts the recall scores for our approach with different annotation strategies: Clone-Seeker (Baseline), Clone-Seeker (Manual), and Clone-Seeker (Automatic). Recall scores are summarized per clone type with the categories introduced above. Since for Clone-Seeker we are reproducing the experiments performed in [2], [37], [74], we directly report in the same table all the results that the authors have obtained on the benchmark for their own approaches such as Siamese, FaCoY and state-of-the-art code clone detectors including NiCaD [75], iClones [76], SourcererCC [74], CCFinderX [77], and Deckard [78].

Our goal, as outlined in Section I, is to build Clone-Seeker as a code-to-code search engine capable of finding Type-4 clones. Nevertheless, for a comprehensive evaluation of the added value of the strategies implemented in our approach, we provide the comparative results of recall values across all clone types. Overall, we notice that the recall performance of Clone-Seeker (Automatic) is better than Clone-Seeker (Baseline) and Clone-Seeker (Manual) across all clone types. The three Clone-Seeker variants produce the highest recall values for Type-4 clones compared to the other related approaches. This depicts that utilizing TF-IDF as search technique and building metadata with the help of identifiers works well in retrieving relevant clone methods. We do not observe significant difference in recall performance between variants of Clone-Seeker with annotation and Clone-Seeker (Baseline). This means that there might still be room for improvement in terms of annotation strategies, to be investigated in the future. With 59% recall for semantic clones (WT3/T4), Clone-Seeker (Manual) achieves the best performance score in the literature.

Clone-Seeker variants outperform the state-of-the-art clone detector and search approaches in terms of retrieving semantic clones (Type-4).

C. RQ3: FINDING CODE CLONES USING NATURAL LANGUAGE QUERY IN BigCloneBench DATASET

In this section, we demonstrate the effectiveness of Clone-Seeker by applying natural language queries (natural-language-to-code search). These queries have been collected from Stack-Overflow.[7] Stack Overflow is a popular online programming community, where programmers ask questions about programming problems and give answers. The website has been found to be useful for software development [55], [79] and also valuable for educational purposes [80]. On Stack Overflow, each conversation contains a question and a list of answers. The answer frequently contains at least one code snippet as a solution to the question asked. Sometimes, the question itself also contains a code snippet. This usually indicates that a developer asks for either a more optimized solution than the one he posts or wishes to discuss some problem in the code. We demonstrate how clone methods are retrieved based on a natural language query with the help of an example (Table 6). Suppose that a developer is interested in searching for clone methods implementing the "Copy file from source to destination" functionality. The natural language query is first pre-processed and words are

[7]https://stackoverflow.com/

TABLE 5. Recall scores(%) for clone-seeker and other related approaches on BigCloneBench.

	Clone Types					
Approaches	**T1**	**T2**	**VST3**	**ST3**	**MT3**	**WT3/T4**
Clone search engines						
Clone-Seeker(Baseline)	100	100	99	86	65	57
Clone-Seeker(Manual)	100	100	99	87	67	59
Clone-Seeker(Automatic)	100	100	99	87	66	57
Siamese [37]	99	99	99	99	88	17
FaCoY [2]	65	90	67	69	41	10
Clone detectors [74]						
SourcererCC	100	98	93	61	5	0
CCFinderX	100	93	62	15	1	0
Deckard	60	58	62	31	12	1
iClone	100	82	82	24	0	0
NiCad	100	100	100	95	1	0

extracted as explained in Section III-C1. Then, this flat list of words is fed into Clone-Seeker (Manual), which generates top-3 most similar natural language documents on the basis of TF-IDF scores, which are mentioned in Table 6. Manual annotation is highlighted with blue color in the natural language documents. Nevertheless, these documents are mapped to their associated clone methods, which are finally presented to the user. We can see in Table 6 that top- 1^{st} retrieved clone method not contain any terms related to "copy file" in the source code. But, because of manual annotation, it helps in retrieving this clone method, when natural language query is applied.

1) EXPERIMENT DESIGN

The main aim of this experiment is to assess whether our methodology can help in retrieving clone methods using a natural language query. In this experiment, we compare the search results obtained from the three annotation strategies in Clone-Seeker: Baseline, Manual Annotation, and Automatic Annotation, as described in Section III-C.

We build a benchmark of 43 queries belonging to the functionality types (clone classes) in BigCloneBench. We follow guidelines of traditional peer-review methodology [81] to build a benchmark. It has been used as a learning process to improve the quality of computer programs for at least 30 years [82]. We type in the keywords belonging to each clone class in the Stack Overflow website and search for the relevant post queries. The queries are manually selected based on following requirements from Stack Overflow: (1) the associated post is related to "Java," (2) the post contains the solution of the question answered, (3) the post includes a code snippet belonging to one of the mentioned functionality types in BigCloneBench, and (4) first author identifies the posts and the second author verifies them. The results are finally reported once a consensus is reached. The full list of the 43 selected queries can be found in Table 7. We do not claim that our list of queries is comprehensive, but at least it should be a good starting point. We invite other researchers to extend this list.

To evaluate search effectiveness, we feed Clone-Seeker with benchmark queries belonging to different clone classes after applying pre-processing steps as mentioned in Section III-C (*Word Extraction*). Then, we inspect the top-10 results retrieved from search corpus, as built from BigCloneBench references and IJaDataset in Section III-D by identifying their clone classes. Afterwards, we calculate MRR and precision of different approaches for each query in the benchmark (Table 8).

2) RESULTS

The overall approach we propose leads to promising results in retrieving clone methods against specified natural language queries. Table 8 presents a comparison between Clone-Seeker (Baseline), Clone-Seeker (Manual), and Clone-Seeker (Automatic) in terms of average MRR, Precision@1, Precision@5, and Precision@10. Overall, we notice that Clone-Seeker (Manual) achieves better MRR and precision as compared to Clone-Seeker (Automatic) and Clone-Seeker (Baseline) and in both cases, it was found to be statistically outperformed as it supports the alternate-hypothesis (H_0:C_{Manual} <= $C_{Automatic}$, H_1:C_{Manual} > $C_{Automatic}$, p <.00001) and (H_0:C_{Manual} <= $C_{Baseline}$, H_1:C_{Manual} > $C_{Baseline}$, p <.00001). We further notice that Clone-Seeker (Automatic) does not achieve significantly better scores as compared to Clone-Seeker (Baseline) (H_0:$C_{Automatic}$ <= $C_{Baseline}$, H_1:$C_{Automatic}$ > $C_{Baseline}$, p =.42074). This requires further investigation to come up with a better annotation strategy, which can outperform baseline.

Clone-Seeker (Manual) outperforms other variants in general, as manually annotated terms quite resembles with natural language queries.

D. RQ4: GENERALIZABILITY OF CLONE-SEEKER IN FINDING CODE CLONES USING CODE AS A QUERY

1) EXPERIMENT DESIGN

In previous sections, we have introduced the fundamental techniques and evaluated them with respect to multiple

TABLE 6. Example of retrieving relevant clone methods using natural language query.

Search query	Error while copying files from source to destination java
Search query (Word Extraction)	error copi file sourc destin java
Relevant Results (Natural Language Documents)	
Top 1	copi file sourc destin java to html io except reader writer util write flush close **Score:**0.464
Top 2	copi file sourc destin string dest java io except buffer input stream in output out avail write read close **Score:**0.459
Top 3	copi file sourc destin java io byte array output stream get url buffer input open util close quiet except err runtim **Score:**0.424
Clone Methods	

Top 1

```java
private void javaToHtml(File source, File destination) throws IOException {
    Reader reader = new FileReader(source);
    Writer writer = new FileWriter(destination);
    JavaUtils.writeJava(reader, writer);
    writer.flush();
    writer.close();
}
```

Top 2

```java
public static void copy(String source, String dest) throws java.io.IOException {
    java.io.BufferedInputStream in = null;
    java.io.BufferedOutputStream out = null;
    try {
        in = new java.io.BufferedInputStream(new java.io.FileInputStream(source), 1000);
        out = new java.io.BufferedOutputStream(new java.io.FileOutputStream(dest), 1000);
        while (in.available() != 0) {
            out.write(in.read());
        }
    } catch (java.io.IOException e) {
        throw e;
    } finally {
        try {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        } catch (java.io.IOException E) {
        }
    }
}
```

Top 3

```java
public static java.io.ByteArrayOutputStream getFileByteStream(URL _url) {
    java.io.ByteArrayOutputStream buffer = new java.io.ByteArrayOutputStream();
    try {
        InputStream input = _url.openStream();
        IOUtils.copy(input, buffer);
        IOUtils.closeQuietly(input);
    } catch (Exception err) {
        throw new RuntimeException(err);
    }
    return buffer;
}
```

aspects by focusing on BigCloneBench dataset. This gives a strong foundation on the feasibility of using our approach.

However, it is possible to conduct our study on other clone related datasets. In order to assess the generalizability of our methodology on another dataset, we choose Project CodeNet dataset. The Project CodeNet dataset consists of a very large collection of metadata, source files and documentation. It is a recently introduced benchmark, which can be used to perform several software engineering tasks such as summarization, completion, code search, and as well as code-to-code translation. It is mined from online judge websites such as AIZU Online Judge[8] and AtCoder.[9] These websites offer programmers an opportunity to test their skills by posing programming problems in the form of courses or contests. Users submit their solution, which is then judged by an automatic review mechanism. The outcome is reported back to the user.

It has several benchmarks available in different languages such as Java, C++, Python etc. We select Java programming benchmark, which contains 250 set of problems. These problems are either representing a puzzle or a generic problem. Each problem contains 300 sample solution files, which are syntactically or semantically similar to each other. Afterwards, we manually analyze the nature of each problem and filter down those problems that are generic in nature by ensuring that their solutions in Java programming language are well discussed by developers over StackOverflow[10] and GeeksForGeeks[11] websites. We choose GeeksForGeeks,

[8]https://onlinejudge.u-aizu.ac.jp/home
[9]https://atcoder.jp/
[10]https://stackoverflow.com/
[11]https://www.geeksforgeeks.org/

TABLE 7. Benchmark queries to evaluate clone-seeker variants on BigCloneBench dataset.

Id	Description
2	How to download and save a file from the Internet using Java?
3	How can I generate an MD5 hash?
4	Error while copying files from source to destination java
5	Java decompress archive file
6	Java: Accessing a File from an FTP Server
7	Basic Bubble Sort with ArrayList in Java
8	GEF editor functionality to view
9	ScrollingGraphicalViewer Select and Unselect listener
10	Java: Rollback Database updates?
11	Creating a Eclipse Java Project from another project, programatically
12	Java Display the Prime Factorization of a number
13	Random shuffling of an array
14	First occurrence in a binary search
15	Java - How to Load a Custom Font From a Resources Folder
17	Issues in RSA encryption in Java class
18	How can I play sound in Java?
19	Is there a way to take a screenshot using Java and save it to some sort of image?
20	printing the results of a fibonacci series
21	implementing GAE XMPP service as an external component to an existing XMPP server (e.g. ejabberd or OpenFire)
22	How to Encrypt/Decrypt text in a file in Java
23	Resize an Array while keeping current elements in Java?
24	How to open the default webbrowser using java
25	Open a file using Desktop(java.awt)
26	Java: get greatest common divisor
27	How to invoke a method in java using reflection
28	Parsing XML file with DOM (Java)
29	Change date format in a Java string
30	How to create a zip file in Java
31	Is it possible to select multiple directories at once with JFileChooser
32	How can I send an email by Java application using GMail, Yahoo, or Hotmail?
33	File containing its own checksum
34	Launching external process from Java : stdout and stderr
35	With Java reflection how to instantiate a new object, then call a method on it?
36	Create MySQL database from Java
37	Reading a binary input stream into a single byte array in Java
38	Get MAC Address of System in Java
39	How to delete a folder with files using Java
40	Fastest way to read a CSV file java
41	Transposing a matrix from a 2D array
42	Using Regular Expressions to Extract a Value in Java
43	How to copy an entire content from a directory to another in Java?
44	Check string for palindrome
45	java pdf file write

as for some problems, we find better aligned solutions compared to StackOverflow. This filtering helps us in building a benchmark of natural language queries (Table 9) to evaluate our approach later on in (Section IV-E). Finally, we get a list of 46 generic problems (Table 11).

We generate three different types of search corpus by applying pre-processing steps mentioned in Section III, namely Clone-Seeker (Manual), Clone-Seeker (Automatic) and Clone-Seeker (Baseline). To build a search corpus for manual annotation, we use the clone class descriptions mentioned in Table 9. Similarly, we apply the same strategy mentioned in Section IV-A1, to annotate search corpus automatically. We observe that with threshold $t = 5$, overall MRR and precision works best among other values. Therefore, we select the top-5 most recurrent keywords for our rest of the experiments related to the Project CodeNet dataset.

2) RESULTS

Our proposed approach produces reasonable results in retrieving clone methods against pre-processed clone method queries. Table 10 presents the results in terms of MRR, and precision (P@k) with threshold values for k such as 1, 3, 5, and 10. We statistically compare MRR and precision of all the Clone-Seeker variants and the result was found to be not statistically different among all cases (H_0:$C_{Manual} = C_{Automatic}$, H_1:$C_{Manual} \neq C_{Automatic}, p = 0.06$), ($H_0$:$C_{Manual} = C_{Baseline}$, H_1:$C_{Manual} \neq C_{Baseline}$, $p = 0.06$), (H_0:$C_{Automatic} = C_{Baseline}$, H_1:$C_{Automatic} \neq C_{Baseline}$, $p = 0.06$). Hence, one might need to come up with better annotation techniques to outperform the baseline for code-to-code search in terms of MRR and precision.

Better annotation techniques need to be explored which can significantly outperform Clone-Seeker (Baseline) approach for code-to-code query.

E. RQ5: GENERALIZABILITY OF CLONE-SEEKER IN FINDING CODE CLONES USING NATURAL LANGUAGE QUERY

1) EXPERIMENTAL DESIGN

The main aim of this experiment is to prove the generalizability of our approach in retrieving clone methods using

TABLE 8. Evaluation results of natural language queries in terms of MRR(%) and precision (P@k) on BigCloneBench dataset.

	Clone-Seeker(Baseline)				Clone-Seeker(Manual)				Clone-Seeker(Automatic)			
Id	**MRR**	**P@1**	**P@5**	**P@10**	**MRR**	**P@1**	**P@5**	**P@10**	**MRR**	**P@1**	**P@5**	**P@10**
2	0.167	0	0	0.2	0.5	0	0.6	0.6	0.167	0	0	0.2
3	1	1	0.8	0.9	1	1	1	1	1	1	0.8	0.9
4	1	1	1	1	1	1	1	1	1	1	1	1
5	0.33	0	0.2	0.1	0.5	0	0.6	0.6	0.33	0	0.2	0.1
6	1	1	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1	1	1
8	0.25	0	0.2	0.2	0.33	0	0.2	0.2	0.25	0	0.2	0.2
9	1	1	0.4	0.2	1	1	0.4	0.3	1	1	0.4	0.2
10	1	1	1	1	1	1	1	1	1	1	1	1
11	0.5	0	0.4	0.5	0.5	0	0.8	0.9	0.5	0	0.4	0.5
12	1	1	1	1	1	1	1	1	1	1	1	1
13	1	1	1	1	1	1	1	1	1	1	1	1
14	1	1	0.8	0.9	0.5	0	0.8	0.9	0.5	0	0.8	0.9
15	1	1	0.4	0.2	1	1	1	1	1	1	0.4	0.2
17	0.1	0	0	0.1	0.167	0	0	0.2	0.1	0	0	0.1
18	1	1	1	1	1	1	1	1	1	1	1	1
19	1	1	0.8	0.8	1	1	1	1	1	1	1	1
20	1	1	0.4	0.2	1	1	1	0.9	1	1	0.4	0.2
21	1	1	0.6	0.6	1	1	0.8	0.7	1	1	0.6	0.6
22	0.111	0	0	0.2	0.142857	0	0	0.2	0.111	0	0	0.2
23	1	1	1	1	1	1	1	0.9	1	1	1	1
24	1	1	0.4	0.3	1	1	0.6	0.7	1	1	0.4	0.3
25	0.5	0	0.4	0.3	0.33	0	0.4	0.4	0.5	0	0.4	0.3
26	0.142857	0	0	0.1	1	1	1	0.5	0	0	0	0
27	0.5	0	0.8	0.6	1	1	0.8	0.9	0.5	0	0.8	0.7
28	1	1	1	0.9	1	1	1	1	1	1	1	0.9
29	0.2	0	0.2	0.4	1	1	0.4	0.3	0.2	0	0.2	0.4
30	0.5	0	0.8	0.9	0.5	0	0.8	0.9	0.5	0	0.8	0.9
31	1	1	0.2	0.4	1	1	0.6	0.6	1	1	0.2	0.5
32	1	1	0.6	0.6	1	1	1	1	1	1	0.6	0.6
33	1	1	0.6	0.7	1	1	0.8	0.7	1	1	0.6	0.7
34	0.5	0	0.8	0.6	1	1	1	0.9	0.5	0	0.8	0.6
35	1	1	0.2	0.2	1	1	0.6	0.3	1	1	0.2	0.2
36	0	0	0	0	1	1	0.4	0.5	0.1	0	0	0.1
37	1	1	0.2	0.1	1	1	1	1	1	1	0.2	0.1
38	1	1	1	0.8	1	1	1	0.8	1	1	1	0.8
39	1	1	0.8	0.7	1	1	1	1	1	1	0.8	0.7
40	0.2	0	0.2	0.2	0.25	0	0.2	0.4	0.2	0	0.2	0.2
41	1	1	1	1	1	1	1	1	1	1	1	1
42	1	1	0.8	0.8	1	1	1	1	1	1	0.8	0.8
43	0.125	0	0	0.1	0.25	0	0.4	0.7	0.111	0	0	0.2
44	1	1	1	1	1	1	1	1	1	1	1	1
45	0.5	0	0.8	0.7	0.5	0	0.8	0.9	0.5	0	0.8	0.7
Average	**0.736**	**0.628**	**0.577**	**0.570**	**0.825**	**0.721**	**0.767**	**0.765**	**0.723**	**0.605**	**0.581**	**0.581**

a natural language query. In this experiment, we perform a similar experiment as done in Section IV-C, this time on the Project CodeNet dataset. We compare the search results obtained from the three annotation strategies in Clone-Seeker (Baseline), Clone-Seeker (Manual), and Clone-Seeker (Automatic), as described in Section III-C.

We build a benchmark of 46 queries belonging to each generic problem or clone class existing in the Project CodeNet dataset. The full list of 46 selected solutions can be found in Table 11, as described in Section IV-D1. We evaluate the effectiveness of the search by feeding Clone-Seeker with benchmark queries belonging to different clone classes after applying the pre-processing steps as mentioned in Section III-C (*Word Extraction*). Then, we investigate the top-10 results retrieved from search corpus, as built from the Project CodeNet dataset in Section III-D by identifying their clone classes. Finally, we calculate the MRR and precision of different Clone-Seeker variants for each query in the benchmark (Table 12).

2) RESULTS

We present a comparison between Clone-Seeker (Baseline), Clone-Seeker (Manual), and Clone-Seeker (Automatic) in terms of average MRR, and Precision@1, Precision@5, and Precision@10 (Table 12). Overall, we notice that Clone-Seeker (Manual) outperforms Clone-Seeker (Automatic) and Clone-Seeker (Baseline) in terms of both MRR and precision and in both cases, it was found to be statistically outperformed as it supports the alternate-hypothesis (H_0:C_{Manual} <= $C_{Automatic}$, H_1:C_{Manual} > $C_{Automatic}$, p <.00001) and (H_0:C_{Manual} <= $C_{Baseline}$, H_1:C_{Manual} > $C_{Baseline}$, p <.00001). This is due to the fact that manual annotation and natural language queries are both human-written. So, by having manually annotated terms at the start of each natural language document increases the probability of retrieving similar clone methods, when natural language queries are applied. In contrast to this, Clone-Seeker (Automatic) does not perform well, because automatic annotated terms do not well resemble with the natural language queries,

TABLE 9. Manual annotation of clone classes for Project CodeNet dataset.

Id	Description
p00001	Sort data in descending order
p00003	Check whether a given length of three sides form a right triangle?
p00005	Calculate greatest common divisor (GCD) and the least common multiple (LCM) of given numbers
p00006	Reverse String Sequence
p02255	Calculate insertion sort
p02256	Find Greatest Common Divisor
p02257	Find Prime Numbers
p02259	Bubble Sort
p02260	Selection Sort
p02263	Evaluate an arithmetic expression in the Reverse Polish notation and prints the computational result
p02264	Simulate the round robin scheduling
p02268	Binary Search
p02381	Standard Deviation
p02388	Calculate power of an integer
p02389	Calculate the perimeter and area of a given rectangle.
p02390	Convert seconds to hours, minutes and seconds.
p02393	Sorting Three Numbers in ascending order
p02394	Check if circle is inside a Rectangle
p02397	Sort two numbers in ascending order
p02399	Calculate quotient and remainder
p02400	Calculate circumference and area of a circle
p02401	Reads two integers and an operator and then prints its value.
p02402	Compute maximum, minimum and sum of integer sequence
p02407	Reversing Numbers in an array
p02410	Matrix Vector Multiplication
p02414	Matrix Multiplication
p02415	Toggling Cases
p02416	Get sum of digits in an integer
p02417	Count alphabetic letters in a string
p02419	Count a number of word appears in a string
p02577	Check whether an integer is a multiple of 9.
p02627	Test if a character is uppercase or lowercase
p02657	Multiply two numbers
p02659	Extract Integer and truncate fractional part
p02705	Calculate circumference of a circle
p02711	Does a number contain a certain digit?
p02725	Traveling Salesman problem
p02730	Check whether the string is a Palindrome or not?
p02731	Find maximum possible volume of a cuboid
p02766	Find the number of digits that number has in base K.
p02778	Replace all characters in a string
p03041	Change character case at specific position
p03308	Find the maximum difference of two integers in a sequence
p03416	Find Palindromic Numbers in a given range
p03455	Determine whether the product of two positive integers is even or odd
p03493	Find number of squares in a grid

which lowers the effectiveness of retrieving similar clone methods and values are found to be very similar with Clone-Seeker (Baseline).

Overall Clone-Seeker (Manual) performs the best among other variants, because manually annotated terms increase the probability of retrieving similar clone methods, when a natural language query is applied.

TABLE 10. MRR and precision of finding code clones using code-to-code search in Project CodeNet dataset.

	MRR	P@1	P@3	P@5	P@10
Baseline	0.931	0.92	0.642	0.567	0.493
Manual	0.855	0.822	0.583	0.522	0.465
Automatic$_{(t=5)}$	0.939	0.922	0.645	0.573	0.498
Automatic$_{(t=10)}$	0.927	0.907	0.637	0.566	0.495
Automatic$_{(t=15)}$	0.895	0.87	0.627	0.561	0.498
Automatic$_{(t=20)}$	0.876	0.85	0.615	0.552	0.491

V. DISCUSSION

With various experiments, we show that our methodology can work effectively in retrieving code clones using code and natural language queries. We primarily conduct our experiments by using the BigCloneBench dataset. However, we also prove that our methodology is generalizable to a considerable extent by conducting experiments on the Project CodeNet dataset. We observe that Clone-Seeker (Manual) outperforms other variants in terms of MRR and precision based on natural language to code search. This is because both manual annotations and natural language queries are human-written, and the chances of overlapping terms are increased, which helps in retrieving similar clone methods effectively. Regarding code to code search, we achieve the best recall among state-of-the art search engine performances. This depicts the effectiveness of our proposed methodology of creating natural language documents associated with specific clone methods. However, we do not see much difference in the performance

TABLE 11. Benchmark queries to evaluate clone-seeker variants on Project CodeNet dataset ('∗' = StackOverflow, '◦' = GeeksForGeeks).

Id	Questions
p00001∗	How to sort ArrayList in decreasing order?
p00003 ∗	Checking if a triangle is a right triangle
p00005∗	How to find GCD, LCM on a set of numbers
p00006 ∗	Reverse a string in Java
p02255 ∗	Simple Insertion Sort in Java
p02256 ∗	Java: get greatest common divisor
p02257 ∗	find all prime numbers from array
p02259 ∗	BubbleSort Implementation
p02260 ∗	Java - Selection Sort Algorithm
p02263 ∗	Evaluate the value of an arithmetic expression in Reverse Polish Notation
p02264 ∗	Round robin scheduling algorithm in Java using AtomicBoolean
p02268 ∗	Binary Search through 2 Array
p02381 ∗	How to calculate standard deviation using JAVA
p02388 ∗	Raising a number to a power in Java
p02389 ∗	Calculating perimeter and area of a rectangle
p02390 ∗	Convert seconds value to hours minutes seconds?
p02393 ∗	Trouble sorting three numbers in ascending order in Java
p02394 ∗	Check if circle contains rectangle
p02397 ∗	sorting integers in order lowest to highest java
p02399 ∗	How to get a remainder in java
p02400 ∗	How to get the circumference, area and radius of a circle in java?
p02401 ∗	Write a program that accepts two numbers and a operator like (+,-,*, /) as command line arguments and perform the operation indicated by operator
p02402 ∗	Program that will compute the maximum and the minimum of a sequence of integers
p02407 ∗	How do I reverse an int array in Java?
p02410 ∗	Matrix-vector Multiplication
p02414 ∗	Matrix multiplication using arrays
p02415 ∗	Java-toggle alphabet case in string
p02416 ∗	How to sum digits of an integer in java?
p02417 ∗	Count letters in a string
p02419 ∗	Count the number of Occurrences of a Word in a String
p02577 ∗	Java, Check if integer is multiple of a number
p02627 ∗	Java Program to test if a character is uppercase/lowercase/number/vowel
p02657 ∗	Java multiplying between two numbers
p02659 ∗	Extracting the integer and fractional part from Bigdecimal in Java
p02705 ∗	Finding the radius, diameter, area and circumference of a circle
p02711 ∗	How to check if a number contains a certain digit?
p02725 ∗	Traveling Salesman in Java
p02730 ∗	Check string for palindrome
p02731 ◦	Maximize volume of cuboid with given sum of sides
p02766 ◦	Given a number N in decimal base, find number of its digits in any base (base b)
p02778 ∗	How to replace all characters in a Java string with stars
p03041 ∗	Change character case at specific position in java
p03308 ∗	Maximum difference in an array
p03416 ∗	How to find out all palindromic numbers
p03455 ◦	Check whether product of 'n' numbers is even or odd
p03493 ∗	Count the squares in the grid

of Clone-Seeker variants among each other for code-to-code search. This requires a more in-depth exploration of different annotation techniques, which can help Clone-Seeker (Manual) and Clone-Seeker (Automatic) to achieve a better performance compared to the baseline. Overall, Clone-Seeker (Baseline) and Clone-Seeker (Automatic) achieve almost the same performance for both search types, i.e. code-to-code search and natural-language to code search. This shows that adding more tokens in terms of automatic annotation does not improve the search a lot and different annotation techniques are required to explore in the future.

VI. LIMITATIONS

There are certain limitations of this work, which can be further explored and mitigated in the future. Clone-Seeker depends upon a user representing each clone class manually. A more accurate manually annotated description can be applied to get more effective results. We only apply limited threshold values for k, in order to annotate each clone class automatically with a set of words. We plan to investigate more threshold values for k such as 1, 2, 3, and 4 etc in the future. We only apply word counting technique to extract the keywords automatically. In the future, we plan to investigate several keyword extraction techniques such as word degree, TF-IDF (which we use for searching in this work, not for keyword extraction), and RAKE [60].

We have used limited natural language queries to validate the effectiveness of Clone-Seeker approaches on the BigCloneBench and Project CodeNet datasets. Although these queries are real-world queries collected from various websites, admittedly they do not cover all types of queries that a developer may ask. In the future, we plan to reduce this threat to validity by investigating more queries. Also, although we have evaluated our approach on the BigCloneBench and Project CodeNet datasets, more evaluation can be conducted on other available clone datasets. This way we can have a more thorough evaluation of the generalizability of our approach.

TABLE 12. Evaluation results of stack overflow queries in terms of MRR(%) and precision (P@k) on Project CodeNet dataset.

	Clone-Seeker(Baseline)				Clone-Seeker(Manual)				Clone-Seeker(Automatic)			
Id	**MRR**	**P@1**	**P@5**	**P@10**	**MRR**	**P@1**	**P@5**	**P@10**	**MRR**	**P@1**	**P@5**	**P@10**
p00001	0.5	0	0.8	0.9	0.33	0	0.4	0.3	0.5	0	0.8	0.9
p00003	1	1	1	0.9	1	1	1	1	1	1	1	0.9
p00005	1	1	1	1	0.25	0	0.4	0.7	1	1	1	1
p00006	1	1	0.8	0.9	0.142857	0	0	0.2	1	1	0.8	0.9
p02255	1	1	1	1	1	1	1	1	1	1	1	1
p02256	1	1	0.8	0.8	1	1	1	1	1	1	0.8	0.8
p02257	1	1	1	0.8	1	1	1	1	1	1	1	0.8
p02259	1	1	0.8	0.9	1	1	1	1	1	1	0.8	0.9
p02260	0.5	0	0.8	0.9	0.5	0	0.8	0.9	0.5	0	0.8	0.9
p02263	1	1	0.8	0.6	1	1	1	1	1	1	0.8	0.6
p02264	0.5	0	0.4	0.2	0.5	0	0.8	0.9	0.5	0	0.4	0.2
p02268	1	1	1	1	1	1	1	1	1	1	1	1
p02381	1	1	0.8	0.6	1	1	1	1	1	1	0.8	0.6
p02388	1	1	0.4	0.2	1	1	1	1	1	1	0.4	0.2
p02389	1	1	0.8	0.6	1	1	1	1	1	1	0.8	0.6
p02390	1	1	1	1	1	1	1	1	1	1	1	1
p02393	1	1	0.4	0.3	1	1	1	1	1	1	0.4	0.3
p02394	1	1	1	0.7	1	1	1	1	1	1	1	0.7
p02397	0	0	0	0	0.25	0	0.4	0.6	0	0	0	0
p02399	0	0	0	0	0.33	0	0.6	0.8	0	0	0	0
p02400	1	1	0.6	0.8	0.5	0	0.4	0.4	1	1	0.6	0.8
p02401	0.125	0	0	0.2	0.25	0	0.2	0.1	0.125	0	0	0.2
p02402	1	1	0.2	0.1	1	1	1	1	1	1	0.2	0.1
p02407	0.33	0	0.2	0.1	0.33	0	0.6	0.8	0.33	0	0.2	0.1
p02410	1	1	1	0.9	1	1	1	1	1	1	1	0.9
p02414	1	1	0.4	0.3	1	1	0.8	0.9	1	1	0.4	0.3
p02415	0.125	0	0	0.1	1	1	1	1	0.125	0	0	0.1
p02416	0.2	0	0.2	0.2	0.33	0	0.6	0.8	0.125	0	0	0.2
p02417	0.33	0	0.4	0.7	1	1	1	1	0.33	0	0.4	0.7
p02419	0.5	0	0.8	0.8	1	1	1	1	0.5	0	0.8	0.8
p02577	0.5	0	0.2	0.2	1	1	1	1	0.5	0	0.2	0.2
p02627	0	0	0	0	0.33	0	0.6	0.8	0	0	0	0
p02657	0	0	0	0	1	1	1	1	0	0	0	0
p02659	0.1	0	0	0.1	1	1	1	1	0.1	0	0	0.1
p02705	0.5	0	0.4	0.2	1	1	0.6	0.6	0.5	0	0.4	0.2
p02711	0.11	0	0	0.2	1	1	1	1	0.11	0	0	0.2
p02725	1	1	1	0.5	1	1	1	1	1	1	1	0.5
p02730	1	1	0.6	0.4	1	1	1	1	1	1	0.6	0.4
p02731	0.5	0	0.8	0.6	1	1	1	1	0.5	0	0.8	0.6
p02766	0.5	0	0.8	0.6	1	1	1	1	0.5	0	0.8	0.6
p02778	0.1	0	0	0.1	1	1	1	0.9	0.11	0	0	0.1
p03041	1	1	0.2	0.3	1	1	1	1	1	1	0.2	0.3
p03308	0.5	0	0.6	0.8	1	1	1	1	0.5	0	0.6	0.8
p03416	0.5	0	0.8	0.6	0.33	0	0.6	0.8	0.5	0	0.8	0.6
p03455	1	1	1	0.9	1	1	1	1	1	1	1	0.9
p03493	0.5	0	0.2	0.1	1	1	1	1	0.5	0	0.2	0.1
Average	**0.651**	**0.5**	**0.543**	**0.502**	**0.813**	**0.717**	**0.843**	**0.880**	**0.650**	**0.5**	**0.540**	**0.502**

VII. CONCLUSION AND FUTURE WORK

In this paper, we propose a novel approach of retrieving clone methods effectively. Our approach can retrieve clone methods effectively, based on a search query in terms of source code terms as well as natural language. We apply different annotation strategies to build metadata (i.e. natural language document) for clone methods. We successfully demonstrate the effectiveness of Clone-Seeker approaches through empirical evaluation on BigCloneBench and Project CodeNet datasets. We achieve the best recall for Type-4 clones as compared to the state-of-the-art on BigCloneBench dataset. Similarly, we demonstrate the effectiveness of our approach by applying natural language queries collected from various websites. We achieve best performance in terms of MRR and precision for manual annotation technique on both datasets.

While we have promising results, our work can be extended in various directions. In this study, our focus is on introducing an effective technique for retrieving clone methods and clone files. However, code clones can be of different types and granularity levels (e.g. simple clones, structural clones, and clones of other artifact types such as models [25], [83]). We plan to investigate whether our approach can be used for different clone types and granularities in the future. We only use the TF-IDF technique in searching and retrieval as a proof of concept, however there are several other information retrieval techniques such as word2vec [84], glove [85], etc., which can be applied and comparatively evaluated. Similarly, neural networks can also be applied to build an effective search approach. Joint neural network models [48] for the natural language documents and clone methods can be explored to have effective search results. Moreover, if comments are a reliable source of information in some particular dataset, then they can also be utilized in annotating clone classes. This may help in effectively retrieving clone methods based on search query.

ACKNOWLEDGMENT

The authors acknowledge SURFsara and TU/e HPC Cluster for providing them computational credits for the experiments.

REFERENCES

[1] M. Allamanis, D. Tarlow, A. Gordon, and Y. Wei, "Bimodal modelling of source code and natural language," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2123–2132.

[2] 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 *Proc. 40th Int. Conf. Softw. Eng.*, 2018, pp. 946–957.

[3] F. Lv, H. Zhang, J.-G. Lou, S. Wang, D. Zhang, and J. Zhao, "CodeHow: Effective code search based on API understanding and extended Boolean model (E)," in *Proc. 30th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2015, pp. 260–270.

[4] C. Sadowski, K. T. Stolee, and S. Elbaum, "How developers search for code: A case study," in *Proc. 10th Joint Meeting Found. Softw. Eng.*, Aug. 2015, pp. 191–201.

[5] I. Keivanloo, J. Rilling, and Y. Zou, "Spotting working code examples," in *Proc. 36th Int. Conf. Softw. Eng.*, May 2014, pp. 664–675.

[6] H. Zhong, T. Xie, L. Zhang, J. Pei, and H. Mei, "MAPO: Mining and recommending API usage patterns," in *Proc. Eur. Conf. Object-Oriented Program.* Berlin, Germany: Springer, Jul. 2009, pp. 318–343.

[7] A. J. Ko, H. H. Aung, and B. A. Myers, "Eliciting design requirements for maintenance-oriented ides: A detailed study of corrective and perfective maintenance tasks," in *Proc. 27th Int. Conf. Softw. Eng. (ICSE)*, 2005, pp. 126–135.

[8] R. Hoffmann, J. Fogarty, and D. S. Weld, "Assieme: Finding and leveraging implicit references in a web search interface for programmers," in *Proc. 27th Int. Conf. Softw. Eng.*, 2007, pp. 13–22.

[9] K. T. Stolee, S. Elbaum, and D. Dobos, "Solving the search for source code," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 3, pp. 1–45, May 2014.

[10] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," *IEEE Trans. Softw. Eng.*, vol. 32, no. 12, pp. 971–987, Dec. 2006.

[11] T. Xie and J. Pei, "MAPO: Mining API usages from open source repositories," in *Proc. Int. Workshop Mining Softw. Repositories (MSR)*, 2006, pp. 54–57.

[12] M. P. Robillard, "What makes APIs hard to learn? Answers from developers," *IEEE Softw.*, vol. 26, no. 6, pp. 27–34, Nov. 2009.

[13] S. K. Bajracharya and C. V. Lopes, "Analyzing and mining a code search engine usage log," *Empirical Softw. Eng.*, vol. 17, nos. 4–5, pp. 424–466, Aug. 2012.

[14] L. Martie, A. V. D. Hoek, and T. Kwak, "Understanding the impact of support for iteration on code search," in *Proc. 11th Joint Meeting Found. Softw. Eng.*, Aug. 2017, pp. 774–785.

[15] X. Xia, L. Bao, D. Lo, P. S. Kochhar, A. E. Hassan, and Z. Xing, "What do developers search for on the web?" *Empirical Softw. Eng.*, vol. 22, no. 6, pp. 3149–3185, Dec. 2017.

[16] J. Kim, S. Lee, S.-W. Hwang, and S. Kim, "Towards an intelligent code search engine," in *Proc. AAAI Conf. Artif. Intell.*, 2010, vol. 24, no. 1, pp. 1–6.

[17] R. P. L. Buse and W. Weimer, "Synthesizing API usage examples," in *Proc. 34th Int. Conf. Softw. Eng. (ICSE)*, Jun. 2012, pp. 782–792.

[18] D. Mandelin, L. Xu, R. Bodík, and D. Kimelman, "Jungloid mining: Helping to navigate the API jungle," *ACM SIGPLAN Notices*, vol. 40, no. 6, pp. 48–61, 2005.

[19] M. R. Marri, S. Thummalapenta, and T. Xie, "Improving software quality via code searching and mining," in *Proc. ICSE Workshop Search-Driven Develop.-Users, Infrastruct., Tools Eval.*, May 2009, pp. 33–36.

[20] C. J. Kapser and M. W. Godfrey, "'Cloning considered harmful' considered harmful: Patterns of cloning in software," *Empirical Softw. Eng.*, vol. 13, no. 6, pp. 645–692, 2008.

[21] M. Hammad, Ö. Babur, H. A. Basit, and M. Van Den Brand, "DeepClone: Modeling clones to generate code predictions," in *Proc. Int. Conf. Softw. Softw. Reuse*. Cham, Switzerland: Springer, Dec. 2020, pp. 135–151.

[22] M. Hammad, Ö. Babur, H. A. Basit, and M. V. D. Brand, "Clone-advisor: Recommending code tokens and clone methods with deep learning and information retrieval," *PeerJ Comput. Sci.*, vol. 7, p. e737, Nov. 2021.

[23] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Sci. Comput. Program.*, vol. 74, no. 7, pp. 470–495, 2009.

[24] C. Ragkhitwetsagul, J. Krinke, and D. Clark, "A comparison of code similarity analysers," *Empirical Softw. Eng.*, vol. 23, no. 4, pp. 2464–2519, 2018.

[25] M. Hammad, H. A. Basit, S. Jarzabek, and R. Koschke, "A systematic mapping study of clone visualization," *Comput. Sci. Rev.*, vol. 37, Aug. 2020, Art. no. 100266.

[26] M. Hammad, H. A. Basit, S. Jarzabek, and R. Koschke, "Visualization of clones," in *Code Clone Analysis*. Singapore: Springer, 2021, pp. 107–120.

[27] 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 *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, Sep. 2014, pp. 476–480.

[28] J. Svajlenko and C. K. Roy, "BigCloneEval: A clone detection tool evaluation framework with BigCloneBench," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Oct. 2016, pp. 596–600.

[29] B. Baudry, S. Allier, and M. Monperrus, "Tailored source code transformations to synthesize computationally diverse program variants," in *Proc. Int. Symp. Softw. Test. Anal. (ISSTA)*, 2014, pp. 149–159.

[30] Y. Ke, K. T. Stolee, C. L. Goues, and Y. Brun, "Repairing programs with semantic code search," in *Proc. 30th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2015, pp. 295–306.

[31] E. T. Barr, M. Harman, Y. Jia, A. Marginean, and J. Petke, "Automated software transplantation," in *Proc. Int. Symp. Softw. Test. Anal.*, Jul. 2015, pp. 257–269.

[32] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais, "The vocabulary problem in human-system communication," *Commun. ACM*, vol. 30, no. 11, pp. 964–971, 1987.

[33] P. Pathak, M. Gordon, and W. Fan, "Effective information retrieval using genetic algorithms based matching functions adaptation," in *Proc. 33rd Annu. Hawaii Int. Conf. Syst. Sci.*, Jan. 2000, pp. 1–8.

[34] E. Hill, L. Pollock, and K. Vijay-Shanker, "Improving source code search with natural language phrasal representations of method signatures," in *Proc. 26th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2011, pp. 524–527.

[35] J. Yang and L. Tan, "Inferring semantically related words from software context," in *Proc. 9th IEEE Work. Conf. Mining Softw. Repositories (MSR)*, Jun. 2012, pp. 161–170.

[36] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies, "Automatic query reformulations for text retrieval in software engineering," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, May 2013, pp. 842–851.

[37] C. Ragkhitwetsagul and J. Krinke, "Siamese: Scalable and incremental code clone search via multiple code representations," *Empirical Softw. Eng.*, vol. 24, no. 4, pp. 2236–2284, Aug. 2019.

[38] R. Puri, D. S. Kung, G. Janssen, W. Zhang, G. Domeniconi, V. Zolotov, J. Dolby, J. Chen, M. Choudhury, L. Decker, V. Thost, L. Buratti, S. Pujar, S. Ramji, U. Finkler, S. Malaika, and F. Reiss, "CodeNet: A large-scale AI for code dataset for learning a diversity of coding tasks," 2021, *arXiv:2105.12655*.

[39] M. Lu, X. Sun, S. Wang, D. Lo, and Y. Duan, "Query expansion via WordNet for effective code search," in *Proc. IEEE 22nd Int. Conf. Softw. Anal., Evol., Reeng. (SANER)*, Mar. 2015, pp. 545–549.

[40] G. Sridhara, E. Hill, L. Pollock, and K. Vijay-Shanker, "Identifying word relations in software: A comparative study of semantic similarity tools," in *Proc. 16th IEEE Int. Conf. Program Comprehension*, Jun. 2008, pp. 123–132.

[41] C. Mcmillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: Finding relevant functions and their usage," in *Proc. 33rd Int. Conf. Softw. Eng.*, May 2011, pp. 111–120.

[42] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, "Mining StackOverflow to turn the IDE into a self-confident programming prompter," in *Proc. 11th Work. Conf. Mining Softw. Repositories (MSR)*, 2014, pp. 102–111.

[43] X. Li, Z. Wang, Q. Wang, S. Yan, T. Xie, and H. Mei, "Relationship-aware code search for JavaScript frameworks," in *Proc. 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, Nov. 2016, pp. 690–701.

[44] J. Davies, D. M. German, M. W. Godfrey, and A. Hindle, "Software Bertillonage: Finding the provenance of an entity," in *Proc. 8th Work. Conf. Mining Softw. Repositories (MSR)*, 2011, pp. 183–192.

[45] S. Sachdev, H. Li, S. Luan, S. Kim, K. Sen, and S. Chandra, "Retrieval on source code: A neural code search," in *Proc. 2nd ACM SIGPLAN Int. Workshop Mach. Learn. Program. Lang.*, Jun. 2018, pp. 31–41.

[46] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, and C. Lopes, "Sourcerer: A search engine for open source code supporting structure-based search," in *Proc. Companion to 21st ACM SIGPLAN Conf. Object-Oriented Program. Syst., Lang., Appl. (OOPSLA)*, 2006, pp. 681–682.

[47] S. Zhou, B. Shen, and H. Zhong, "Lancer: Your code tell me what you need," in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2019, pp. 1202–1205.

[48] X. Gu, H. Zhang, and S. Kim, "Deep code search," in *Proc. 40th Int. Conf. Softw. Eng. (ICSE)*, May 2018, pp. 933–944.

[49] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "CodeSearchNet challenge: Evaluating the state of semantic code search," 2019, *arXiv:1909.09436*.

[50] M. Kamp, P. Kreutzer, and M. Philippsen, "SeSaMe: A data set of semantically similar Java methods," in *Proc. IEEE/ACM 16th Int. Conf. Mining Softw. Repositories (MSR)*, May 2019, pp. 529–533.

[51] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, "Convolutional neural networks over tree structures for programming language processing," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 1–7.

[52] F. Ullah, H. Naeem, S. Jabbar, S. Khalid, M. A. Latif, F. Al-Turjman, and L. Mostarda, "Cyber security threats detection in Internet of Things using deep learning approach," *IEEE Access*, vol. 7, pp. 124379–124389, 2019.

[53] T. Fritz, G. C. Murphy, E. Murphy-Hill, J. Ou, and E. Hill, "Degree-of-knowledge: Modeling a developer's knowledge of code," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 2, pp. 1–42, Mar. 2014.

[54] S. Yan, H. Yu, Y. Chen, B. Shen, and L. Jiang, "Are the code snippets what we are searching for? A benchmark and an empirical study on code search with natural-language queries," in *Proc. IEEE 27th Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Feb. 2020, pp. 344–354.

[55] C. Ragkhitwetsagul, J. Krinke, M. Paixao, G. Bianco, and R. Oliveto, "Toxic code snippets on stack overflow," *IEEE Trans. Softw. Eng.*, vol. 47, no. 3, pp. 560–581, Mar. 2021.

[56] J. Pustejovsky and A. Stubbs, *Natural Language Annotation for Machine Learning: A Guide to Corpus-Building for Applications*. Sebastopol, CA, USA: O'Reilly Media, 2012.

[57] A. Hulth, "Improved automatic keyword extraction given more linguistic knowledge," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2003, pp. 216–223.

[58] C. Zhang, "Automatic keyword extraction from documents using conditional random fields," *J. Comput. Inf. Syst.*, vol. 4, no. 3, pp. 1169–1180, 2008.

[59] E. Cano and O. Bojar, "Keyphrase generation: A multi-aspect survey," in *Proc. 25th Conf. Open Innov. Assoc. (FRUCT)*, Nov. 2019, pp. 85–94.

[60] Z. Nasar, S. W. Jaffry, and M. K. Malik, "Textual keyword extraction and summarization: State-of-the-art," *Inf. Process. Manage.*, vol. 56, no. 6, Nov. 2019, Art. no. 102088.

[61] A. Baron, P. Rayson, and D. Archer, "Word frequency and key word statistics in corpus linguistics," *Anglistik*, vol. 20, no. 1, pp. 41–67, 2009.

[62] P. Rayson and R. Garside, "Comparing corpora using frequency profiling," in *Proc. Workshop Comparing Corpora*, 2000, pp. 1–6.

[63] M. E. I. Kipp and D. G. Campbell, "Patterns and inconsistencies in collaborative tagging systems: An examination of tagging practices," *Proc. Amer. Soc. Inf. Sci. Technol.*, vol. 43, no. 1, pp. 1–18, Oct. 2007.

[64] J. Ganesh, R. Parthasarathi, T. V. Geetha, and J. Balaji, "Pattern based bootstrapping technique for tamil POS tagging," in *Mining Intelligence and Knowledge Exploration*. Cham, Switzerland: Springer, 2014, pp. 256–267.

[65] A. G. Lewis, H. Schriefers, M. Bastiaansen, and J.-M. Schoffelen, "Assessing the utility of frequency tagging for tracking memory-based reactivation of word representations," *Sci. Rep.*, vol. 8, no. 1, pp. 1–12, Dec. 2018.

[66] A. Chacoma and D. H. Zanette, "Word frequency–rank relationship in tagged texts," *Phys. A, Stat. Mech. Appl.*, vol. 574, Jul. 2021, Art. no. 126020.

[67] C. Tribble and G. Jones, *Concordances in the Classroom: A Resource Guide for Teachers*. Athelstan, 1997.

[68] A. Marcus and G. Antoniol, "On the use of text retrieval techniques in software engineering," in *Proc. 34th IEEE/ACM Int. Conf. Softw. Eng.*, Jun. 2012.

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

[70] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, Jun. 2008.

[71] J. Beel, B. Gipp, S. Langer, and C. Breitinger, "Paper recommender systems: A literature survey," *Int. J. Digit. Libraries*, vol. 17, no. 4, pp. 305–338, Nov. 2016.

[72] K. M. Ramachandran and C. P. Tsokos, *Mathematical Statistics With Applications in R*. New York, NY, USA: Academic Press, 2020.

[73] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to Information Retrieval*, vol. 39. Cambridge, U.K.: Cambridge Univ. Press, 2008.

[74] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "SourcererCC: Scaling code clone detection to big-code," in *Proc. 38th Int. Conf. Softw. Eng.*, May 2016, pp. 1157–1168.

[75] J. R. Cordy and C. K. Roy, "The NiCad clone detector," in *Proc. IEEE 19th Int. Conf. Program Comprehension*, Jun. 2011, pp. 219–220.

[76] N. Göde and R. Koschke, "Incremental clone detection," in *Proc. 13th Eur. Conf. Softw. Maintenance Reeng.*, 2009, pp. 219–228.

[77] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multilinguistic token-based code clone detection system for large scale source code," *IEEE Trans. Softw. Eng.*, vol. 28, no. 7, pp. 654–670, Jul. 2008.

[78] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "DECKARD: Scalable and accurate tree-based detection of code clones," in *Proc. 29th Int. Conf. Softw. Eng. (ICSE)*, May 2007, pp. 96–105.

[79] T. Diamantopoulos and A. Symeonidis, "Employing source code information to improve question-answering in stack overflow," in *Proc. IEEE/ACM 12th Work. Conf. Mining Softw. Repositories*, May 2015, pp. 454–457.

[80] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming Q&A in StackOverflow," in *Proc. 28th IEEE Int. Conf. Softw. Maintenance (ICSM)*, Sep. 2012, pp. 25–34.

[81] P. M. Papadopoulos, T. D. Lagkas, and S. N. Demetriadis, "How to improve the peer review method: Free-selection vs assigned-pair protocol evaluated in a computer networking course," *Comput. Educ.*, vol. 59, no. 2, pp. 182–195, Sep. 2012.

[82] A. Luxton-Reilly, "A systematic review of tools that support peer assessment," *Comput. Sci. Educ.*, vol. 19, no. 4, pp. 209–232, Dec. 2009.

[83] Ö. Babur, L. Cleophas, and M. van den Brand, "Metamodel clone detection with SAMOS," *J. Comput. Lang.*, vol. 51, pp. 57–74, Apr. 2019.

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

[85] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1532–1543.