1-2024

Big Code Search: A Bibliography

Kisub KIM
Singapore Management University, kisubkim@smu.edu.sg

Sankalp GHATPANDE

Dongsun KIM

Xin ZHOU
Singapore Management University, xinzhou.2020@phdcs.smu.edu.sg

Kui LIU

See next page for additional authors

Citation

KIM, Kisub; GHATPANDE, Sankalp; KIM, Dongsun; ZHOU, Xin; LIU, Kui; BISSYANDE, Tegawende F.; KLEIN, Jacques; and LE, Traon Yves. Big Code Search: A Bibliography. (2024). *ACM Computing Surveys*. 56, (1), 1-49.
Available at: https://ink.library.smu.edu.sg/sis_research/8281

Big Code Search: A Bibliography

KISUB KIM, Singapore Management University, Singapore
SANKALP GHATPANDE, Independent Researcher, Luxembourg
DONGSUN KIM, Kyungpook National University, Republic of Korea
XIN ZHOU, Singapore Management University, Singapore
KUI LIU, Huawei, China
TEGAWENDÉ F. BISSYANDÉ, JACQUES KLEIN, and YVES LE TRAON, University of Luxembourg, Luxembourg

Code search is an essential task in software development. Developers often search the internet and other code databases for necessary source code snippets to ease the development efforts. Code search techniques also help learn programming as novice programmers or students can quickly retrieve (hopefully good) examples already used in actual software projects. Given the recurrence of the code search activity in software development, there is an increasing interest in the research community. To improve the code search experience, the research community suggests many code search tools and techniques. These tools and techniques leverage several different ideas and claim a better code search performance. However, it is still challenging to illustrate a comprehensive view of the field considering that existing studies generally explore narrow and limited subsets of used components. This study aims to devise a grounded approach to understanding the procedure for code search and build an operational taxonomy capturing the critical facets of code search techniques. Additionally, we investigate evaluation methods, benchmarks, and datasets used in the field of code search.

CCS Concepts: • **Software and its engineering → Maintaining software;**

Additional Key Words and Phrases: Code search, code recommendation, code retrieval, find code, code snippet, code search procedure

This work was supported by the European Research Council (ERC), under the European Union's Horizon 2020 research and innovation programme (grant agreement 949014); and Fonds National de la Recherche (FNR), Luxembourg, under FNR-AFR PhD/11623818 and the National Research Foundation, Singapore, under its Industry Alignment Fund–Pre-positioning (IAF-PP) Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore. This work was also supported by a National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (2021R1A5A1021944 and 2021R1I1A3048013); the National Natural Science Foundation of China (62172214); the Natural Science Foundation of Jiangsu Province, China (BK20210279). Additionally, the research was supported by Kyungpook National University Research Fund, 2020.
Authors' addresses: K. Kim and X. Zhou, Singapore Management University, 81 Victoria St, Singapore 188065, Singapore; emails: kisubkim@smu.edu.sg, xinzhou.2020@phdcs.smu.edu.sg; S. Ghatpande, Independent Researcher, 6 Rue Richard Coudenhove-Kalergi, 1359 Kirchberg, Luxembourg; email: contact@sghatpande.eu; D. Kim (corresponding author), Kyungpook National University, 80 Daehak-ro, Buk-gu, Daegu, South Korea; email: darkrsw@knu.ac.kr; K. Liu (corresponding author), Huawei, No. 360, Jiangshu Road, Binjiang District Hangzhou Zhejiang, 310051, China; email: brucekuiliu@gmail.com; T. F. Bissyandé, J. Klein, and Y. L. Traon, University of Luxembourg, 6 Rue Richard Coudenhove-Kalergi, 1359 Kirchberg, Luxembourg; emails: tegawende.bissyande@uni.lu, jacques.klein@uni.lu, Yves.LeTraon@uni.lu.

0360-0300/2023/08-ART25 $15.00
https://doi.org/10.1145/3604905

ACM Reference format:
Kisub Kim, Sankalp Ghatpande, Dongsun Kim, Xin Zhou, Kui Liu, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2023. Big Code Search: A Bibliography. *ACM Comput. Surv.* 56, 1, Article 25 (August 2023), 49 pages.
https://doi.org/10.1145/3604905

1 INTRODUCTION

Code search is one of the most frequent activities in software development, as developers tend to write programs based on existing programs rather than writing them from scratch [94, 182, 234, 248, 252, 295]. Many programs consist of routines, data structures, and resources that are also implemented by other programs [70]. Developers are indeed recurrently writing code to address similar tasks or cloning (e.g., via copy/paste) other code. Toward easing software development, searching for source code on the internet became a typical activity for developers [19, 249, 262]. In particular, developers often search for source code to reuse or to consider as reference examples [72, 249] to help them identify the programming concepts that are required for solving coding tasks [14, 30, 94, 176, 248] or to fact-checking (i.e., in contrast to exploratory usages [174]) on the availability of different implementations for a given algorithm. Furthermore, the core concept and principles of code search research is also applied to other software engineering tasks such as concept/feature/concern location [58], code clone detection [126], code completion [32, 229], match and transform [207], vulnerability detection [37, 300], bug localization [3], and automatic program repair [161].

Developers often search other external sources rather than local sources for necessary code snippets. The external sources may include the internet, super repositories (e.g., GitHub [289] and Bitbucket [25]), blogs, and **Question and Answer (Q&A)** forums (e.g., Stack Overflow [257]). We define this type of code search as "big code search" since it explores a large space of resources. In contrast to big code search, local code search, also called *concept/feature/concern location*, scans features and functionalities available only in the local projects or search space [58]. Local code search approaches often take a single software repository as the search base. This intra-project setting tends to provide high retrieval performance and fast speed since the search base is relatively small, and such approaches are implemented considering their characteristics [232]. Despite the high retrieval quality, such approaches (e.g., [65, 67]) have clear limitations with the variety of the source code and the generalization of the search engine. The research community started to propose *big code search approaches* to address these issues, containing at least two repositories as their search base to provide abundant source code. Especially, many of the big code search approaches are built upon super repositories such as GitHub [289] or SourceForge [256] to cover various requirements from the developers and users.

Given the importance of code search in software development, the research community has invested substantial effort in the field of code search, developing new techniques, applying new methods, and collecting new data to improve efficiency and effectiveness. In broad terms, code search is a procedure composed of a series of activities aiming to retrieve relevant code snippets according to the user specification. These activities include (1) selection of the output type, (2) creation of a search base, (3) indexing the search base, (4) formulation of user specifications into search queries, (5) obtainment of code snippets relevant to the user query, and (6) demonstration of relevant results to users. The ability to characterize the properties of each activity remains crucial toward understanding the core concepts and principles of code search. To that end, we summarize the properties and characteristics of code search approaches and use them to establish a procedure for code search. Eventually, we propose a grounded approach to establish a standard procedure for code search and build an operational taxonomy on this procedure.

Based on the preceding observations, this survey provides a comprehensive view of big code search by addressing the following challenges. First, practitioners often have difficulties selecting an appropriate tool or technique since there are too many code search approaches to consider. Second, the **Vocabulary Mismatch Problem (VMP)** [142, 221, 241, 253] still exists in the field of code search. Third, despite the substantial efforts by the researchers, code search benchmarks still remain of low quality. Fourth, recent studies invested less effort in extensibility (i.e., for multiple programming languages) and usability (i.e., for practical usage like binary code search because of the obfuscation). Fifth, most existing approaches lack consensus for the specific requirements (e.g., a code snippet that is memory efficient).

It is still challenging to explore the world of big code search due to a lack of well-organized knowledge that links to the building/managing procedures and suitable literature. Yet, to the best of our knowledge, there has not been any attempt to undertake work that would address this issue and provide literature covering the big code search domain in its entirety. Although Liu et al. [158] recently submitted a survey that is based on the input and output of code search engines, their work focuses mainly on understanding the publication trend within the domain that provides a segmentation between the types of publication undertaken, venues, and trends. We believe that further in-depth analysis is necessary for a better understanding of big code search.

This survey aims to provide the reader with a complete view of the field, starting from the first known publication of big code search. We consider a total of 137 big code search approaches, which are all based on multiple projects (i.e., at least two projects as a search base). Due to the strict page limitation, we append big tables in Appendix A.[1]

Concretely, our work provides many essential contributions to the code search field, such as the following:

- A novel systematic literature review on 137 code search approaches published until the end of 2020.
- A profound literature search strategy with snowballing to bring to light unrevealed approaches of code search.
- An overview of the field of code search and its historical evolution trend to highlight what has been done so far.
- Identifying a general procedure of code search that can help to understand fundamental concepts of code search.
- An operational taxonomy of code search that can guide researchers and practitioners to locate code search approaches that are most suited to their tasks.
- Analysis of existing dataset and the benchmarks of code search.
- Discussion of the open issues and potential research directions.

2 DIFFERENCES FROM OTHER SURVEYS ON CODE SEARCH TECHNIQUES

As there have been other existing surveys on code search, we clarify their insights and differences from our survey in this section. We found several surveys [5, 56, 125, 158, 219, 245] that can be directly or potentially related to the field of code search. We clarify the differences from the most relevant survey [158] as follows:

[1]https://github.com/FalconLK/BigCodeSearch/blob/main/Survey_Appendix.pdf.

- *Survey scope*: We conducted a strict snowballing to collect the complete list of code search techniques by including DBLP and arXiv. Especially, arXiv contains major industrial publications on the field, which tend to have high research values. This allowed us to discover 137 studies, whereas Liu et al. [158] found 81.
- *Taxonomy*: Thanks to the complete list, we identify further categories of code search studies (e.g., code search based on dynamic information described in Section 5.2).
- *Procedures*: We not only provide a review of the techniques but also illustrate the full procedure of code search, which assists in better understanding by linking each procedure with our taxonomy.
- *Deeper investigation*: Finding more techniques and illustrating the full procedure of code search allowed us to investigate deeper into the field, which led to the introduction of further techniques. For instance, Liu et al. [158] only introduce the "inverted" technique for indexing of code search, whereas we include all the other variants such as "graph indexing" and "ID-based indexing." We believe that this comprehensive and extensive investigation could help researchers and practitioners have better understanding and may motivate them to propose better approaches.
- *Better correlations*: Our code search procedure derives an intuitive understanding of the taxonomy of code search. For example, the query formulation phase in the procedure is directly related to the studies in "code search based on query reformulation" of the taxonomy. This can be an easy guide for the selection of the most suitable techniques during the design/implementation process of each task.
- *Opportunity discovery*: There exist missed and under-explored research opportunities we newly disclosed. For instance, a well-designed query language is good at modeling the structure (e.g., loops), and it can accurately express more complicated search patterns than typical text-based queries. Although this is not a trendy topic for the code search field, these may further improve the performance of tools.
- *Additional issues*: We discover further phenomenons such as "There exist too many code search approaches to consider," which may disturb practitioners as they must select the most suitable one for specific tasks.

Furthermore, we illuminate the differences from the most recent survey [56] in the following points:

- *Categorization*: We figure out a finer-grained categorization of code search techniques, as a more detailed and nuanced categorization of techniques can help readers better understand the similarities and differences between different approaches to code search. This can be particularly helpful for both researchers and practitioners who are interested in implementing or comparing different techniques.
- *Comprehensiveness*: We have concrete tables for each category, and diverse sub-techniques may help readers comprehend the field easier. This can be valuable, as providing clear and organized information in tables can help readers quickly and easily understand the landscape of the field. It can also make it easier for readers to compare and contrast different techniques within each category.
- *Datasets*: We further support in-depth details of the datasets that can be directly and potentially used for code search engines. It is crucial for researchers to replicate experiments and build upon previous work. Our systematic literature review can provide a valuable resource for researchers who are interested in practical applications as well as developing or evaluating code search engines.

- *Complementary points*: The different open issues and challenges are found, and both of them can be complementary to each other. For example, this article raises the VMP, extensibility, and usability as the issues in the field, whereas the similar survey [56] includes additional usage scenarios and cross fertilization with other fields.

3 PAPER COLLECTION AND REVIEW SCHEMA

This section describes the survey scope, paper collection methodology, and brief statistics of the collected papers.

3.1 Survey Scope

The scope of our survey targets comprehensive internet-scale code search engines that take input from users and then retrieve/recommend/suggest code snippets/examples.

We apply the following criteria for the inclusion of papers in this survey:

- Papers that propose or discuss a general idea of code search.
- Papers that introduce an implementation of a code search/retrieval/recommendation techniques.
- Papers that propose an approach/study targeting specific code search techniques.
- Papers that present a dataset or benchmark especially designed for code search.

Some studies are considered beyond the scope of our work, even though they address code search ideas. These studies adopt code search engines to improve the performance for other research fields, such as code clone detection, which implies that such papers do not improve the field of code search. Furthermore, our survey excludes studies that introduce code search techniques but never retrieve code snippets or related information (i.e., we only cover end-to-end approaches). For instance, some approaches improve user queries by reformulating them (e.g., [106]), but they do not retrieve code snippets. Such approaches are not being considered in the scope of the survey. Moreover, we excluded the papers that explicitly mention a single search base scope (i.e., approaches limited to a single specific software repository), as they are local approaches (e.g., feature location) rather than big code search.

Another criterion we have applied is discarding approaches that focus on locating specific code elements. Even if the same approach is applicable toward searching, detecting, or locating a code element, its use cases differ. There are code clone detection and feature location techniques that closely resemble the code search. Additionally, some code search engines leverage them to evaluate the performance. However, we do not consider code clone detection, as code search as their purpose is generally different from code search; they try to locate the concerned spot within a single project.

We also exclude a set of incomplete work from this survey. The literature of code search also consists of posters that present either work in progress or ideas by different authors at various venues. As most of the posters are later extended in the form of a conference paper or a journal, we do not consider posters in our study as separate ideas or approaches. However, we cover the short papers (i.e., papers that introduce interesting ideas to improve the code search without performing a proper/complete evaluation, and they are usually less than seven pages), as they also contribute to the field.

3.2 Methodology for Literature Identification

To collect the papers across different research avenues that would cover as many papers as feasible, we initiated the keyword search first on popular scientific databases. The databases are listed as follows: ACM Digital Library,[2] IEEE XPlore,[3] DBLP,[4] Springer Link,[5] Wiley Online Library,[6] Elsevier Online Library,[7] and arXiv.[8]

Researchers have used diverse keywords for the fundamentally identical concept (e.g., "code snippet"-"code example" and "retrieval"-"recommendation"). Therefore, we employed keyword combinations for code-related keywords (i.e., source code, code snippet, code fragment, code example) and search related (i.e., retrieve, recommend, and suggest) for the text searching across the repositories and published until 2020. We manually checked all the titles and abstracts to extract papers related to big code search.

To further ensure that we cover all the big code search [72] papers and avoid confusion from missing papers, we conducted snowballing on each paper found by text searching following a well-known guideline [293]. Snowballing is a process that traces citations of papers continuously until there are no missing papers in a domain. The main idea of such a process is to avoid missing related studies that are not discovered by keyword search. This process allowed us to add studies that satisfy our inclusion criteria in Section 3.1. For instance, CodeLikeThis [176] and Strathcona [95–97] are missing from the keyword search but found by such a process.

We observed an increasing trend in the number of code search approaches published in various venues since the inception of the year 2005. Figure 1 illustrates the trend. As this figure reports, the research within the code search domain gradually increased from 2006, reaching its peak in 2019 and 2020. Finally, we ended up with 175 code search approaches. We carefully excluded all duplicate papers (i.e., journal first and short versions of regular papers are not retained) for our taxonomy. Overall, 137 approaches were kept. These trends emphasize the importance of code search, and it is still in growth.

Conferences are popular venues for code search research papers. As shown in Figure 2, conference papers (including full and short versions) account for more than 70% of all papers we have studied. Other venues are academic journals and e-Print archives, which account for 7% and 19% of all papers surveyed, respectively. Note that papers published at arXiv are non-peer reviewed, but this should be included in our survey since it is one of the major industrial R&D publication venues. The International Conference on Software Engineering (ICSE) has been the aim of 20 out of 102 overall approaches that target the conference as a publishing venue. Similarly, the journals *IEEE Transactions on Software Engineering* (TSE), *Empirical Software Engineering* (EMSE), *Journal of Systems and Software* (JSS), and *IEEE Access* are the most target journals in the domain. The public repository that contains all the publications is available at our GitHub repository.[9] Conferences and journals that cater to the code search domain are listed in Table 1 of Appendix A.1.[10]

4 CODE SEARCH ENGINES: GENERAL PROCEDURE

Figure 3 illustrates the typical process under which any general code search is conducted. This process is summarized from the literature, after taking into account all details in the manuscripts. A procedure of code search approach consists of several steps: (1) selecting search type, (2) search base creation, (3) indexing data, (4) formulating input (query), (5) building retrieval model and

[2]https://dl.acm.org/.
[3]https://ieeexplore.ieee.org/.
[4]https://dblp.org/.
[5]https://link.springer.com.
[6]https://onlinelibrary.wiley.com/.
[7]https://elsevier.com/.
[8]https://arxiv.org/.
[9]https://github.com/FalconLK/CodeSearch_Survey.git.
[10]https://github.com/FalconLK/BigCodeSearch/blob/main/Survey_Appendix.pdf.

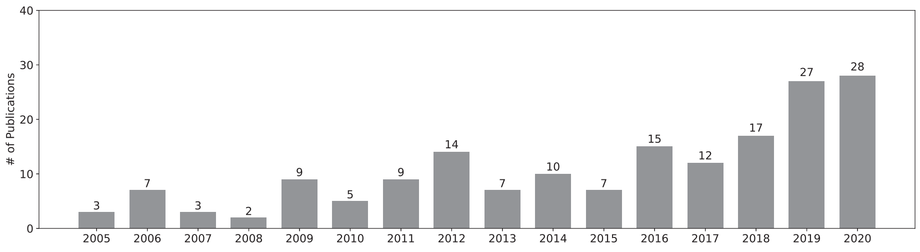


Fig. 1. The number of papers published ranged from 2005 to 2020.

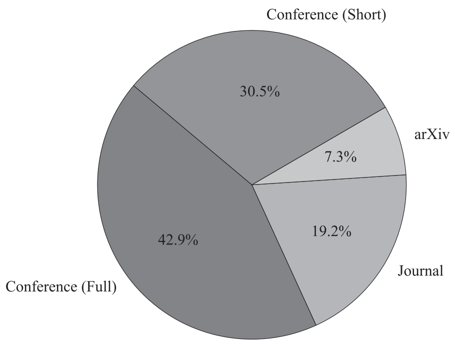


Fig. 2. Publication venue distribution of code search studies.

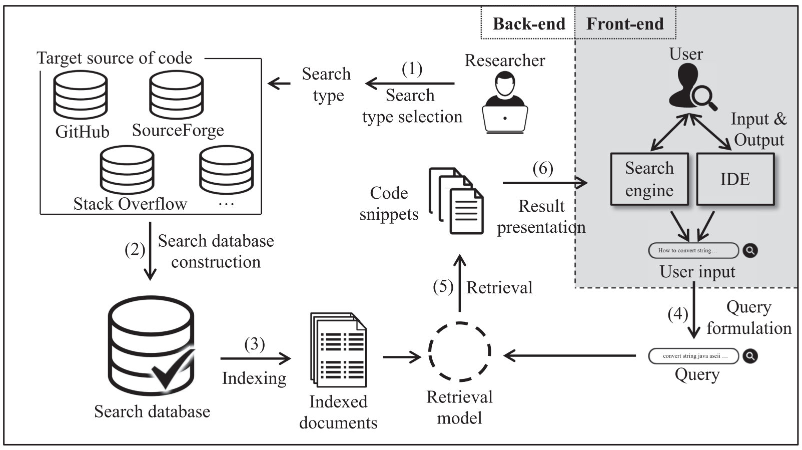


Fig. 3. General code search process.

retrieve, and (6) presenting the results. These steps are therefore guiding the characterization that we will make in our review of the code search literature.

Providing a general procedure can derive a baseline (i.e., how to design a code search approach) for researchers while developers can understand the characteristics of each step and figure out the best approach to apply or use for conducting their own tasks. In other words, this should serve as a practical guide to researchers who are new to this field and developers who are confused about picking a code search approach.

Depending on the design of each step in the procedure, a code search approach can have a significant difference in terms of various performance metrics. For instance, creating a search base with good-quality code snippets can improve the baseline performance, implying that poor code leads to poor results even though the other steps are well designed. Formulating the query with a specific query language may help avoid a well-known VMP to enhance the relevancy of the code snippets. Designing the retrieval model with learning-based approaches significantly improves the performance, although classical models are still much faster to retrieve code snippets.

4.1 Search Target

The first step to designing a code search approach is to decide what kind of search targets it can provide. Developers want to search for various types of code. For example, some need code snippets related to a specific **Application Programming Interface (API)**, whereas others require interface-related code. However, most developers search for available code snippets to address their problems or implement their tasks. These types vary by the desire of target users:

- *General code snippet*: Generally, code search approaches retrieve the most relevant general code snippets rather than other source code types (e.g., API usages, GUI code, or binary code). Some representative examples are as follows: [13, 133, 253].
- *API usage example*: Many software developers leverage the use of APIs that provide standard functionalities toward smooth developments. The developers understand and learn about the API by referring to the API documentation and tutorials (e.g., JDK), looking toward Q&A forums such as Stack Overflow or searching for code examples on super repositories such as GitHub. Some representative examples are as follows: [194, 217].
- *GUI or binary code*: A code is used not just for writing software that provides specific functionalities but also for user interface development that serves as the interface for the user's interaction with underlying software. Some representative examples are as follows: [37, 49, 299]. Moreover, there exists software without its compilable source code; instead, they exist with only compiled binary files. Some representative examples are as follows: [22, 297].

4.2 Search Base Creation

The search base is a repository or dataset where a code search approach can search for and take the code snippets relevant to the user's specification. This is important because the overall performance of a search engine is influenced by the quantity and the quality of data [208]. Considering the importance of the data, we focus on how developers and researchers create their search base in this section and leave the description of the data itself used for code search in Section 6. Many approaches within the literature [133, 166, 220, 253] leverage a super repository such as GitHub [289] for their search base because such repositories are a rich and well-maintained source from the community.

Many studies [41, 66, 195, 211] implied that narrowing the search base allows the search engines to hit more relevant results to the queries. To do so, project metadata such as the programming language, creation date, and popularity are further considered in the code search domain. For some approaches, the researchers [133] consider specific repositories that a certain number of users have starred to avoid toy projects that contribute as noise within the data. Furthermore, some studies (e.g., [109]) leverage supplement information like commit logs (e.g., commit messages) to improve their engines' performance by narrowing the search base or mining them for specific topics. A dataset from Q&A forums where questions are mapped with answers containing code snippets is used in many studies (e.g., [133, 150, 160, 216, 217, 220, 221, 253]).

4.3 Index

Indexing is a process that makes the data access efficient within its search base. Generating and storing an index optimizes the speed and performance of finding the relevant results for a given query. Without such an index, the engine would need to parse every code snippet within the dataset for any given query, which leads to higher time and space complexity. Many studies leverage different indexing techniques to speed up source code retrieval in the code search field:

- *Inverted indexing*: This is an index data structure containing a mapping from content such as words to its locations in a document or a set of documents. Inverted indexing is the most popular indexing technique that many search engines, such as Google, employ in the real world. Many code search engines also take this technique to index their source code snippets in their search base. To apply inverted indexing, a number of researchers [12, 133, 237, 253] utilized Lucene [290], a representative open source search engine library. This library is commonly used to demonstrate many code search techniques (e.g., where they do not need specific indexing techniques).
- *B+ Tree indexing*: This technique is an alternative mechanism to index a sequential set of data elements. A B+ Tree is primarily utilized for implementing dynamic indexing on multiple levels. It stores the data pointers only at the tree's leaf nodes, making the retrieval, addition, and deletion process more accurate and code search faster. Some researchers [22, 71, 143] in code search have relied on this indexing for boosting speed performance.
- *Graph indexing*: Graphs have been used extensively to model the complicated structures and relationships between different entities within programming code. Every query provided as input toward these approaches is transformed in graph structure to retrieve similar or related graphs from the search base. Researchers [124, 265, 297] have leveraged the graph structure and graph indexing techniques for code search engines. The algorithm indexes all the reachable relationships between node labels.
- *ID-based indexing*: This indexing mechanism (also called *file based*) is used to optimize an access to data managed in the form of a single file or access point. This mechanism typically leverages a specifically created index file that stores only the search key's value and a pointer toward the file's storage location. Within code search engines, the use of ID-based indexing means that for each of the potential searchable source code files, an index is stored that can point toward its location. Specific code search engines [34, 120, 274] leverage the ID-based index by assigning a unique searchable ID toward each file, class, or method that can be matched for optimized retrieval.
- *Positional indexing*: Positional indexing is the mechanism that leverages the existence of tokens within a document. Each positional information (e.g., line number, the position of a token within a sentence) of every term is indexed and leveraged with the retrieval techniques.

4.4 Input (Query)

Formulating input to a query is one of the procedures related to the user side (also known as the front-end). In particular, the client-side of code search starts with taking the input from the user. The form of input varies because each user has their own specific tasks. For instance, for developers who want to find a similar code to their code, a code search approach takes a code as the input is needed. Such tasks are mostly related to software maintenance (e.g., refactoring and performance optimization). If a developer has test cases to pass, the perfect-fit code search engine may take the test case as the input. It is crucial to design the input variety in code search according to these situations. We report the inputs and their properties used to design code search approaches:

- *NL query*: Developers often find it easier to formulate the query in natural language for the desired search describing the expected code snippet. An NL query allows a developer to enter terms in any form, either a statement, a question, or a list of keywords associated with programming elements (e.g., class or method name). As many developers do not always have the knowledge about the technical expression required to express their search formats, most code search approaches are designed for NL queries. Community-driven forums for developers such as Stack Overflow [257] and super repositories like GitHub [289] take NL input for the user query.
- *Code fragment*: We define *code fragment* as the source code that the search engine takes as input and *code snippet* as the source code that the search engine retrieves as output; code-to-code search engines leverage the fragments and snippets as query and results, respectively. The code-to-code approach is beneficial for research directions in code transplantation, code diversity, and patch recommendation to find essential ingredients for their techniques. One of the popular commercial engines, Krugle [134], has its snippet-based search in the advanced option, whereas Searchcode [244] is capable of taking input as both NL and code fragment in its main interface.
- *Query language*: A query language is specifically designed to model the structure and properties (e.g., loops, method calls, and exception handling). Modeling such properties can express more complicated search patterns such as "find all code examples that call the ABC method in a loop and handle an exception of type abcException." In the case of typical text-based input, this cannot be accurately expressed [239, 255].
- *Binary code*: In some cases, developers may need to search for code similar to functions within an existing binary file. Developers mostly use the use case of binary code search to search for critical vulnerabilities or find code snippets that implement the given binary input file's functions. Researchers have proposed approaches (i.e., [37, 49, 300]) that take a binary file as input and perform code search to retrieve code snippets that implement semantically similar functions.
- *Test case*: A test case is executed and reaches a state of pass if the output from the code and the provided output matches are considered to have failed. Test cases are used heavily in software development and form the basis of test-driven development[21]. In many cases, **Test-Driven Code Search (TDCS)** approaches take either test cases or a set of keywords that can define a required test case. Test cases are beneficial, as they provide instant feedback about the suitability of a particular code result.
- *Class/Interface type*: Similar to test cases, **Interface-Driven Code Search (IDCS)** approaches leverage the interface types existing within the input snippet. These interface or class types are similar to existing interfaces within a function such as *string f(string)* in Java. In the example, the types (as input of a code search) are "string." Some researchers [95–97, 144] have leveraged the use of interface or class types for matching, which reduces the potential search space and provides effective code search engines.
- *Software specification*: Software specification is what the user specifies as their requirements and conditions toward the search. For example, a user may wish to retrieve code with only a limited set of parameters or asynchronous-only API code. Researchers [173, 263] have leveraged software specifications to specify the security constraints (i.e., method pre- and post-conditions) for retrieving results aligned with the user's needs.

4.5 Retrieval Model

A retrieval model predicts and explains what a user desires given the input (i.e., it matches the query and retrieves the source code). The correctness of the model's predictions can be validated in various experiments. Moreover, different retrieval models usually categorize code search approaches as they take the position of the principle behind them. The retrieval phase of code search generally consists of matching text, computation of matrix, or vector similarity measure depending on a specific technique:

- *Textual similarity*: In code search, the textual similarity consists of matching tokens, keywords, or even a sequence of tokens between the user query and source code from the search base. As for the standard models, the **Boolean Model (BM)** [136] and **Vector Space Model (VSM)** [238] are famous classical information retrieval models adopted many times by code search engines. Query and the target code files are conceived as sets of terms in the BM model, and the retrieval is based on whether or not the code files include the terms from the query. It has an explicit limitation that at least one of the query terms must be present on the document side when an OR connective is used. However, VSM represents the document (source code files) as the vector of identifiers such as method name, class name, or tokens. Generally, the comparison between such two vectors is derived using the cosine similarity. It does not support structural queries that consist of complex AND/OR relations. Therefore, researchers revised the VSM integrating with BM to address the limitations by weighting both query and document terms.
- *Graph similarity*: As the tokens in source code are not just text, one might need a different form of representation to capture the semantics in the code. Graph representation [6] is an optimal way to model the different source code elements and leverage them for the code search. The code search approaches that leverage graph-based retrieval do not explicitly accept input in a “graph” form; instead, it accepts a typical query in the form of NL or code fragments. The query is transformed into a suitable graphical representation such as an abstract syntax tree before being passed as input for the underlying search technique. The type of graphical representation selected for a particular search engine depends on the context of the targeted input. For instance, a binary code search engine is likely to transform the binary input into a control flow graph; a code search engine targeting object-oriented programming language would use a dataflow graph. In the form of similarity computation, this comparison consists of leveraging graph traversing techniques for optimized results.
- *Matrix computation*: Matrix computation is an elaborate method used as a retrieval technique in some code search approaches [241, 291]. The result of the computation provides a characterized correspondence between the matrix elements. Many applications have adopted these matrix computations because of their scalability that retains the results’ predictive accuracy. Similarly, code search employs the matrix computation for measuring the co-occurrences of specific features (e.g., terms) within the source code.
- *Embedding vector similarity*: Recently, code search approaches (e.g., [33, 80, 160, 233]) using neural networks have been a trend. Common across such approaches is the idea of embedding user queries and code into vectors (i.e., an embedding refers to a real-valued vector representation [132]). The computation of the distance between the embedding vectors is the correlation that forms the basis of retrieving the suitable candidates. As code search approaches based on neural networks show that they can adequately learn the features of both the query and code from a large dataset, it helps to address the issues from the previous techniques.
- *Execution trace*: A source code in binary format is hard to read and understand [147]. An efficient way toward understanding what a binary file is performing is to trace its execution flow, known as the execution trace [31]. This execution flow provides information on the behavior of a program, such as the logical sequence of execution, and changes in the

data variables. The similarity between the execution traces of two syntactically different programs provides information on how similar (or dissimilar) the two binary programs are.

- *Clone detection*: Code search is another way to identify similar or identical source code snippets. Generally, code-to-code approaches sometimes leverage code clone detection techniques in the middle of their retrieval process to narrow down the search space by clustering similar source code [4, 127, 162, 188]. Clone detection techniques are also directly used to determine the lexical similarity between the query code and candidate snippets [218].
- *Type link*: Type link [10] is a methodology that resolves a given function name by referencing the function and attempting to link it with its canonical form for the same type. For example, a class extending another class may have complex inheritance and nested types from its parent class. Such a type of class or code context requires a different approach when searching within them.
- *Solver*: Solver or SMT solvers [261] (short for satisfiability modulo theories) are instances represented as a formula in first-order logic. These formulas represent functions and symbols that follow specifications in the form of input and output of a program. For any given query, represented in the input and output of a program, the solver finds programs for which these specifications and constraints are satisfied. The satisfied candidates thus form part of the results presented to the user.

4.6 Implementation

The final step of the code search procedure delivers and presents the results in an appropriate platform. As each user has different requirements and feasibility, code search approaches should be provided on different platforms such as independent code search engines, an **Integrated Development Environment (IDE)**, or presented as an idea in a research paper. Independent code search engines can be either local or online to interact with developers with the underlying techniques, whereas others are implemented as a plugin of different IDEs allowing them to leverage the search within the development process. Additionally, some are presented just as an idea form as potential work in code search:

- *Search engine*: A search engine catered explicitly toward the code search is the popular means of presenting the search approach results. These search engines are either working online (accessible over the internet) or offline (where the engine is deployed locally over the dataset chosen by the user).
- *IDE extension*: Rather than providing the code search in a search engine, some researchers have attempted to provide code search facilities within the IDE such as Eclipse as a plugin. The majority of the code search implementation in the form of IDE extensions take as input either a pure NL query or a code fragment to output a relevant code snippet. The usage of the IDE plugin allows the code search to leverage the written code tokens as given input for the approach to find similar or potential code examples.
- *Idea*: Among all the different code search approaches, not all propose an evaluation or implementation; instead, there are ideas that can be incorporated into a potential code search implementation. Such approaches are classified as "idea" that proposes an approach without any evaluation or implementation details described. Such ideas are valuable contributions within the field since some of them are later implemented as part of a code search engine or an IDE plugin.

5 TAXONOMY OF CODE SEARCH TECHNIQUES

This section presents a taxonomy of code search tools and techniques. The taxonomy provides an overview of different techniques and their categories in the code search literature. Since there are

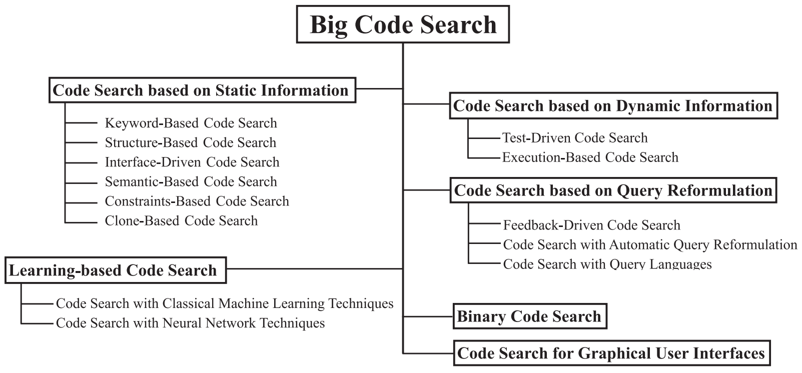

Big Code Search
Code Search based on Static Information
Keyword-Based Code Search
Structure-Based Code Search
Interface-Driven Code Search
Semantic-Based Code Search
Constraints-Based Code Search
Clone-Based Code Search
Learning-based Code Search
Code Search with Classical Machine Learning Techniques
Code Search with Neural Network Techniques
Code Search based on Dynamic Information
Test-Driven Code Search
Execution-Based Code Search
Code Search based on Query Reformulation
Feedback-Driven Code Search
Code Search with Automatic Query Reformulation
Code Search with Query Languages
Binary Code Search
Code Search for Graphical User Interfaces

Fig. 4. Taxonomy for big code search techniques.

hundreds of code search techniques, it can be challenging to figure out their characteristics and details. A clear taxonomy can help to understand the techniques with a concise view on their key differences.

Based on the general procedure of code search, we have identified that code search approaches are broadly differentiable based on the kind of information that is used to drive the search (i.e., the query). Beyond that, the method under which the query is matched is also a key differentiator. Overall, we have defined six top-level categories to classify code search techniques. These categories represent the most common and popular ideas used in many code search techniques, whereas other details of each individual technique would be different from others. The overall taxonomy is presented in Figure 4, demonstrating the orders and sub-categories for better understanding. When a specific technique leverages ideas of multiple categories, we classify the technique as one category, which is the principal idea of the technique. Note that all the big tables for our taxonomy are located in Appendix A.[11]

5.1 Code Search Based on Static Information

Code search can leverage static information such as text in source code, which is similar to general-purpose search engines [74, 185]. For example, a code search tool can scan source code files and make an index of class, name, variable names, and parameter types. Recent code search techniques with static information consider more information, such as code comments; they generate code summaries to match against the query. We have identified six sub-categories of code search engines utilizing static information: keyword-based, structure-based, semantic-based, interface-based, constraints-based, and clone-based approaches. The details for each technique are presented in Table 2 of Appendix A.2.

5.1.1 Keyword-Based Code Search. Many early code search engines rely on general text retrieval approaches; they treat source code as either plain or structured text. To that end, those approaches build an index of the textual information (i.e., keywords). Given a user query, the approaches try to compute a token-level similarity between the query and index. This type of code search is categorized into **Keyword-Based Code Search (KBCS)** in this study. The earliest known KBCS approaches [65, 67] were based on retrieving code snippets without considering any code-specific information (structure, sequence, etc.). Although these approaches cannot be classified as big code search approaches, it is meaningful to consider in the survey for historical

[11] https://github.com/FalconLK/BigCodeSearch/blob/main/Survey_Appendix.pdf.

trends. Later, KBCS techniques often leverage additional information available after parsing abstract syntax trees [251]; when building an index, those techniques annotate keywords with code entity information such as package, interface, class, method, constructor, field, and initializer. In addition to tokens in source code, it is able to use tokens in other sources such as API documentation [77, 78, 180, 183, 253], which are linked to source code as code search inevitably suffers from the VMP [253]. KBCS techniques are also applied to code search in Q&A posts [311], code-to-code search [191, 193, 272], and code summarization [114].

5.1.2 Structure-Based Code Search. In contrast to tokens and keywords used in general search engines, source code often has specific structures such as call graphs, code trees, developer activities, and common API usage patterns. It might take some time to extract such information, but it is much less than dynamic information (see Section 5.2) as the information is essentially static.

Graphical information is one of the most common structures used for code search. For example, function [45, 166, 171, 181, 184, 236, 274] or API call graphs [36, 151] are common types of structure information used in code search engines. This information helps to expand the search space, followed by the graphs, compared with KBCS in which the search space is limited to code entities containing tokens in a given query. The system dependency graph [212] and program expression graphs (E-PEGs) [212] are used for code search as well. Computing tree similarity is another way of utilizing static structural information, as source code is fundamentally a tree form after parsing. Code search can leverage tree similarity based on hashing [4] or intermediate models [55, 235, 275, 276]. Hierarchical structures are also used for code search, such as code blocks [100] and class hierarchy [163], as they can expand the search space.

There have been code search approaches combining structural information and other entity types together. Sourcerer [12] provides a ranked list of code entities, and the ranking is computed by keywords, structures, and graphs available in source code. Another approach [192] collects developer activities recorded in development tools and incorporates them with structural information such as classes, types, and parameters available in source code to compute a degree-of-interest model, which improves the code search performance. Barbosa et al. [17, 18] proposed an approach to searching for exception handling structures. This approach combines matching keywords in a user query with exception handling blocks.

Leveraging API usage patterns is another line of structure-based code search. By mining API documents (or other types of documentation) and usage patterns together, code search approaches [10, 140, 173, 186, 194, 198, 206, 214, 296] figure out relationships between them. For a given user query (e.g., what a user wants to implement, described in a natural language), an approach in this category first searches for similar API descriptions to the query and then retrieves API usage cases corresponding to the descriptions.

5.1.3 Interface-Driven Code Search. Functions (also known as *methods* in some languages, e.g., Java) are often a target unit of code search. IDCS approaches take an information of a function signature (i.e., interface) [312]. The information may include function name, parameters (with type information), and return type. Based on the information, IDCS can reduce the search space (e.g., by restricting the parameter and return type of a function to *int*) and generally improve the performance of code search. Strathcona [95–97] is an example of IDCS. Another approach [144] extends Sourcerer [13] to implement IDCS.

5.1.4 Semantic-Based Code Search. Keywords and structural information often represent functional aspects, whereas other types of information convey semantic aspects of source code. The semantic information includes part-of-speech, topics, and source code types. These data can identify latent semantics of queries and code entities corresponding to the semantic. Thus, code search tools based on semantic information often extract the information both from user queries and source code.

Several code search engines have utilized diverse semantic information available in source code. For example, it is common to use semantic information collected from natural language processing. Part-of-speech [159, 279] and phrasal concepts [92] after parsing entity names (e.g., function signature, variable names, and types) are often leveraged to extract semantics of queries and source code. Since the meaning of the collected words cannot be discovered directly from the source code, those code search engines often adopt word models (e.g., the Software Word Usage model) [90]. Topics are another standard semantic information used in code search. Topic modeling can identify common topic information from a bunch of documents. Several recent code search tools [9, 182, 278, 315] extract topics by using latent Dirichlet allocation [26], latent system analysis [53], feature identification approach [60], or the N-gram model [87] to identify code entities highly relevant to a given query even though they share very few keywords. Another type of semantic information used in code search is production and test code relationships. Test Recommender [209] identifies the relationships by collecting changesets in versioning histories. The changesets are used for ranking search results.

5.1.5 Constraints-Based Code Search. Some code search approaches [43, 121, 260–263] attempt to use example-based query models to conceptualize what users want to search for. The approaches encode source code snippets as constraints. Each query (e.g., an input/output example) is transformed into constraints and sent as input to an SMT solver [51] along with the encodings of the source code repositories. The SMT solver identifies the code from the repository that meets the I/O example, which forms the results.

5.1.6 Clone-Based Code Search. Code clone detection [231] and code search share several common ideas and techniques. Thus, many code search approaches [127] employ techniques used in code clone detection. For example, CodeNuance [162] directly uses CCFInderX [123], a code clone detector, to identify similar code (i.e., code clones) and build an exploration graph based on non-duplicate methods. Other approaches [188, 218] in this category are designed in a similar way to support code search. Sometimes, code clone detection techniques help to overcome the limitations with non-compilable code or play a compensatory role of other similarity measures. It can say that the approaches in this category leverages static information since code clones used in code search are mostly Type-1, Type-2, and rarely Type-3 clones rather than Type-4 clones [231]. Note that the Type-4 clones are semantic clones, whereas other clone types are defined by lexical similarity.

5.2 Code Search Based on Dynamic Information

Code search engines driven by dynamic information focus on behaviors captured after running programs, whereas search engines based on static information utilize data collected without executing the programs. To extract dynamic information, it is necessary to run a program with concrete inputs (e.g., from test cases), which may cause an additional cost. Despite the cost, dynamic information can improve the performance of code search. For example, the dynamic information can assist developers who lack the expertise of the desired code [71] and non-native speakers of the language in which the repository is based [143], assure faster retrieval of code snippets [137], and provide more guarantee of the correctness of the behavior of retrieved code snippets [137]. Table 3 of Appendix A.2 illustrates the details of code search techniques based on dynamic information.

5.2.1 Test-Driven Code Search. What if a user can specify input/output example pairs (i.e., test cases) when searching for a certain source code snippet? TCDS is one of the dynamic approaches in code search that takes a test case(s) and identifies the appropriate code snippets that pass the given test case. The use of test cases helps (1) define the behavior of the desired functionality to be searched and (2) test whether the search results are suitable in the local context [137]. Typically, TDCS consists of the following procedures: define test cases, feed the test cases to the search engine, and run the test cases on each search result (e.g., code snippet or function), given by the search engine, to validate whether it satisfies the search requirements specified by the test cases.

There have been several TDCS techniques. CodeGenie [137–139, 141] is one of the earliest technique that leverages Sourcerer [12], a well-known code search engine, as it carries out KBCS with keywords extracted from test cases. Then, CodeGenie runs the test cases for each search result. This technique has been extended by using *thesaurus-based tag clouds* [143] as well to mitigate the VMP. Code Conjurer [111, 117] attempts to implement a more proactive code search based on TDCS than CodeGenie. EQMINER [120] generates random test input generation by using symbolic [148] or concolic execution [135], then checks the abstract memory states [131] to compute function similarity based on execution outputs. Additionally, code search engines (e.g., S6 [28, 29, 223–226], HUNTER [287], and that of Kessel and Atkinson [129]) augment TDCS ideas to enhance search performance.

5.2.2 Execution-Based Code Search. Code search engines can leverage dynamic information extracted after running a program. Program source code is not just text, and it is meaningful once it is executed. Fundamentally, the users of code search engines want to locate programs that carry out the meaning specified in the queries rather than those containing the same tokens. DyCLINK [265] is one of the examples of code search engines utilizing execution traces. This tool takes a code snippet as a query and constructs its **Dynamic Dependency Graph (DDG)** that captures behavior at the instruction level. The query's DDG is compared with DDGs of other programs to identify similar code snippets. CodeHint [71] is another code search engine leveraging execution traces. This engine exploits the Java runtime capabilities by executing the debugger at runtime to collect data. This data is used for matching the information specified in a user query.

5.3 Code Search Based on Query Reformulation

The users of code search engines often try several different queries for a single search campaign when their first queries are not effective searching for necessary code snippets. Some code search engines proactively help or simulate the trials of different consecutive queries. This process is often called *query reformulation.* Search approaches in this category reason about more precise query strings based on initial user queries. A subset of the approaches leverage interactions between search engines and users to improve search queries (i.e., feedback-driven code search). Another subset of the approaches automatically reformulate the initial query and provide the search results (automatic query reformulation). Other approaches in this category use domain-specific query language to enhance the query reformulation process. The related particulars are exhibited in Table 4 of Appendix A.2.

5.3.1 Feedback-Driven Code Search. Code search tools in this category assume that the first user queries are highly likely to be incomplete. Instead, these tools provide a series of feedback (or clarification questions) to improve the user queries. The feedback includes adding, removing, or editing their initial queries [15, 30, 94, 247, 258]. This process is iterated until the developer is satisfied with the results. Note that developers often do not have a clear idea of what they are working with [50, 250]. Thus, the feedback loop can help the users complete the search queries [50, 63, 167, 188]. Multiple researchers have investigated an approach toward building feedback-driven (i.e., iterative) approaches based on such observation [1, 39, 85, 91, 93, 150, 164, 176, 177, 230, 239, 255, 264, 284].

5.3.2 Code Search with Automatic Query Reformulation. A query plays one of the most critical roles for the code search engines [93, 210], and many researchers have worked toward improving the search quality by reformulating the queries automatically (i.e., without the user's intervention). Query reformulation is conducted because the VMP [88] (multiple words for the same topic), polysemy (one word with multiple meanings), and the general words in the query complicate the code search engines. A query can be reduced, expanded (augmented) [2, 34, 61, 105, 107–109, 122, 133, 142, 160, 169, 201, 216, 220, 221, 240–242, 253, 254, 294, 302, 304, 313, 317], or even entirely alternated by their properties [35, 84, 153, 165, 172, 217, 298, 303].

5.3.3 Code Search with Query Languages. When creating search queries, the users can benefit from a domain-specific language for code search. This type of language is called *query language*. A query language defines the structure of a query and the property of a token in the query. A user can annotate additional information of each element in a query. For example, SnipMatch [292] incorporates crowd-sourced source code and introduces a simple markup language used for indexing and matching relevant code snippets. Another example is the dependence query language [283, 286]. This language allows users to formulate the queries by describing dependence properties on top of textual properties. AutoQuery [282] alleviates the burden of users when creating a query based on a query language.

5.4 Learning-Based Code Search

Leveraging machine learning techniques is popular in building a code search engine. Code search is fundamentally a prediction task for a given input (search query) to provide a set of code snippets as output so that machine learning techniques fit the task. Additionally, code search techniques benefit from the recent advancement of deep learning. Nevertheless, machine learning techniques such as support vector machine [47] are useful for code search campaigns as well. The encapsulated details are shown in Table 5 of Appendix A.2.

5.4.1 Code Search with Classical Machine Learning Techniques. Code search can be modeled as supervised (e.g., classification of a given query string into a set of files) or unsupervised (e.g., clustering of similar source code files) learning tasks. Thus, many classical machine learning techniques are applied to code search approaches to improve the quality of search results. The approaches leverage different machine learning techniques, such as collaborative filtering [291], clustering [81, 102, 268], classification [119, 124, 197, 199], graph embedding [318], and Bayesian networks [190].

5.4.2 Code Search with Neural Network Techniques. Deep neural networks opened a new research direction for code search as they do the same for other computer science topics. Like the code search approaches utilizing classical machine learning techniques, many recent approaches define code search as supervised and unsupervised learning tasks especially for deep learning. Those approaches use diverse deep learning techniques such as embedding (for word, sentence, or paragraph) [3, 33, 59, 80, 89, 154, 200, 233, 305], recurrent neural networks [80, 154, 160], convolutional neural networks [52, 104, 149, 246, 281], long short-term memory [79, 80, 86, 103, 112, 115, 146, 154, 228, 243, 246, 267, 280, 305, 307, 316], auto-encoders [42], transformers [285, 309], feed-forward neural networks [68, 104], graph neural networks [40, 155, 280], and generative adversarial networks [314].

5.5 Binary Code Search

Static information is not limited to source code. Some code search approaches [37, 49, 130, 152, 299, 300] address binary code as the search space instead of source code. These approaches are helpful when available resources have only compiled binary files. The approaches often create additional

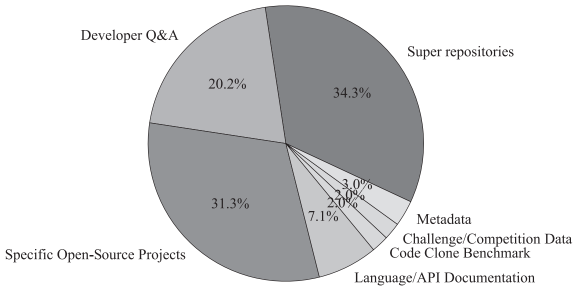


Fig. 5. Statistics of dataset types used in code search studies.

indices after decompiling the binary code or retrieving control flow graphs from the binary files. The detailed information per each technique is demonstrated in Table 6 of Appendix A.2.

5.6 Code Search for Graphical User Interfaces

Researchers have proposed different approaches that aim to simplify and automate exploring and building **Graphical User Interfaces (GUIs)** by lettering users' reuse similar GUIs within data repositories. The retrieved result is provided in the form of an interface for users to interact with the code to replicate them. Some approaches [22, 297] take input sketches from the user, identify its features, and apply various code search techniques to retrieve similar designs from the search space. Another approach [227] empowers the search by seeking users to "draw" the required interface design within the search engine, potentially with additional context such as keywords relevant toward the search. Each characteristics are displayed in Table 7 of Appendix A.2.

6 DATASETS AND BENCHMARKS FOR CODE SEARCH

The choice of datasets in which performing the search has a high impact on the performance of code search techniques. Additionally, those play a crucial role in evaluating the techniques from the research perspectives. For instance, using a readily available dataset allows reproducibility of the approach, and the larger the dataset, the better the coverage of the code search approach. This phenomenon makes the dataset a crucial component when designing a code search engine, as it directly impacts its performance. The dataset may also contain additional information such as Q&A posts of open developer forums and software metadata, whereas benchmarks can incorporate other features such as fixed queries and metrics. This section explains the source code dataset, non-code dataset, and benchmarks used within the code search domain to create search bases or evaluate the code search engines. Figure 5 illustrates the statistics of dataset types used in the code search studies.

6.1 Specific Open Source Projects

In an earlier time before the rise of open source software, only a limited number of projects and their source codes were actively maintained and accessible to the public. As a result, researchers intending to apply code search ideas had to leverage these sparsely available open source projects. These projects ranged from different communities such as Apache HTTP Server [273] and the Linux operating system [73]. Portfolio [181, 184] and Coogle [235] are representative examples of approaches that utilize specific open source projects to demonstrate their results.

6.2 Data Sources from Super Repositories

There has been widespread use and adaptation of open source software in the past decade. This has created thriving open source software development communities that leverage collaborative coding systems and are made centrally available for everyone to use. Similarly, researchers within the code search have adopted the data sources that form a large set of different open source projects when testing their approaches [12, 274].

The leveraged data source, known as the dataset, is in a file archive, where most of the source code files are extracted from different software projects. These repositories are often accessible online on code hosting services such as `GitHub` [289] or `SourceForge` [256]. The hosting services are built upon version control systems such as `Git` [156]. Software repositories are a rich source of code snippets created and curated by developers around the globe. Furthermore, the curated source code snippets in the form of datasets provide opportunities to investigate and research new ways for code search techniques.

Many of commercial code search approaches [44, 134, 204, 244] and research prototypes [133, 169, 177, 178, 186, 189, 214, 215, 218, 221, 253, 262, 288, 304] have utilized source code files collected from the code repository platform `Github` as their search base. `Github` is one of the largest super-repositories built on top of the `Git` version control system [16]. It is the most superior hosting service [222], with more than 100 million repositories hosted as of January 2020. `Github` also hosts huge and popular projects such as Homebrew [179] (a package manager) or Django (a web framework) [98].

`SourceForge` is another super repository used for constructing code search datasets. It hosts more than 500,000 projects. Similar to `GHTorrent` for `GitHub`, the `FLOSSMole` [99] project supports creating a dataset [187] from `SourceForge`. Exemplar [180], Lukins et al. [168], and Hsu and Lin [101] crawled and utilized the data from `SourceForge`.

For Android-centric code search approaches, there exists a website named *F-droid*.[12] It is a repository of Android applications of various sizes and categories. The dataset[13] contains meta-data (name, description, and version), the source code of each major version, and its most recent "apk" file. Some code search approaches [119, 201] leveraged such datasets for Android code search.

Although these super repositories have been used frequently to construct various code search approaches, their use has several disadvantages. For example, users tend to upload (commit and push) files unrelated to source code, such as security-related tokens or documents containing personal information. These can affect the quality of the search base. Such accidental exposure of information can have dire consequences (e.g., disclosing personal information to the public) [271] even through the code search engines. Another disadvantage of super repositories for research is that the quality of the projects and developers within such community is hard to validate. A significant amount of time needs to be dedicated to curating good candidate projects.

6.3 Other Data Sources

Other resources have been leveraged for different approaches toward code search problems. Resources such as the developer Q&A forums, benchmarks, metadata, and API documents are applied for some approaches. Such additional resources have demonstrated an increase in the code search techniques' efficiency and accuracy.

6.3.1 Software Developer Q&A Forum. Q&A forums are community-driven platforms that allow users to share knowledge with other users who participate in them. Q&A forums offer social mechanisms to evaluate and improve the quality of both the question and answer that implicitly leads to brevity in questions and qualitative answers, potentially with source code snippets [133]. The posts within such forms tend to include code snippets within the question and its answer. It makes them

[12] https://f-droid.org.
[13] https://gitlab.com/fdroid/fdroiddata.

ideal for forming the part of the dataset for some code search approaches. Many of the code search approaches [9, 116, 127, 133, 144, 151, 169, 175, 186, 201, 202, 218, 253, 259, 262, 263, 278, 279, 311] have utilized the developer Q&A forum called `Stack Overflow`. `Stack Overflow` is the largest Q&A forum that mainly contains questions and answers on programming-related topics [76]. Answers from `Stack Overflow` often are an alternative explanation for corresponding official product documentation wherein the documentation is either insufficient, does not exist, or lacks in-depth information [19]. Furthermore, a significant portion of answers includes code snippets that demonstrate the solution for the corresponding programming problem.

Recently, a large and systematically mined dataset using Stack Overflow was proposed, and learning-based code search approaches [52, 86, 103, 281, 314] leveraged these in their evaluation. The CoNaLa dataset by Yin et al. [310] consists of two parts, a manually curated parallel corpus of training and test examples as well as systematically mined pair examples. Unlike CoNaLa, StaQC [306] is specifically intended for code search (i.e., not for code generation or summarization), and it consists of 148,000 systematically mined Python and SQL question-code pair datasets.

Even though such data from Q&A forums have advantages for code search, they are not primarily designed for code reuse or the search for particular code snippets [19]. Thus, not all the posts include source code, as some are mere questions and answers either with incomplete code or devoid of any code. For example, there are multiple instances where the code is written with ellipses (i.e., "…") [253]. This implies that some people prefer to express natural language expressions that make it hard to leverage the data. Furthermore, pervasive unqualified names [266] and ambiguity in enclosing class names of method calls [48] for code snippets can affect the accuracy of the overall source code approach.

6.3.2 Language/API Documentation. Some code search studies [38, 132, 175, 180] leveraged well-known API documentation. Generally, developers refer to the API documentation for examples [151] of a concerned project where such documents are the official source of information, such as`JavaDoc` [205]. These documentations are known and followed by all vendors for consistencies usually written by multiple people [54].

6.3.3 Code Clone Benchmark. Several researchers have used a benchmark designed explicitly for code clone detection to evaluate their code search approaches. For example, `BigCloneBench` [8, 269, 270] is a benchmark that clearly distinguishes between the actual Java clones mined from a cross-project dataset, `IJaDataset` 2.0 [7, 8]. FaCoY [133], a semantic-based code search approach, uses `BigCloneBench` in a different perspective (i.e., code clone detection using a code search approach). Its evaluation consists partially of `BigCloneBench`'s data as its search base and is evaluated against the state-of-the-art clone detection tools (e.g., [237]).

6.3.4 Challenge/Competition Data. Coding competitions are events where participants try to code a program based on specific problems. For example, Google Code Jam [75] is an annual online coding competition hosted by Google. Each submission for a defined problem is expected to perform semantically similar even though they may syntactically be different. Researchers have leveraged the problems and the different solutions to evaluate the code search engines. For example, DyCLINK [265] and FaCoY [133] applied the code written for Code Jam to identify code relatives at the granularity of methods. The latest challenge known as CodeSearchNet [113] aims to encourage researchers and practitioners to study and propose new approaches toward code search. Several learning-based approaches [79, 155, 267] leveraged this challenge dataset.

6.3.5 Metadata. Software project metadata includes information such as the project name, owner, and description [23, 24, 46, 62, 83, 234]. Code search studies [27, 72, 82, 128, 133, 177, 181, 253] have leveraged project metadata by claiming that metadata can play an important role in designing and implementing code search approaches.

6.4 Benchmarks for Code Search

Common benchmarks are necessary to compare the performance of different code search approaches. Many studies on code search collected their datasets from various sources and incorporated several evaluation metrics. Approaching this direction leads to inconsistencies for fair comparison because their search base has different snippets, and different metrics may indicate different results. Furthermore, different queries for the same task make the retrieved results vary. These problems motivate researchers to build benchmarks for code search using common search bases and metrics or to have the same queries and answer code snippets. Therefore, the ideal benchmarks should provide the dataset, metrics, and sample results that lead the users to a fair comparison in the code search evaluation phase.

Several code search benchmarks have been proposed recently. For instance, Li et al. [145] introduced a benchmark in the field of code search by providing the results of their neural code search approaches [33, 233]. The benchmark consists of a dataset as a natural language query (from Stack Overflow) and code snippet pairs about Android software development (from GitHub). They further employed two neural code search engines named *NCS* [233] and *UNIF* [33] to build the benchmark providing the results. Another benchmark, CosBench [301], combines a search base (from GitHub), query and answer pairs (from Stack Overflow), and a set of metrics. To show the usefulness of the benchmark, they employed four information retrieval based [165, 169, 201, 290] and two learning-based [80, 308] approaches to compare in the evaluation. A recent deep learning approach [149] has leveraged this benchmark to study user query and source code interactions.

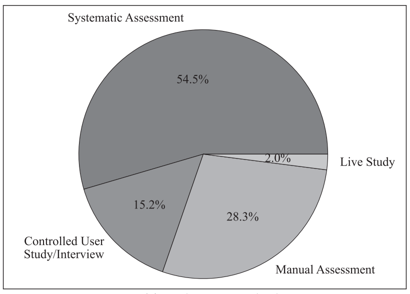
There is another notable benchmark dataset: Project CodeNet [213]. This dataset stands out from other similar datasets, as it is exceptionally large with code samples written in more than 50 programming languages. The code samples in Project CodeNet are also extensively annotated, providing information such as code size, memory footprint, and CPU runtime. The dataset also includes benchmarks that potentially can be leveraged to evaluate new code search approaches built upon AI techniques.

7 EVALUATION

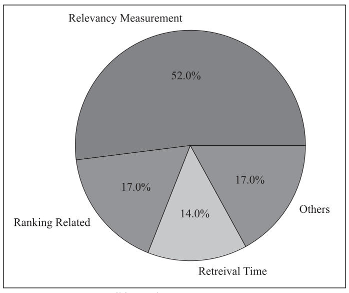
Evaluating the performance of the approaches provides an overview of how efficient and accurate the code search engine is compared to others or over a particular dataset. In particular, code search approaches should be evaluated in several different aspects, such as the accuracy of the relevancy between the query and retrieved source code, the time consumed for the retrieval, or if the results pass specific test cases. To quantify such aspects, researchers leverage several ways (e.g., systematic assessment or live study) to measure the satisfactory performance of the approaches. This section investigates the different evaluation methods and the metrics used within the code search approaches.

7.1 Evaluation Methods

The evaluation methods are essential when disseminating the results of a particular approach. The research community has invested considerable efforts to find appropriate ways to measure the various performances. Evaluation of classical search engines has been limited toward manual relevancy checks by domain experts [172]. Similarly, code search engines' performance was manually measured (e.g., [96, 236, 274] in the early stage until researchers found systematic ways to remove the subjective bias. Since then, systematic evaluation methods (e.g., comparison against the state-of-the-art approaches and models) were used to provide a more precise evaluation of the



(a) Evaluation methods



(b) Evaluation metrics

Fig. 6. Distribution of evaluation methods and metrics used in code search studies.

proposed approach [33]. Moreover, other methods, such as controlled user study/interview, provide opinions and insights from experts, developers, or a more significant portion of people capable of providing adequate evaluation [93, 279]. Recently, with the development forum's high growth, the live study method is being applied to leverage crowd knowledge [253]. Figure 6(a) presents how many approaches take certain evaluation methods, and Table 8 of Appendix A.3 demonstrates the details of each approach.

7.1.1 Manual Assessment. One of the practical evaluation methods, known as manual internal assessment, requires the researchers' manual efforts. In this method, the researchers manually validate the results returned by the code search engines, determining their relevance to the user query and assigning a score for each of them. However, this method is known to be inefficient and expensive [160] (i.e., the evaluation's manual process is quite time consuming). Furthermore, as the evaluation subjects are usually the authors, it may have a subjective bias.

7.1.2 Systematic Assessment. To overcome the potential issues of bias in the manual assessment, researchers undertook a systematic evaluation of the results. The essential systematic evaluation consists of a comparison against other state-of-the-art approaches and models. For example, researchers (e.g., [34, 169, 217]) utilize the dataset (e.g., queries and search space) and configure similar environments (e.g., computing power, parameters) used in other state of the art. Yet, a significant challenge when conducting the systematic assessment lies in the lack of a replication package of the state of the art [157]. Another challenge is the non-agnostic nature of search engines (i.e., specific target programming languages or different software paradigms). Similarly, the learning-based approaches [33, 80, 233] utilize specific learning models or a combination of models from different approaches to bring new insights into the learning algorithm's characteristics.

7.1.3 Controlled User Study/Interview. Practical evaluation of a new approach can be challenging, as there might not be state-of-the-art techniques available to compare. To address this issue, researchers [141, 169, 177, 282] conducted a controlled user study/interview session that provides researchers with opinions and insights on the approach from real-world users who are likely to be domain experts. As there is no standard or defined process/rule on the number or occupation of involved users, type of queries, or topics, researchers have tried to involve as many users as possible to understand the generic opinions. This type of evaluation can be subjective, but the results are acceptable if the evaluation method is designed concretely with a solid rationale and the users are professionals within the software domain. Large industrial entities like Microsoft tend to leverage this method to evaluate their approaches [113].

7.1.4 Live Study. The goal of a live study is to evaluate the approach in the real world where any user (professional or otherwise) would use and evaluate the results. These users generally are in the form of utilizing the users of popular developer forums. For example, Sirres et al. [253] took the questions from Stack Overflow as their testing queries and posted the code snippets from their code search engine CoCaBu as the answer. The public (i.e., developers from Stack Overflow) judges the questions and answers by voting and commenting in the system. Researchers utilize this method on various platforms (e.g., Stack Overflow and GitHub).

7.2 Evaluation Metrics

There exist several metrics to assess code search engines. We split them into four dimensions: overall performance, ranking, retrieval time, and others (e.g., simple counting, statistical metrics, and user satisfaction). These metrics are adopted for various purposes, such as to measure how the results of a code search engine are relevant for a given query, how fast it can perform, or how much query range it can cover. Figure 6(b) shows the distribution of the metrics for assessing code search approaches. The overall performance metrics are primarily based on utilizing the confusion matrix. Table 9 of Appendix A.3 illustrates the list of performance metrics mainly used in the code search domain.

Precision [11] concerns multiple codes relevant to a query, whereas **Mean Average Precision (MAP)** measures the average. Recall [11] quantifies the number of correct predictions made out of all positive examples in the dataset. F-measure [277] is the harmonic mean of the precision and recall. Similarly, **Normalized Discounted Cumulative Gain (NDCG)** [118] measures the performance considering the recommended candidates' order (weight-based). Accuracy [11] is the proportion of correct predictions (both true positives and true negatives) among the total number of cases. SuccessRate [11] measures the percentage of queries for which more than one correct result could exist in the results. A ROC curve [64] is created by plotting the true-positive rate against the false-positive rate with various thresholds. Furthermore, two approaches leveraged false-positive and false-negative rates to check the errors of the search engine.

The ranking metrics primarily consider the location of the correct answers among the results. Several approaches are evaluated by checking the rank of the correct answers manually. FRrank (also known as best hit rank) is the rank of the first hit result in the result list [214]. Rank of the first correct (RFC) and **Expected Reciprocal Rank (ERR)** are measured to check if the user should review a specific number of results. **Mean Reciprocal Rank (MRR)** [11] is the average of the reciprocal ranks of results for a set of queries. The reciprocal rank of a query is the inverse of the rank of the first hit result. Differently, one paper [22] measured the quality of a candidate ranking against users' opinions to prove the ranking performance. Table 10 of Appendix A.3 shows the ranking metrics utilized within the code search field.

The metrics used to evaluate code search approaches vary based on the context of the approach. Table 11 of Appendix A.3 introduces all the details of the other metrics that have been used within the code search. Other metrics include retrieval time; several statistical tests like mean squared error; and relevancy score-based and measuring metrics such as Kendall's correlation coefficient, Spearman's correlation coefficient, Pearson's correlation coefficient, and the Mann-Whitney U test. Furthermore, user studies in code search utilized user satisfaction metrics such as experience score and mouse clicks. Other quality capturing metrics such as the METEOR score and BLEU-4, rate of passing the test cases, and query coverage, as well as metrics based on an external library (specifically, `org.eclipse.compare-plug-in`), are also shown to be used in particular code search approaches (e.g., [40, 223, 235]).

8 OPEN ISSUES AND POTENTIAL RESEARCH DIRECTIONS

Researchers have put tremendous effort into the field to avoid serving a poor-quality code search engine that can drag developers out, as documented in many published studies. Despite these proposed approaches and advancements, the following open issues we discovered may impede rapid and efficient software development.

Specific Tasks vs. Code Search Techniques. Given a wide variety of existing approaches to each code search procedure, developers may not be sure which one is the best code search technique for their specific tasks. For instance, developers who are new to their field may need a code search engine that can adequately reformulate their queries because of their lack of knowledge. Some developers need a feedback-driven code search engine to address multiple and complex requirements. As another example, code-to-code search approaches can be utilized to refactor the existing code by finding either syntactically or semantically similar code snippets. These task-characteristic pairs should be analyzed, and as a result, the best way to address a specific task remains an open issue.

Vocabulary Mismatch Problem. One of the crucial issues that affect information retrieval, in general, is the VMP. During our review process, we found out that the VMP applies to most of the code search techniques as well. Specifically, techniques investigated in multiple sections (i.e., Sections 5.1, 5.3, and 5.4) focus on addressing this specific problem. The VMP states that the likelihood of two people choosing the same keyword for a familiar concept is only between 10% and 15% [69]. Recently, researchers have focused on addressing this with learning-based techniques, especially to lower the gap between NL and code. However, we observed that the recent results of the approaches indicate that the VMP within the code search needs further efforts on tokenization and translation of NL to code to improve the retrieval performance.

Benchmarks. A trustworthy conclusion for the code search will be drawn with a high-quality benchmark [12], yet there is a glaring lack of such a benchmark for code search engines. Our investigation on the dataset (Section 6) for evaluations of each technique explicitly revealed that there exist many different and individually collected datasets. However, these recent benchmarks also have open issues: (1) these are not validated yet by code search researchers, (2) they focus on learning-based approaches (i.e., classical and learning-based approaches are different), and (3) they still need standard and concrete metrics to test various approaches. Furthermore, other considerations, such as different NL queries for a specific set of tasks, should provide a similar set of results. Every user has a unique way of describing a problem, especially when representing in NL. Thus, generating a set of techniques that would consider different NL representations for the same set of questions would provide a more normalized set of results.

Extensibility. Software development consists of using multiple different programming languages for developing one full product. Consequently, code search approaches do not always prove to be a good solution because of their limitation on specific programming languages. We substantiate this in our taxonomy, and Tables 2 through 7 of Appendix A.2[14] show this information as the last column. The extensibility of code search approaches toward all viable programming languages is an important issue that is yet to be addressed in the domain. Note that the literature reviewed in this survey always retrieves/targets one language at a time. Applying a particular search approach to different programming languages (i.e., multi-language) would provide more usefulness and convenience.

Usability. We observed that specific code search approaches such as binary code search are limited in their usability in a real-world scenario while reviewing binary code search approaches

[14]https://github.com/FalconLK/BigCodeSearch/blob/main/Survey_Appendix.pdf.

in Section 5.5. Although finding similar binary code is useful for specific domains such as vulnerability detection, code search is rather severely limited. This is mainly because a majority of the approaches rely on an essential assumption that the binary code is not obfuscated. Furthermore, the rise in compiler-level optimization or even hardware-specific optimization would prevent disassembly techniques from limiting the searching capabilities of code search approaches.

Replicability. Although this survey encompasses many approaches that support code search, most do not have publicly available replication packages. This poses an obstacle for developers who need to apply such an approach. A shared replication package helps developers in two significant ways, as (1) a time-efficient way to deploy and test the approach and (2) a reference implementation to check for errors and issues. Reimplementation of an approach is a time-consuming and error-prone task, even if the approach is well explained. Therefore, sharing source code can improve efficiency in the field of code search.

Multi-Modal Query Supported Code Search. Code search techniques can utilize multiple query models. For example, such techniques could allow users to input different formats of queries, such as free-form text, query language, and input-output examples, at the same time. The more detailed information the user provides, the higher the probability that the tool could give accurate recommendations; however, this direction is not addressed in previous studies according to our taxonomy illustrated in Section 5. Thus, we need a new code search system that can support multi-modal queries and may bring substantial improvements.

9 CONCLUSION

Through the comprehensive study undertaken in this survey, we expect it to serve as a thorough introduction for newcomers, helping them familiarize themselves in the field. At the same time, the practitioners can extend their understanding toward identifying techniques based on the context of their exploration. Moreover, existing code search researchers can identify the different contributions and advances made within the different categories. To this end, we undertook a procedure-driven systematic literature review process to identify 137 code search approaches that form the bases of our operational taxonomy. The taxonomy classifies all the approaches to permit a fair comparison and identifies potential future research areas that can be explored. Furthermore, the survey shed light on the existing open issues within the code search, such as the lack of a standard benchmark that provides a fair evaluation between various code search engines. Additionally, the survey established the directions that researchers can tackle further in code search. Finally, our procedure-driven systematic literature review provideed an analysis on each phase, and it should deliver insights for the overall field.

A APPENDIX

This section includes supplementary information for our survey.

A.1 Paper Collection and Review Schema

This section contains Figure 7 showing the cumulative number of papers published ranging from 2005 to 2020 and Table 1, which presents the publication venues of code search studies.

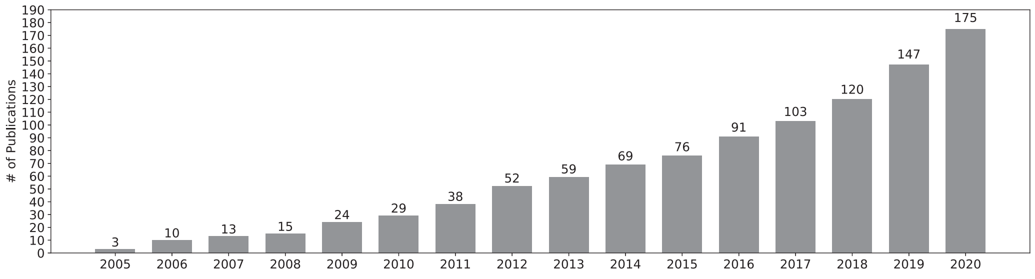


Fig. 7. The cumulative number of papers published ranged from 2005 to 2020.

Table 1. Publication Venues of Code Search Studies

Category	Abbreviation	Full Name	Count
Conference	ICSE	International Conference on Software Engineering	20
	ASE	Automated Software Engineering	13
	MSR	International Conference on Mining Software Repositories	8
	SUITE	Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation	5
	PLDI	ACM SIGPLAN Conference on Programming Language Design and Implementation	4
	OOPSLA	Object-Oriented Programming, Systems, Languages, and Applications	4
	RSSE	International Workshop on Recommendation Systems for Software Engineering	4
	SANER	IEEE International Conference on Software Analysis, Evolution, and Reengineering	3
	COMPSAC	Annual Computer Software and Applications Conference	3
	CSMR-WCRE	IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering	3
	ESEC/FSE	European Software Engineering Conference and Symposium on the Foundations of Software Engineering	2
	SAC	ACM Symposium on Applied Computing	2
	WWW	The World Wide Web Conference	2
	SBES	Brazilian Symposium on Software Engineering	2
	SCAM	International Working Conference on Source Code Analysis and Manipulation	2
	VL/HCC	Visual Languages and Human-Centric Computing	2
	ISSTA	International Symposium on Software Testing and Analysis	1
	MAPL	ACM SIGPLAN International Workshop on Machine Learning and Programming Languages	1
	FASE	International Conference on Fundamental Approaches to Software Engineering	1
	RecSys	ACM Conference on Recommender Systems	1
	ACIIDS	Intelligent Information and Database Systems	1
	UIST	ACM Symposium on User Interface Software and Technology	1
	WEH	International Workshop on Exception Handling	1
	SBCARS	Brazilian Symposium on Software Components, Architectures, and Reuse	1
	ACL	Annual Meeting of the Association for Computational Linguistics	1
	ICoICT	International Conference on Information and Communication Technology	1
	WSDM	ACM International Conference on Web Search and Data Mining	1
	ICCIT	International Conference on Computer and Information Technology	1
	CCS	ACM SIGSAC Conference on Computer and Communications Security	1
	RCoSE	International Workshop on Rapid Continuous Software Engineering	1
	Programming	International Conference on the Art, Science, and Engineering of Programming	1
	Internetware	Asia-Pacific Symposium on Internetware	1
	MOBILESoft	International Conference on Mobile Software Engineering and Systems	1
	IWSC	International Workshop on Software Clones	1
	SERVICES	IEEE World Congress on Services	1
	ASC	ACM Southeast Conference	1
	ICSEW	International Conference on Software Engineering Workshops	1
	IJCNN	International Joint Conference on Neural Networks	1
	CIRCLE	CEUR Workshop	1
		Subtotal (Conference)	102
Journal	TSE	*IEEE Transactions on Software Engineering*	3
	EMSE	*Empirical Software Engineering*	3
	JSS	*Journal of Systems and Software*	3
	IEEE Access	*IEEE Access*	3
	SPE	*Software: Practice and Experience*	3
	TOSEM	*ACM Transactions on Software Engineering and Methodology*	2
	ASE Journal	*Automated Software Engineering Journal*	2
	IST	*Information and Software Technology*	2
	TSC	*IEEE Transactions on Services Computing*	2
	SCIS	*Science China Information Sciences*	2
	JIFS	*Journal of Intelligent & Fuzzy Systems*	1
	ISF	*Information Systems Frontiers*	1
	JPCS	*Journal of Physics: Conference Series*	1
	PACMPL	*Proceedings of the ACM on Programming Languages*	1
	IEEE Software	*IEEE Software*	1
	KBS	*Knowledge-Based Systems*	1
	KIES	*International Journal of Knowledge-Based and Intelligent Engineering Systems*	1
	WUJNS	*Wuhan University Journal of Natural Sciences*	1
	JIT	*Journal of Internet Technology*	1
	SEKE	*International Journal of Software Engineering and Knowledge Engineering*	1
		Subtotal (Journal)	35
		Total	137

A.2 Taxonomy of Code Search Techniques

This section presents detailed information reflecting the proposed taxonomy. Each of Tables 2 through 7 classifies all the investigated techniques based on their characteristics.

Table 2. Dissection of Code Search Techniques Based on Static Information

Category	Approach	Output		Dataset					Indexing				Input							Retrieval					Presentation			Languages
		General Code	API Usage	Specific Open Source Projects	Super Repositories	Developer Q&A	Language/API Documentation	Others	Inverted	Database (B+ Tree)	Graph Index	File Prefix	Natural Language	Code Fragment	Input/Ouput	Software Specification	Query Language	Binary	Class/Interface Type	Textual Similarity	Graph Similarity	Solver	Type Links	Embedding Vector Similarity	Search Engine	Idea	IDE Extension	
Keyword based	Jsearch [251]	✓									✓			✓						✓							✓	Java
	CoCaBu [253]	✓			✓	✓			✓				✓							✓					✓			Java
	Murakami et al. [192]	✓		✓						✓				✓						✓						✓		Java
	McMillan et al. [183]		✓	✓					✓				✓							✓					✓			Java
	Exemplar [77, 78, 180]	✓			✓				✓				✓							✓					✓			Java
	Example Overflow [311]	✓				✓			✓				✓							✓					✓			Java
	Selene [191, 272]	✓		✓					✓					✓						✓					✓		✓	Java
	SoCeR [114]	✓		✓					✓				✓											✓	✓			Python
Structure based	Aroma [166]	✓			✓					✓				✓							✓			✓	✓			Java, JavaScript, Python
	Prospector [171]	✓					✓												✓		✓						✓	Java
	Portfolio [45, 181, 184]	✓		✓					✓				✓							✓	✓				✓			C, C++, Java
	XSnippet [236]	✓					✓			✓									✓		✓						✓	Java
	PARSEWeb [274]	✓				✓	✓					✓							✓		✓						✓	Java
	Chan et al. [36]		✓	✓							✓		✓							✓	✓				✓			Java
	RACS [151]		✓	✓		✓	✓			✓			✓							✓	✓				✓			JavaScript
	YOGO [212]	✓				✓				✓				✓							✓				✓			Python, Java
	Akhin et al. [4]	✓		✓							✓			✓							✓					✓		
	Coogle [235]	✓		✓										✓							✓						✓	Java
	Mendel [163]	✓		✓										✓							✓				✓			
	Hsu and Lin. [100]	✓			✓				✓				✓							✓					✓			Java
	Sourcerer [12]	✓			✓				✓				✓							✓					✓			Java
	Murakami et al. [192]	✓		✓						✓				✓						✓						✓		Java
	Barbosa et al. [17, 18]	✓			✓					✓				✓						✓					✓			Java
	AUSearch [10]		✓		✓												✓						✓		✓			Java
	MAPO [296]	✓	✓										✓							✓					✓			Java
	SUSHI [198]	✓						✓						✓				✓		✓				✓	✓			Android
	PropER-Doc [173]		✓	✓												✓							✓				✓	Java
	PRIME [186]	✓	✓	✓									✓							✓					✓			Java
	LibFinder [206]	✓			✓									✓										✓			✓	Java
	SWIM [214]	✓	✓		✓								✓							✓					✓			C#
	APIREC [194]	✓	✓											✓						✓	✓						✓	Java
	Lee et al. [140]		✓		✓								✓							✓							✓	Java
Interface driven	Strathcona [95–97]									✓				✓					✓								✓	
	Lemos et al. [144]	✓			✓					✓			✓						✓	✓					✓			Java
Semantic based	ANNE [279]	✓				✓							✓							✓					✓		✓	Java
	CodeMatcher [159]	✓			✓				✓				✓							✓					✓			
	Hill et al. [92]	✓		✓									✓							✓					✓			Java
	JECO [9]	✓				✓							✓							✓					✓			Java
	Vinayakarao [278]	✓				✓				✓				✓						✓						✓		Java
	McMillan et al. [182]	✓			✓					✓			✓							✓					✓			Java, C#
	Lancer [315]	✓			✓					✓				✓						✓	✓						✓	Java
	Test Recommender [209]	✓		✓						✓				✓						✓							✓	Java
Constraints based	Satsy [260–262]	✓		✓					✓					✓	✓							✓			✓	✓		Java, C
	Extended Satsy [263]	✓		✓					✓		✓				✓	✓					✓	✓			✓			Java
	Quebio [121]	✓			✓									✓	✓						✓	✓			✓			Java
	Extended Quebio [43]	✓			✓	✓							✓		✓					✓		✓			✓			Java
Clone based	Keivanloo et al. [127]	✓			✓				✓											✓					✓			
	CodeNuance [162]	✓			✓				✓											✓								
	Rahman and Roy [218]	✓			✓				✓					✓							✓						✓	Java
	MUSE [188]		✓	✓																	✓						✓	Java

Table 3. Dissection of Code Search Techniques Based on Dynamic Information

Category	Approach	Output		Dataset		Index			Input						Retrieval				Presentation			Language
		General Code	Test Case/Test Code	Specific Open Source Projects	Super Repositories	Database (B+ Tree)	Graph Index	File Prefix	Natural Language	Code Fragment	Query Language	Test Case	Class/Interface Type	Software Specification	Textual Similarity	Graph Similarity	Test Case/Tested Input	Linear Programming	Search Engine	Idea	IDE Extension	
Test driven	CodeGenie [137–139, 141]	✓				✓				✓						✓					✓	Java
	Lemos et al. [143]	✓			✓	✓				✓					✓						✓	Java
	Code Conjurer [111, 117]		✓			✓				✓					✓						✓	Java
	EQMINER [120]		✓	✓				✓		✓						✓	✓		✓			C
	S6 [28, 29, 223–226]					✓			✓					✓		✓			✓			Java
	Kessel and Atkinson [129]	✓			✓	✓				✓							✓			✓		
	HUNTER [287]	✓			✓	✓					✓	✓						✓	✓			Java
Execution based	CodeHint [71]	✓		✓		✓								✓	✓						✓	Java
	DyCLINK [265]	✓		✓			✓			✓						✓					✓	MySQL

Table 4. Dissection of Code Search Techniques Based on Query Reformulation

Category	Approach	Output		Database						Index				Input			Retrieval				Presentation			Language
		General Code	API Usage	Specific Open Source Projects	Super Repositories	Developer Q&A or Tutorial	Language/API Documentation	Code Clone	Challenge/Competition	Inverted	Database (B+ Tree)	ID-based	Positional	Natural Language	Code Fragment	Query Language	Textual Similarity	Matrix Computation	Graph Similarity	Embedding Vector Similarity	Search Engine	Idea	IDE Extension	
Feedback driven	Mica [264]		✓	✓			✓			✓				✓			✓				✓			Java
	SAS [1]		✓		✓									✓			✓				✓			Java
	Conquer [230]	✓		✓										✓			✓						✓	Java
	Extended Conquer [93]	✓		✓										✓			✓						✓	Java
	Wang et al. [284]	✓		✓										✓			✓				✓			C,C++
	CodeExchange [177]	✓			✓					✓				✓			✓				✓			Java
	CodeLikeThis [176]	✓			✓					✓				✓			✓				✓			Java
	INQRES [164]	✓		✓										✓			✓				✓			Java
	Cosoch [150]	✓				✓								✓			✓				✓			Java
	Contextual Search [91]	✓			✓									✓			✓						✓	Java
	Refoqus [85]	✓		✓						✓				✓			✓						✓	Java, C++
	ALICE [255]	✓		✓										✓		✓				✓			✓	Java
	SNIPR [239]													✓		✓	✓					✓		
	DeepAPIRec [39]		✓		✓										✓					✓	✓			Java
AQR	QECK [201]	✓			✓	✓				✓				✓			✓				✓			Android
	Yang and Tan [302]	✓		✓										✓			✓				✓			Java
	Durão et al. [61]	✓		✓								✓		✓			✓						✓	Java
	NLP2CODE [34]	✓	✓			✓								✓			✓						✓	Java
	SCP [254]	✓		✓									✓	✓			✓				✓			Java, C, C++
	QExpandator [241]	✓		✓										✓				✓			✓			Java
	FWSMF [242]	✓		✓						✓				✓			✓				✓			Java
	CodeX [240]	✓		✓						✓				✓			✓					✓		N/A
	SnippetGen [105, 294, 304]	✓			✓					✓				✓			✓				✓			C#
	CodeGenie 2.0 [142]	✓			✓					✓				✓			✓						✓	Java
	CoCaBu [253]	✓			✓	✓				✓				✓			✓				✓			Java
	FaCoY [133]	✓			✓	✓		✓	✓	✓					✓		✓				✓			Java
	CodeHow [169]	✓			✓					✓				✓			✓						✓	C#
	RACK [220, 221]	✓			✓	✓	✓							✓			✓						✓	Java
	NLP2API [216]	✓				✓				✓				✓						✓	✓			Java
	NQE [160]	✓			✓	✓				✓				✓						✓	✓			Android
	SENSORY [2]	✓			✓							✓		✓			✓				✓			Java
	QESR [122]	✓			✓		✓			✓				✓			✓				✓			Android, Java
	QECC [109]	✓			✓					✓				✓			✓				✓		✓	Java
	GKSR [107]	✓			✓					✓				✓			✓				✓			C#, Java, Android
	QESC [108, 317]	✓			✓		✓			✓				✓			✓				✓			Java
QL	AutoQuery [282]	✓		✓							✓				✓				✓		✓			C, C++
	SnipMatch [292]	✓		✓										✓			✓						✓	Java

AQR, automatic query reformulation; QL, query language.

Table 5. Dissection of Learning-Based Code Search Techniques

Category	Approach	Output		Dataset							Index		Input		Retrieval				Presentation			Language
		General Code	API Usage	Specific Open Source Projects	Super Repositories	Developer Q&A	Language/API Documentation	Code Clone	Challenge/Competition	Existing Benchmark	Inverted	Graph Index	Natural Language	Code Fragment	Textual Similarity	Graph Similarity	Matrix Computation	Embedding Vector Similarity	Search Engine	Idea	IDE Extension	
	MMMF [291]	✓		✓										✓			✓		✓			Java
	Surisetty [268]	✓		✓							✓		✓					✓		✓		
	ROSF [119]	✓		✓							✓		✓		✓			✓	✓			Java
	Source Forager [124]	✓			✓							✓		✓				✓			✓	C/C++
Machine learning	Zou et al. [318]	✓				✓	✓					✓	✓					✓	✓			Java
	CodeKernel [81]		✓	✓			✓					✓		✓				✓	✓			Java
	CODEC [190]	✓			✓									✓	✓	✓		✓	✓			Java
	ExAssist [197, 199]	✓		✓										✓	✓				✓		✓	Java
	CodeMF [102]	✓				✓							✓		✓				✓			C#, SQL
	Nguyen et al. [200]		✓	✓			✓						✓		✓			✓	✓			Java
	CODEnn [80]	✓			✓								✓					✓	✓			Java
	MP-CAT [86]	✓				✓							✓					✓	✓			Python
	CSDA [228]	✓			✓								✓					✓	✓			Java
	CARLCS-CNN [246]	✓			✓								✓					✓	✓			Java
	BVAE [42]	✓				✓							✓					✓	✓			C#, SQL
	SCOR [3]	✓			✓						✓		✓					✓	✓			Java
	SLAMPA [316]	✓						✓						✓				✓	✓			Java
	SCS [112]	✓		✓									✓					✓	✓			Python
	NCS [233]	✓			✓	✓					✓		✓					✓	✓			Android
	UNIF [33]	✓			✓	✓					✓		✓					✓	✓			Android
	Fujiwara et al. [68]	✓						✓						✓				✓	✓			Java
	MMAN [280]	✓			✓								✓					✓	✓			C
	AdaCS [154]	✓			✓								✓					✓	✓			Java
	CoaCor [305]	✓				✓							✓					✓	✓			Python, C#
	CODE-NN [115]	✓				✓							✓		✓				✓			C#, SQL
Neural network	Ye et al. [307]	✓				✓							✓					✓	✓			Python, SQL
	TranS^3 [285]	✓			✓								✓					✓	✓			Python
	Yin et al. [309]	✓				✓							✓					✓	✓			Python, SQL
	CDRL [104]	✓			✓								✓					✓	✓			Java
	Schumacher et al. [243]	✓			✓								✓					✓	✓			Python
	HECS [146]	✓				✓							✓					✓	✓			Python, C#
	MSR [59]	✓		✓									✓		✓			✓	✓			Java
	PSCS [267]	✓							✓				✓					✓	✓			Python, JavaScript, Java
	Heyman and Cutsem [89]	✓				✓							✓					✓	✓			Python
	COIL [149]	✓			✓					✓			✓					✓	✓			Java
	CoNCRA [52]	✓				✓							✓					✓	✓			Python, SQL
	COSEA [281]	✓				✓							✓					✓	✓			Python, SQL
	DGMS [155]	✓			✓	✓			✓	✓			✓					✓	✓			Java, Python
	APIRec-CST [40]		✓		✓		✓						✓						✓			Java
	Zhao and Sun [314]	✓				✓							✓					✓	✓			Python, SQL
	CRaDLe [79]	✓							✓				✓					✓	✓			Python
	NJACS [103]	✓				✓							✓					✓	✓			C#, SQL, Java, Python

Table 6. Dissection of Binary Code Search Techniques

Approach	Output	Dataset			Index		Input	Retrieval				Presentation	Language
	Binary Code	Specific Open Source Projects	Super Repositories	Others	Inverted	Graph Index	Binary	Textual Similarity	Graph Similarity	Execution Trace	Embedding Vector Similarity	Search Engine	
Tracelets [49]	✓		✓		✓		✓		✓	✓		✓	Binary (x86)
Rendezvous [130]	✓			✓	✓	✓	✓	✓		✓		✓	C, C++
BINGO [37]	✓			✓			✓		✓	✓		✓	Binary C
BINGO-E [300]	✓			✓			✓		✓	✓		✓	Binary C
Gemini [299]	✓	✓				✓	✓				✓	✓	Binary C

Table 7. Dissection of Code Search for GUIs

Approach	Input	Dataset		Index			Input			Retrieval		Presentation		Language
	Sketches/GUI	Specific Open Source Projects	Super Repositories	Inverted	Database (B+ Tree)	Graph Index	Sketch File	Natural Language	Code Fragment	Textual Similarity	Graph Similarity	Search Engine	IDE Extension	
GUIFetch [22]	✓		✓		✓		✓	✓		✓		✓		Java GUI
SUSIE [227]	✓		✓					✓	✓	✓		✓		Java GUI
Xie et al. [297]	✓	✓		✓		✓	✓			✓	✓		✓	Java GUI

A.3 Evaluation

This section demonstrates various evaluation methods and metrics used in the field of code search per each approach, which are presented in Tables 8 through 11.

Table 8. Evaluation Methods Used in Code Search Techniques

Evaluation Method	Techniques
Manual assessment	Prospector [171], Strathcona [95–97], Jsearch [251], XSnippet [236], Coogle [235], PARSEWeb [274], Contextual Search [91], McMillan et al. [183], Wang et al. [286], Selene [191, 272], PropER-Doc [173], Exemplar [77, 78, 180], Example Overflow [311], Mentor [170], Barbosa et al. [17, 18], Chan et al. [36], Yang and Tan [302], PRIME [186], SCP [254], CodeHint [71], CodeGenie 2.0 [142], Tracelets [49], Keivanloo et al. [127], JECO [9], Vinayakarao [278], RACS [151], CODE-NN [115], SWIM [214], BINGO [37], QualBoa [57], FWSMF [242], Source Forager [124], LibFinder [206], Gemini [299], CoCaBu [253], FaCoY [133], SLAMPA [316], Quebio [121], GUIFetch [22], CodeNuance [162], ALICE [255], ExAssist [197, 199], BINGO-E [300], Xie et al. [297], Huang et al. [110], SoCeR [114], YOGO [212], CodeMatcher [159], CODEC [190], Extended Quebio [43], AUSearch [10]
Systematic assessment	Strathcona [95–97], XSnippet [236], PARSEWeb [274], Example Overflow [311], Chan et al. [36], Yang and Tan [302], SCP [254], Keivanloo et al. [127], CODE-NN [115], LibFinder [206], Gemini [299], CoCaBu [253], FaCoY [133], SLAMPA [316], BINGO-E [300], Xie et al. [297], CodeMatcher [159], CODEC [190], Extended Quebio [43], CodeGenie [137–139, 141], SNIFF [38], S6 [223–225], MMMF [291], Hill et al. [92], Hsu and Lin [100], Portfolio [45, 181, 184], Wang et al. [283], McMillan et al. [182], Satsy [260–262], Rahman and Roy [218], Lemos et al. [144], CodeHow [169], DyCLINK [265], QECK [201], QExpandator [241], ROSF [119], Extended Satsy [263], APIREC [194], Niu et al. [203], CodeLikeThis [176], NLP2CODE [34], SnippetGen [105, 294, 304], RACK [220, 221], INQRES [164], NLP2API [216, 217], QECC (InstaRec) [109], Zou et al. [318], CODEnn [80], BVAE [42], NCS [233], Lee et al. [140], Lancer [315], Aroma [166], Cosoch [150], NQE [160], SENSORY [2], QESR [122], GKSR [107], QESC [108, 317], CodeKernel [81], SCOR [3], UNIF [33], MMAN [280], CoaCor [305], Yin et al. [309], CodeMF [102], CSDA [228], CARLCS-CNN [246], AdaCS [154], Ye et al. [307], TranS^3 [285], CDRL [104], HECS [146], MSR [59], PSCS [267], COIL [149], COSEA [281], DGMS [155], APIRec-CST [40], Zhao and Sun [314], CRaDLe [79], NJACS [103], CodeGenie 2.0 [142], RACS [151], Source Forager [124], ALICE [255], Sourcerer [12], Durão et al. [61], APPROX [20], Lemos et al. [143], Rendezvous [130], Extended Conquer [93], ANNE [279], Nguyen et al. [200], DeepAPIRec [39], Li et al. [152], PCR [196], MPCAT [86], Schumacher et al. [243], Heyman and Cutson [89], CoNCRA [52]
Controlled user study/interview	XSnippet [236], LibFinder [206], CoCaBu [253], CodeGenie [137–139, 141], Portfolio [45, 181, 184], McMillan et al. [182], CodeHow [169], QECK [201], QExpandator [241], Niu et al. [203], CodeLikeThis [176], NLP2CODE [34], INQRES [164], CodeKernel [81], Exemplar [77, 78, 180], GUIFetch [22], CodeNuance [162], ALICE [255], SnipMatch [292], Wang et al. [284], Test Recommender [209], CodeExchange [177], MUSE [188], AutoQuery [282], HUNTER [287]
Live study	CoCaBu [253], SnipMatch [292], CodeExchange [177], TranS^3 [285]

Table 9. Relevancy Metrics Used for Evaluating Code Search Techniques

Metric	Techniques
Precision	Durão et al. [61], MMMF [291], Hill et al. [92], Portfolio [45, 181, 184], Exemplar [77, 78, 180], Mentor [170], Chan et al. [36], McMillan et al. [182], Yang and Tan [302], Satsy [260–262], Rendezvous [130], Keivanloo et al. [127], Rahman and Roy [218], JECO [9], Vinayakarao [278], AutoQuery [282], FWSMF [242], Zou et al. [318], SLAMPA [316], ALICE [255], CodeKernel [81], SoCeR [114], AUSearch [10]
Precision@k	Satsy [260–262], SCP [254], QECK [201], QExpandator [241], ROSF [119], Extended Satsy [263], BINGO [37], SnippetGen [105, 294, 304], LibFinder [206], CoCaBu [253], FaCoY [133], QECC (InstaRec) [109], CODEnn [80], Lee et al. [140], SENSORY [2], SCOR [3], CodeMatcher [159], CODEC [190], CDRL [104], HECS [146], MSR [59], COSEA [281]
MAP	SCP [254], Extended Satsy [263], QualBoa [57], Source Forager [124], SCOR [3], Zhao and Sun [314]
MAP@k	Rahman and Roy [218], RACK [220, 221], NLP2API [216, 217], QESR [122], GKSR [107], QESC [108, 317], COIL [149]
Recall	Strathcona [95–97], Sourcerer [12], Durão et al. [61], MMMF [291], Hill et al. [92], Selene [191, 272], Mentor [170], Chan et al. [36], Yang and Tan [302], Satsy [260–262], Rendezvous [130], CodeGenie 2.0 [142], Rahman and Roy [218], JECO [9], Vinayakarao [278], Lemos et al. [144], AutoQuery [282], FWSMF [242], FaCoY [133], Zou et al. [318], SLAMPA [316], ALICE [255], CodeKernel [81]
Recall@k	SCP [254], LibFinder [206], NLP2API [216, 217], QECC (InstaRec) [109], CODEnn [80], Aroma [166], SCOR [3], CodeMatcher [159], CodeMF [102], MPCAT [86], CARLCS-CNN [246], HECS [146], Heyman and Cutson [89], CRaDLe [79], NJACS [103]
Accuracy	Tracelets [49], Rahman and Roy [218], DeepAPIRec [39], NQE [160], Schumacher et al. [243]
Accuracy@k	SWIM [214], APIREC [194], Nguyen et al. [200], LibFinder [206], INQRES [164], Yin et al. [309], PCR [196], CoNCRA [52], APIRec-CST [40]
SuccessRate	Jsearch [251], HUNTER [287], CodeNuance [162], PSCS [267]
SuccessRate@k	RACS [151], RACK [220, 221], NLP2API [216, 217], CODEnn [80], SLAMPA [316], Lancer [315], MMAN [280], Li et al. [152], CODEC [190], CSDA [228], AdaCS [154], TranS^3 [285], CDRL [104], COIL [149], DGMS [155]
NDCG	Exemplar [77, 78, 180], Wang et al. [284], Extended Satsy [263], SnippetGen [105, 294, 304], Ye et al. [307], TranS^3 [285], COSEA [281], Zhao and Sun [314]
NDCG@k	Wang et al. [284], QECK [201], ROSF [119], Niu et al. [203], RACK [220, 221], QECC (InstaRec) [109], Cosoch [150], SENSORY [2], QESR [122], GKSR [107], QESC [108, 317], Li et al. [152], MSR [59]
F-measure	Durão et al. [61], Contextual Search [91], MMMF [291], Hill et al. [92], Chan et al. [36], Rendezvous [130], AutoQuery [282], ALICE [255], CodeKernel [81]
ROC curve	Tracelets [49], Gemini [299], Quebio [121]
Sensitivity	EQMINER [120], Extended Satsy [263]

Table 10. Ranking Metrics Used for Evaluating Code Search Techniques

Metric	Techniques
MRR	Strathcona [95–97], CodeHow [169], CODE-NN [115], Extended Satsy [263], CoCaBu [253], Zou et al. [318], CODEnn [80], BVAE [42], SLAMPA [316], Lancer [315], Cosoch [150], NQE [160], UNIF [33], MMAN [280], CoaCor [305], Yin et al. [309], CodeMatcher [159], CODEC [190], CodeMF [102], MP-CAT [86], CARLCS-CNN [246], AdaCS [154], Ye et al. [307], TranS^3 [285], CDRL [104], HECS [146], PSCS [267], Heyman and Cutson [89], CoNCRA [52], COSEA [281], DGMS [155], APIRec-CST [40], CRaDLe [79], NJACS [103]
MRR@k	RACK [220, 221], NLP2API [216, 217], CSDA [228], COIL [149]
FRank	PARSEWeb [274], XSnippet [236], SNIFF [38], McMillan et al. [183], Example Overflow [311], SWIM [214], BINGO [37], Quebio [121], CODEC [190], CodeMF [102], CSDA [228], HECS [146]
FRank@k	UNIF [33]
Simple rank	Prospector [171], PRIME [186], BINGO [37], Huang et al. [110]
ERR	Niu et al. [203]
Significance and cohesiveness	PropER-Doc [173], GUIFetch [22]

Table 11. Supplementary Metrics Used for Evaluating Code Search Techniques

Metric Type	Metric	Approach
Statistical test	Correlation analysis	GUIFetch [22], SCOR [3], Xie et al. [297]
	Mean squared error	Xie et al. [297]
	Hypothesis test	Contextual Search [91], Portfolio [45, 181, 184], Exemplar [77, 78, 180], McMillan et al. [182], Lemos et al. [143], SCP [254], Wang et al. [284], CodeGenie 2.0 [142], Lemos et al. [144], Code-Exchange [177], MUSE [188], CODE-NN [115], Niu et al. [203], ANNE [279], CodeLikeThis [176], LibFinder [206], QESC [108, 317]
User satisfaction	Experience score	CodeHint [71], Extended Conquer [93], Test Recommender [209], CodeExchange [177], CodeHow [169], MUSE [188], ANNE [279], CodeLikeThis [176], NLP2CODE [34]
	Mouse click	Example Overflow [311]
Counting	Absolute matching	Code Conjurer [111, 117], PRIME [186], Lemos et al. [144], Dy-CLINK [265], Quebio [121], GUIFetch [22], YOGO [212], Schumacher et al. [243]
	Top-k recommendation	INQRES [164], NCS [233], NQE [160], ExAssist [197, 199], BINGO-E [300], Extended Quebio [43]
Time	Retrieval/implementation time	Prospector [171], Jsearch [251], XSnippet [236], CodeGenie [137–139, 141], Code Conjurer [111, 117], S6 [223–225], Wang et al. [286], APPROX [20], Wang et al. [283], SnipMatch [292], Chan et al. [36], Satsy [260–262], Rendezvous [130], CodeHint [71], Tracelets [49], CodeExchange [177], AutoQuery [282], HUNTER [287], Dy-CLINK [265], Extended Satsy [263], SWIM [214], APIREC [194], ANNE [279], Source Forager [124], LibFinder [206], Gemini [299], Quebio [121], CodeNuance [162], Lancer [315], Aroma [166], DeepAPIRec [39], BINGO-E [300], Li et al. [152], CodeMatcher [159], CODEC [190], MP-CAT [86], AdaCS [154], PSCS [267], APIRec-CST [40], Extended Quebio [43]
Other metrics	External library	Coogle [235]
	Rate of passing test cases	S6 [223–225], APIRec-CST [40]
	BLEU	BVAE [42], CoaCor [305]
	METEOR	BVAE [42]
	Query coverage	CodeGenie 2.0 [142], Keivanloo et al. [127]

REFERENCES

[1] Sushil Bajracharya, Joel Ossher, and Cristina Lopes. 2010. Searching API usage examples in code repositories with sourcerer API search. In *Proceedings of the ICSE Workshop on Search-Driven Development: Users, Infrastructure, Tools and Evaluation*. 5–8.

[2] Lei Ai, Zhiqiu Huang, Weiwei Li, Yu Zhou, and Yaoshen Yu. 2019. Sensory: Leveraging code statement sequence information for code snippets recommendation. In *Proceedings of the 2019 IEEE 43rd Annual Computer Software and Applications Conference*, Vol. 1. IEEE, Los Alamitos, CA, 27–36.

[3] S. Akbar and A. Kak. 2019. SCOR: Source code retrieval with semantics and order. In *Proceedings of the 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR'19)*. 1–12.

[4] M. Akhin, N. Tillmann, M. Fähndrich, J. de Halleux, and M. Moskal. 2012. Search by example in TouchDevelop: Code search made easy. In *Proceedings of the 2012 4th International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation*. IEEE, Los Alamitos, CA, 5–8.

[5] Miltiadis Allamanis, Earl T. Barr, Premkumar Devanbu, and Charles Sutton. 2018. A survey of machine learning for big code and naturalness. *ACM Computing Surveys* 51, 4 (2018), 1–37.

[6] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. 2017. Learning to represent programs with graphs. *arXiv:1711.00740* (2017).

[7] Ambient Software Evolution Group. 2022. IJaDataset 2.0. Retrieved June 25, 2023 from https://onedrive.live.com/?authkey=%21AKDB2aMepVDO8as&id=8BFCB70AA333DB15%21260605&cid=8BFCB70AA333DB15&parId=root&parQt=sharedby&o=OneUp.

[8] Ambient Software Evolution Group. 2022. BigCloneBench. Retrieved June 26, 2023 from https://github.com/clonebench/BigCloneBench.

[9] A. Arwan, S. Rochimah, and R. J. Akbar. 2015. Source code retrieval on StackOverflow using LDA. In *Proceedings of the 2015 3rd International Conference on Information and Communication Technology (ICoICT'15)*. 295–299.

[10] M. H. Asyrofi, F. Thung, D. Lo, and L. Jiang. 2020. AUSearch: Accurate API usage search in GitHub repositories with type resolution. In *Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution, and Reengineering (SANER'20)*. 637–641.

[11] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. 1999. *Modern Information Retrieval*, Vol. 463. ACM, New York, NY.

[12] Sushil Bajracharya, Trung Ngo, Erik Linstead, Yimeng Dou, Paul Rigor, Pierre Baldi, and Cristina Lopes. 2006. Sourcerer: A search engine for open source code supporting structure-based search. In *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications*. ACM, New York, NY, 681–682.

[13] Sushil Bajracharya, Joel Ossher, and Cristina Lopes. 2014. Sourcerer: An infrastructure for large-scale collection and analysis of open-source code. *Science of Computer Programming* 79, Suppl. C (Jan. 2014), 241–259.

[14] Sushil Krishna Bajracharya and Cristina Videira Lopes. 2010. Analyzing and mining a code search engine usage log. *Empirical Software Engineering* 17, 4-5 (Sept. 2010), 424–466.

[15] Sushil Krishna Bajracharya and Cristina Videira Lopes. 2012. Analyzing and mining a code search engine usage log. *Empirical Software Engineering* 17, 4 (2012), 424–466.

[16] S. Baltes, R. Kiefer, and S. Diehl. 2017. Attribution required: Stack Overflow code snippets in GitHub projects. In *Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C'17)*.

[17] E. A. Barbosa, A. Garcia, and M. Mezini. 2012. Heuristic strategies for recommendation of exception handling code. In *Proceedings of the 2012 26th Brazilian Symposium on Software Engineering*. 171–180.

[18] E. A. Barbosa, A. Garcia, and M. Mezini. 2012. A recommendation system for exception handling code. In *Proceedings of the 2012 5th International Workshop on Exception Handling (WEH'12)*. 52–54.

[19] Ohad Barzilay, Christoph Treude, and Alexey Zagalsky. 2013. Facilitating crowd sourced software engineering via stack overflow. In *Finding Source Code on the Web for Remix and Reuse*. Springer, New York, NY, 289–308.

[20] S. Bazrafshan, R. Koschke, and N. Gode. 2011. Approximate code search in program histories. In *Proceedings of the 2011 18th Working Conference on Reverse Engineering*. 109–118.

[21] Kent Beck. 2003. *Test-Driven Development: By Example*. Addison-Wesley Professional.

[22] Farnaz Behrang, Steven P. Reiss, and Alessandro Orso. 2018. GUIFetch: Supporting app design and development through GUI search. In *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*. ACM, New York, NY, 236–246.

[23] Sumit Bhatia, Suppawong Tuarob, Prasenjit Mitra, and C. Lee Giles. 2011. An algorithm search engine for software developers. In *Proceedings of the 3rd International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation*. ACM, New York, NY, 13–16.

[24] T. F. Bissyande, F. Thung, D. Lo, Lingxiao Jiang, and L. Reveillere. 2013. Orion: A software project search engine with integrated diverse software artifacts. In *Proceedings of the 2013 18th International Conference on Engineering of Complex Computer Systems (ICECCS'13)*. 242–245.

[25] Bitbucket. 2022. Home Page. Retrieved June 26, 2023 from https://bitbucket.org.

[26] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet allocation. *Journal of Machine Learning Research* 3 (March 2003), 993–1022.

[27] Alessandro Bozzon, Marco Brambilla, and Piero Fraternali. 2010. Searching repositories of web application models. In *Web Engineering*. Springer, Berlin, Germany, 1–15.

[28] Andrew Bragdon, Steven P. Reiss, Robert Zeleznik, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra, and Joseph J. LaViola Jr. 2010. Code bubbles: Rethinking the user interface paradigm of integrated development environments. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, Vol. 1. 455–464.

[29] Andrew Bragdon, Robert Zeleznik, Steven P. Reiss, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra, and Joseph J. LaViola Jr. 2010. Code bubbles: A working set-based interface for code understanding and maintenance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2503–2512.

[30] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. 2009. Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, New York, NY, 1589–1598.

[31] John R. Brown and R. H. Hoffman. 1972. Evaluating the effectiveness of software verification: Practical experience with an automated tool. In *Proceedings of Fall Joint Computer Conference, Part I*. 181–190.

[32] Marcel Bruch, Martin Monperrus, and Mira Mezini. 2009. Learning from examples to improve code completion systems. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. 213–222.

[33] Jose Cambronero, Hongyu Li, Seohyun Kim, Koushik Sen, and Satish Chandra. 2019. When deep learning met code search. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, New York, NY, 964–974.

[34] Brock Angus Campbell and Christoph Treude. 2017. NLP2Code: Code snippet content assist via natural language tasks. In *Proceedings of the 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME'17)*. 628–632.

[35] Claudio Carpineto and Giovanni Romano. 2012. A survey of automatic query expansion in information retrieval. *ACM Computing Surveys* 44, 1 (2012), Article 1, 50 pages.

[36] Wing-Kwan Chan, Hong Cheng, and David Lo. 2012. Searching connected API subgraph via text phrases. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, New York, NY, 1–11.

[37] Mahinthan Chandramohan, Yinxing Xue, Zhengzi Xu, Yang Liu, Chia Yuan Cho, and Hee Beng Kuan Tan. 2016. Bingo: Cross-architecture cross-os binary search. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 678–689.

[38] Shaunak Chatterjee, Sudeep Juvekar, and Koushik Sen. 2009. SNIFF: A search engine for Java using free-form queries. In *Fundamental Approaches to Software Engineering*. Springer, Berlin, Germany, 385–400.

[39] Chi Chen, Xin Peng, Jun Sun, Zhenchang Xing, Xin Wang, Yifan Zhao, Hairui Zhang, and Wenyun Zhao. 2019. Generative API usage code recommendation with parameter concretization. *Science China Information Sciences* 62, 9 (2019), 192103.

[40] Chi Chen, Xin Peng, Zhenchang Xing, Jun Sun, Xin Wang, Yifan Zhao, and Wenyun Zhao. 2020. Holistic combination of structural and textual code information for context based API recommendation. *arXiv:2010.07514 [cs]* (2020). https://arxiv.org/abs/2010.07514v1.

[41] Hao Chen, Shi Ying, Jin Liu, and Wei Wang. 2004. SE4SC: A specific search engine for software components. In *Proceedings of the 2004 4th International Conference on Computer and Information Technology (CIT'04)*. IEEE, Los Alamitos, CA, 863–868.

[42] Qingying Chen and Minghui Zhou. 2018. A neural framework for retrieval and summarization of source code. In *Proceedings of the 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE'18)*. 826–831.

[43] Zhengzhao Chen, Renhe Jiang, Zejun Zhang, Yu Pei, Minxue Pan, Tian Zhang, and Xuandong Li. 2020. Enhancing example-based code search with functional semantics. *Journal of Systems and Software* 165 (2020), 110568.

[44] Tabnine. 2022. Home Page. Retrieved April 1, 2022 from https://www.codota.com/.

[45] Collin McMillan. 2011. Finding relevant functions in millions of lines of code. In *Proceedings of the 33rd International Conference on Software Engineering*. ACM, New York, NY, 1170–1172.

[46] Megan Conklin. 2007. Project entity matching across FLOSS repositories. In *Open Source Development, Adoption and Innovation (IFIP—The International Federation for Information Processing)*. Springer, Boston, MA, 45–57.

[47] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine Learning* 20, 3 (1995), 273–297.

[48] B. Dagenais and M. P. Robillard. 2012. Recovering traceability links between an API and its learning resources. In *Proceedings of the 2012 34th International Conference on Software Engineering (ICSE'12)*. 47–57.
[49] Yaniv David and Eran Yahav. 2014. Tracelet-based code search in executables. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, New York, NY, 349–360.
[50] Janet E. Davidson and Robert J. Sternberg (Eds.). 2003. *The Psychology of Problem Solving*. Cambridge University Press.
[51] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. 337–340.
[52] Marcelo de Rezende Martins and Marco Aurélio Gerosa. 2020. CoNCRA: A convolutional neural networks code retrieval approach. In *Proceedings of the 34th Brazilian Symposium on Software Engineering*. ACM, New York, NY, 526–531.
[53] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American Society for Information Science* 41, 6 (Sept. 1990), 391–407.
[54] U. Dekel and J. D. Herbsleb. 2009. Improving API documentation usability with knowledge pushing. In *Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*. 320–330.
[55] Serge Demeyer, Sander Tichelaar, and Stéphane Ducasse. 2001. *FAMIX 2.1—The FAMOOS Information Exchange Model*. Technical Report. University of Bern.
[56] Luca Di Grazia and Michael Pradel. 2023. Code search: A survey of techniques for finding code. *ACM Computing Surveys* 55, 11 (2023), 1–31.
[57] T. Diamantopoulos, K. Thomopoulos, and A. Symeonidis. 2016. QualBoa: Reusability-aware recommendations of source code components. In *Proceedings of the 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR'16)*. 488–491.
[58] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. 2013. Feature location in source code: A taxonomy and survey. *Journal of Software: Evolution and Process* 25, 1 (2013), 53–95.
[59] Donzhen Wen, Liang Yang 0003, Yingying Zhang, Yuan Lin, Kan Xu, and Hongfei Lin. 2020. Multi-level semantic representation model for code search. In *Proceedings of the Joint Conference of the Information Retrieval Communities in Europe (CIRCLE'20)*.
[60] Horatiu Dumitru, Marek Gibiec, Negar Hariri, Jane Cleland-Huang, Bamshad Mobasher, Carlos Castro-Herrera, and Mehdi Mirakhorli. 2011. On-demand feature recommendations derived from mining public product descriptions. In *Proceedings of the 33rd International Conference on Software Engineering*. 181–190.
[61] Frederico A. Durão, Taciana A. Vanderlei, Eduardo S. Almeida, and Silvio R. de L. Meira. 2008. Applying a semantic layer in a source code search tool. In *Proceedings of the 2008 ACM Symposium on Applied Computing*. ACM, New York, NY, 1151–1157.
[62] Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. 2015. Boa: Ultra-large-scale software repository and source-code mining. *ACM Transactions on Software Engineering Methodology* 25, 1 (2015), Article 7, 34 pages.
[63] Françoise Détienne and Frank Bott. 2001. *Software Design—Cognitive Aspects*. Springer-Verlag.
[64] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern Recognition Letters* 27, 8 (2006), 861–874.
[65] Gerhard Fischer, Scott Henninger, and David Redmiles. 1991. Cognitive tools for locating and comprehending software objects for reuse. In *Proceedings of the 13th International Conference on Software Engineering*. 318–328.
[66] Denis Foo Kune and Yongdae Kim. 2010. Timing attacks on pin input devices. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*. 678–680.
[67] W. B. Frakes and B. A. Nejmeh. 1986. Software reuse through information retrieval. *ACM SIGIR Forum* 21, 1 (1986), 30–36.
[68] Yuji Fujiwara, Norihiro Yoshida, Eunjong Choi, and Katsuro Inoue. 2019. Code-to-code search based on deep neural network and code mutation. In *Proceedings of the 2019 IEEE 13th International Workshop on Software Clones (IWSC'19)*. 1–7.
[69] George W. Furnas, Thomas K. Landauer, Louis M. Gomez, and Susan T. Dumais. 1987. The vocabulary problem in human-system communication. *Communications of the ACM* 30, 11 (1987), 964–971.
[70] Mark Gabel and Zhendong Su. 2010. A study of the uniqueness of source code. In *Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 147–156.
[71] Joel Galenson, Philip Reames, Rastislav Bodik, Björn Hartmann, and Koushik Sen. 2014. CodeHint: Dynamic and interactive synthesis of code snippets. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, New York, NY, 653–663.
[72] Rosalva E. Gallardo-Valencia and Susan Elliott Sim. 2009. Internet-scale code search. In *Proceedings of the 2009 ICSE Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation*. IEEE, Los Alamitos, CA, 49–52.
[73] Gitlab. 2022. Kernel.org Git Repositories. Retrieved January 26, 2023 from https://git.kernel.org.

[74] Google. 2022. Home Page. Retrieved June 26, 2023 from https://www.google.com.
[75] Google Code Jam. Home Page. 2022. Retrieved April 1, 2022 from https://developers.googleblog.com/2023/05/celebrate-googles-coding-competitions.html.
[76] Georgios Gousios, Bogdan Vasilescu, Alexander Serebrenik, and Andy Zaidman. 2014. Lean GHTorrent: GitHub data on demand. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, New York, NY.
[77] Mark Grechanik, Chen Fu, Qing Xie, Collin McMillan, Denys Poshyvanyk, and Chad Cumby. 2010. EXEMPLAR: EXEcutable exaMPLes ARchive. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, Vol. 2. ACM, New York, NY, 259–262.
[78] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. Cumby. 2010. A search engine for finding highly relevant applications. In *Proceedings of the ACM/IEEE 32nd International Conference on Software Engineering*, Vol. 1. 475–484.
[79] Wenchao Gu, Zongjie Li, Cuiyun Gao, Chaozheng Wang, Hongyu Zhang, Zenglin Xu, and Michael R. Lyu. 2020. CRaDLe: Deep code retrieval based on semantic dependency learning. *arXiv:2012.01028* (2020).
[80] X. Gu, H. Zhang, and S. Kim. 2018. Deep code search. In *Proceedings of the 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE'18)*. 933–944.
[81] X. Gu, H. Zhang, and S. Kim. 2019. CodeKernel: A graph kernel based approach to the selection of API usage examples. In *Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE'19)*. 590–601.
[82] Florian S. Gysin. 2010. Improved social trustability of code search results. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, Vol. 2. ACM, New York, NY, 513–514.
[83] Florian S. Gysin and Adrian Kuhn. 2010. A trustability metric for code search based on developer karma. In *Proceedings of 2010 ICSE Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation*. ACM, New York, NY, 41–44.
[84] Sonia Haiduc, Gabriele Bavota, Andrian Marcus, Rocco Oliveto, Andrea De Lucia, and Tim Menzies. 2013. Automatic query reformulations for text retrieval in software engineering. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE, Los Alamitos, CA, 842–851.
[85] Sonia Haiduc, Giuseppe De Rosa, Gabriele Bavota, Rocco Oliveto, Andrea De Lucia, and Andrian Marcus. 2013. Query quality prediction and reformulation for source code search: The Refoqus tool. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE, Los Alamitos, CA, 1307–1310.
[86] Rajarshi Haldar, Lingfei Wu, Jinjun Xiong, and Julia Hockenmaier. 2020. A Multi-perspective architecture for semantic code search. *arXiv:2005.06980 [cs]* (2020).
[87] Vincent J. Hellendoorn and Premkumar Devanbu. 2017. Are deep neural networks the best choice for modeling source code? In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 763–773.
[88] S. Henninger. 1994. Using iterative refinement to find reusable software. *IEEE Software* 11, 5 (Sept. 1994), 48–59.
[89] Geert Heyman and Tom Van Cutsem. 2020. Neural code search revisited: Enhancing code snippet retrieval through natural language intent. *arXiv:2008.12193* (2020).
[90] Emily Hill. 2010. *Integrating Natural Language and Program Structure Information to Improve Software Search and Exploration*. University of Delaware.
[91] Emily Hill, Lori Pollock, and K. Vijay-Shanker. 2009. Automatically capturing source code context of NL-queries for software maintenance and reuse. In *Proceedings of the 31st International Conference on Software Engineering*. IEEE, Los Alamitos, CA, 232–242.
[92] Emily Hill, Lori Pollock, and K. Vijay-Shanker. 2011. Improving source code search with natural language phrasal representations of method signatures. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, Los Alamitos, CA, 524–527.
[93] Emily Hill, Manuel Roldan-Vega, Jerry Alan Fails, and Greg Mallet. 2014. NL-based query refinement and contextualized code search results: A user study. In *Proceedings of 2014 Software Evolution Week: IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE'14)*. 34–43.
[94] Reid Holmes. 2009. Do developers search for source code examples using multiple facts? In *Proceedings of the 2009 ICSE Workshop on Search-Driven : Users, Infrastructure, Tools, and Evaluation*. IEEE, Los Alamitos, CA, 13–16.
[95] Reid Holmes and Gail C. Murphy. 2005. Using structural context to recommend source code examples. In *Proceedings of the 27th International Conference on Software Engineering*. ACM, New York, NY, 117–125.
[96] Reid Holmes, Robert J. Walker, and Gail C. Murphy. 2005. Strathcona example recommendation tool. In *Proceedings of the 10th European Software Engineering Conference Held Jointly with the 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, New York, NY, 237–240.
[97] R. Holmes, R. J. Walker, and G. C. Murphy. 2006. Approximate structural context matching: An approach to recommend relevant examples. *IEEE Transactions on Software Engineering* 32, 12 (2006), 952–970.

[98] Adrian Holovaty and Jacob Kaplan-Moss. 2009. *The Definitive Guide to DJANGO: Web Development Done Right*. Apress.

[99] James Howison, Megan Squire, and Kevin Crowston. 2008. FLOSSmole: A collaborative repository for FLOSS research data and analyses. *International Journal of Information Technology and Web Engineering* 1 (Sept. 2008), 17–26.

[100] Sheng-Kuei Hsu and Shi-Jen Lin. 2011. A block-structured model for source code retrieval. In *Intelligent Information and Database Systems*, Ngoc Thanh Nguyen, Chong-Gun Kim, and Adam Janiak (Eds.). Springer, 161–170.

[101] Sheng-Kuei Hsu and Shi-Jen Lin. 2011. A block-structured model for source code retrieval. In *Intelligent Information and Database Systems*. Springer, Berlin, Germany, 161–170.

[102] Gang Hu, Min Peng, Yihan Zhang, Qianqian Xie, Wang Gao, and Mengting Yuan. 2020. Unsupervised software repositories mining and its application to code search. *Software: Practice and Experience* 50, 3 (2020), 299–322.

[103] Gang Hu, Min Peng, Yihan Zhang, Qianqian Xie, and Mengting Yuan. 2020. Neural joint attention code search over structure embeddings for software Q&A sites. *Journal of Systems and Software* 170 (2020), 110773.

[104] Q. Huang, A. Qiu, M. Zhong, and Y. Wang. 2020. A code-description representation learning model based on attention. In *Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution, and Reengineering (SANER'20)*. 447–455.

[105] Qing Huang, Xudong Wang, Yangrui Yang, Hongyan Wan, Rui Wang, and Guoqing Wu. 2017. SnippetGen:Enhancing the code search via intent predicting. In *Proceedings of the 29th International Conference on Software Engineering and Knowledge Engineering*. 307–312.

[106] Qing Huang and Guoqing Wu. 2019. Enhance code search via reformulating queries with evolving contexts. *Automated Software Engineering* 26, 4 (2019), 705–732.

[107] Qing Huang and Huaiguang Wu. 2019. QE-integrating framework based on GitHub knowledge and SVM ranking. *Science China Information Sciences* 62, 5 (2019), 52102.

[108] Qing Huang, Yang Yang, and Ming Cheng. 2019. Deep learning the semantics of change sequences for query expansion. *Software: Practice and Experience* 49, 11 (2019), 1600–1617.

[109] Qing Huang, Yangrui Yang, Xue Zhan, Hongyan Wan, and Guoqing Wu. 2018. Query expansion based on statistical learning from code changes. *Software: Practice and Experience* 48, 7 (2018), 1333–1351.

[110] Y. Huang, Q. Kong, N. Jia, X. Chen, and Z. Zheng. 2019. Recommending differentiated code to support smart contract update. In *Proceedings of the 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC'19)*. 260–270.

[111] O. Hummel, W. Janjic, and C. Atkinson. 2008. Code conjurer: Pulling reusable software out of thin air. *IEEE Software* 25, 5 (2008), 45–52.

[112] Hamel Husain. 2018. Towards natural language semantic code search. *GitHub Blog*. Retrieved June 26, 2023 from https://github.blog/2018-09-18-towards-natural-language-semantic-code-search/.

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

[114] M. M. Islam and R. Iqbal. 2020. SoCeR: A new source code recommendation technique for code reuse. In *Proceedings of the 2020 IEEE 44th Annual Computers, Software, and Applications Conference*. 1552–1557.

[115] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2073–2083.

[116] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2073–2083.

[117] Werner Janjic, Dietmar Stoll, Philipp Bostan, and Colin Atkinson. 2009. Lowering the barrier to reuse through test-driven search. In *Proceedings of the 2009 ICSE Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation*. IEEE, Los Alamitos, CA, 21–24.

[118] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems* 20, 4 (2002), 422–446.

[119] H. Jiang, L. Nie, Z. Sun, Z. Ren, W. Kong, T. Zhang, and X. Luo. 2016. ROSF: Leveraging information retrieval and supervised learning for recommending code snippets. *IEEE Transactions on Services Computing* 12, 1 (2016), 34–46.

[120] Lingxiao Jiang and Zhendong Su. 2009. Automatic mining of functionally equivalent code fragments via random testing. In *Proceedings of the 18th International Symposium on Software Testing and Analysis*. ACM, New York, NY, 81–92.

[121] Renhe Jiang, Zhengzhao Chen, Zejun Zhang, Yu Pei, Minxue Pan, and Tian Zhang. 2018. Semantics-based code search using input/output examples. In *Proceedings of the 2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM'18)*. 92–102.

[122] Huan Jin and Lei Xiong. 2019. A query expansion method based on evolving source code. *Wuhan University Journal of Natural Sciences* 24, 5 (2019), 391–399.

[123] Toshihiro Kamiya. 2021. CCFinderX: An interactive code clone analysis environment. In *Code Clone Analysis.* Springer, 31–44.

[124] Vineeth Kashyap, David Bingham Brown, Ben Liblit, David Melski, and Thomas Reps. 2017. Source Forager: A search engine for similar source code. *arXiv:1706.02769* (2017).

[125] Amandeep Kaur and Gaurav Dhiman. 2019. A review on search-based tools and techniques to identify bad code smells in object-oriented systems. In *Harmony Search and Nature Inspired Optimization Algorithms: Theory and Applications.* Advances in Intelligent Systems and Computing, Vol. 741. Springer, 909–921.

[126] Iman Keivanloo, Juergen Rilling, and Philippe Charland. 2011. SeClone—A hybrid approach to Internet-scale real-time code clone search. In *Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension.* IEEE, Los Alamitos, CA, 223–224.

[127] Iman Keivanloo, Juergen Rilling, and Ying Zou. 2014. Spotting working code examples. In *Proceedings of the 36th International Conference on Software Engineering.* ACM, New York, NY, 664–675.

[128] I. Keivanloo, L. Roostapour, P. Schugerl, and J. Rilling. 2010. SE-CodeSearch: A scalable semantic Web-based source code search infrastructure. In *Proceedings of the 2010 IEEE International Conference on Software Maintenance.* 1–5.

[129] Marcus Kessel and Colin Atkinson. 2018. Integrating reuse into the rapid, continuous software engineering cycle through test-driven search. In *Proceedings of the 2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE'18).* 8–11.

[130] W. M. Khoo, A. Mycroft, and R. Anderson. 2013. Rendezvous: A search engine for binary code. In *Proceedings of the 2013 10th Working Conference on Mining Software Repositories (MSR'13).* 329–338.

[131] Heejung Kim, Yungbum Jung, Sunghun Kim, and Kwankeun Yi. 2011. MeCC: Memory comparison-based clone detector. In *Proceedings of the 2011 33rd International Conference on Software Engineering (ICSE'11).* 301–310.

[132] Jinhan Kim, Sanghoon Lee, Seung-Won Hwang, and Sunghun Kim. 2010. Towards an intelligent code search engine. In *Proceedings of the 24th Conference on Artificial Intelligence.*

[133] Kisub Kim, Dongsun Kim, Tegawendé F. Bissyandé, Eunjong Choi, Li Li, Jacques Klein, and Yves Le Traon. 2018. FaCOY—A code-to-code search engine. In *Proceedings of the 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE'18).* 946–957.

[134] Krugle. 2022. Home Page. Retrieved June 26, 2023 from http://krugle.com.

[135] Daniel E. Krutz and Emad Shihab. 2013. CCCD: Concolic code clone detection. In *Proceedings of the 2013 20th Working Conference on Reverse Engineering (WCRE'13).* 489–490.

[136] Frederick Wilfrid Lancaster and Emily Gallup. 1973. *Information Retrieval On-Line.* Technical Report. Melville Publishing Company.

[137] Otávio Augusto Lazzarini Lemos, Sushil Bajracharya, Joel Ossher, Paulo Cesar Masiero, and Cristina Lopes. 2009. Applying test-driven code search to the reuse of auxiliary functionality. In *Proceedings of the 2009 ACM Symposium on Applied Computing.* ACM, New York, NY, 476–482.

[138] Otávio Augusto Lazzarini Lemos, Sushil Bajracharya, Joel Ossher, Paulo Cesar Masiero, and Cristina Lopes. 2011. A test-driven approach to code search and its application to the reuse of auxiliary functionality. *Information and Software Technology* 53, 4 (2011), 294–306.

[139] Otávio Augusto Lazzarini Lemos, Sushil Krishna Bajracharya, and Joel Ossher. 2007. CodeGenie: A tool for test-driven source code search. In *Companion to the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications Companion.* ACM, New York, NY, 917–918.

[140] Shin-Jie Lee, Xavier Lin, Wu-Chen Su, and Hsi-Min Chen. 2018. A comment-driven approach to API usage patterns discovery and search. *Journal of Internet Technology* 19, 5 (2018), 1587–1601.

[141] Otávio Augusto Lazzarini Lemos, Sushil Krishna Bajracharya, Joel Ossher, Ricardo Santos Morla, Paulo Cesar Masiero, Pierre Baldi, and Cristina Videira Lopes. 2007. CodeGenie: Using test-cases to search and reuse source code. In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering.* ACM, New York, NY, 525–526.

[142] Otávio A. L. Lemos, Adriano C. de Paula, Felipe C. Zanichelli, and Cristina V. Lopes. Thesaurus-based automatic query expansion for interface-driven code search. In *Proceedings of the 11th Working Conference on Mining Software Repositories.* ACM, New York, NY, 212–221.

[143] Otavio Augusto Lazzarini Lemos, Adriano Carvalho de Paula, Gustavo Konishi, Joel Ossher, Sushil Bajracharya, and Cristina Lopes. Using thesaurus-based tag clouds to improve test-driven code search. In *Proceedings of the 2013 VII Brazilian Symposium on Software Components, Architectures, and Reuse.* 99–108.

[144] O. A. L. Lemos, A. C. de Paula, H. Sajnani, and C. V. Lopes. 2015. Can the use of types and query expansion help improve large-scale code search? In *Proceedings of the IEEE 15th International Working Conference on Source Code Analysis and Manipulation.* 41–50.

[145] Hongyu Li, Seohyun Kim, and Satish Chandra. 2019. Neural code search evaluation dataset. *arXiv:1908.09804* (2019).

[146] R. Li, G. Hu, and M. Peng. 2020. Hierarchical embedding for code search in software Q&A sites. In *Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN'20).* 1–10.

[147] Shengying Li. 2004. *A Survey on Tools for Binary Code Analysis*. Stony Brook University, 37–52.

[148] Sihan Li, Xusheng Xiao, Blake Bassett, Tao Xie, and Nikolai Tillmann. 2016. Measuring code behavioral similarity for programming and software engineering education. In *Proceedings of the 38th International Conference on Software Engineering Companion.* ACM, New York, NY, 501–510.

[149] W. Li, H. Qin, S. Yan, B. Shen, and Y. Chen. 2020. Learning code-query interaction for enhancing code searches. In *Proceedings of the 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME'20).* 115–126.

[150] Wei Li, Shuhan Yan, Beijun Shen, and Yuting Chen. 2019. Reinforcement learning of code search sessions. In *Proceedings of the 2019 26th Asia-Pacific Software Engineering Conference (APSEC'19).* 458–465.

[151] Xuan Li, Zerui Wang, Qianxiang Wang, Shoumeng Yan, Tao Xie, and Hong Mei. 2016. Relationship-aware code search for JavaScript frameworks. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering.* ACM, New York, NY, 690–701.

[152] Yang Li, Suhang Wang, Quan Pan, Haiyun Peng, Tao Yang, and Erik Cambria. 2019. Learning binary codes with neural collaborative filtering for efficient recommendation systems. *Knowledge-Based Systems* 172 (2019), 64–75.

[153] Zhixing Li, Tao Wang, Yang Zhang, Yun Zhan, and Gang Yin. 2016. Query reformulation by leveraging crowd wisdom for scenario-based software search. In *Proceedings of the 8th Asia-Pacific Symposium on Internetware.* ACM, New York, NY, 36–44.

[154] Chunyang Ling, Zeqi Lin, Yanzhen Zou, and Bing Xie. 2020. Adaptive deep code search. In *Proceedings of the 28th International Conference on Program Comprehension.* 48–59.

[155] Xiang Ling, Lingfei Wu, Saizhuo Wang, Gaoning Pan, Tengfei Ma, Fangli Xu, Alex X. Liu, Chunming Wu, and Shouling Ji. 2020. Deep graph matching and searching for semantic code retrieval. *arXiv:2010.12908* (2020).

[156] Linus Torvalds and Junio C. Hamano. 2022. Home Page. Retrieved April 1, 2022 from https://git-scm.com/.

[157] Chao Liu, Cuiyun Gao, Xin Xia, David Lo, John Grundy, and Xiaohu Yang. 2020. On the replicability and reproducibility of deep learning in software engineering. *arXiv preprint arXiv:2006.14244* (2020).

[158] Chao Liu, Xin Xia, David Lo, Cuiyun Gao, Xiaohu Yang, and John Grundy. 2020. Opportunities and challenges in code search tools. *arXiv:2011.02297 [cs]* (Nov. 2020).

[159] Chao Liu, Xin Xia, David Lo, Zhiwei Liu, Ahmed E. Hassan, and Shanping Li. 2020. Simplifying deep-learning-based model for code search. *arXiv:2005.14373* (2020).

[160] Jason Liu, Seohyun Kim, Vijayaraghavan Murali, Swarat Chaudhuri, and Satish Chandra. 2019. Neural query expansion for code search. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages.* ACM, New York, NY, 29–37.

[161] Kui Liu, Anil Koyuncu, Kisub Kim, Dongsun Kim, and Tegawendé F. Bissyandé. 2018. LSRepair: Live search of fix ingredients for automated program repair. In *Proceedings of the 2018 25th Asia-Pacific Software Engineering Conference (APSEC'18).* IEEE, Los Alamitos, CA, 658–662.

[162] Wenjian Liu, Xin Peng, Zhenchang Xing, Junyi Li, Bing Xie, and Wenyun Zhao. 2018. Supporting exploratory code search with differencing and visualization. In *Proceedings of the 2018 IEEE 25th International Conference on Software Analysis, Evolution, and Reengineering.* 300–310.

[163] Angela Lozano, Andy Kellens, and Kim Mens. 2011. Mendel: Source code recommendation based on a genetic metaphor. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering.* IEEE, Los Alamitos, CA, 384–387.

[164] Jinting Lu, Ying Wei, Xiaobing Sun, Bin Li, Wanzhi Wen, and Cheng Zhou. 2018. Interactive query reformulation for source-code search with word relations. *IEEE Access* 6 (2018), 75660–75668.

[165] Meili Lu, X. Sun, S. Wang, D. Lo, and Yucong Duan. 2015. Query expansion via WordNet for effective code search. In *Proceedings of the 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER'15).* 545–549.

[166] Sifei Luan, Di Yang, Celeste Barnaby, Koushik Sen, and Satish Chandra. 2019. Aroma: Code recommendation via structural code search. *Proceedings of the ACM on Programming Languages* 3 (2019), Article 152, 28 pages.

[167] George F. Luger, P. Johnson, C. Sterm, Jean E. Newman, and Ronald Yeo. 1994. *Cognitive Science: The Science of Intelligent Systems.* Academic Press.

[168] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn. 2008. Source code retrieval for bug localization using latent Dirichlet allocation. In *Proceedings of the 2008 15th Working Conference on Reverse Engineering.* 155–164.

[169] Fei Lv, Hongyu Zhang, Jian-Guang Lou, Shaowei Wang, Dongmei Zhang, and Jianjun Zhao. 2015. CodeHow: Effective code search based on API understanding and extended Boolean model. In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering.* 260–270.

[170] Y. Malheiros, A. Moraes, C. Trindade, and S. Meira. 2012. A source code recommender system to support newcomers. In *Proceedings of the 2012 IEEE 36th Annual Computer Software and Applications Conference.* 19–24.

[171] David Mandelin, Lin Xu, Rastislav Bodík, and Doug Kimelman. 2005. Jungloid mining: Helping to navigate the API jungle. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation.* ACM, New York, NY, 48–61.

[172] Christopher D. Manning, Hinrich Schütze, and Prabhakar Raghavan. 2008. *Introduction to Information Retrieval.* Cambridge University Press.

[173] L. W. Mar, Y. Wu, and H. C. Jiau. 2011. Recommending proper API code examples for documentation purpose. In *Proceedings of the 2011 18th Asia-Pacific Software Engineering Conference.* 331–338.

[174] Gary Marchionini. 2006. Exploratory search: From finding to understanding. *Communications of the ACM* 49, 4 (2006), 41–46.

[175] L. Martie and A. van der Hoek. 2013. Toward social-technical code search. In *Proceedings of the 2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE'13).* 101–104.

[176] Lee Martie, André van der Hoek, and Thomas Kwak. 2017. Understanding the impact of support for iteration on code search. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, New York, NY, 774–785.

[177] L. Martie, T. D. LaToza, and A. van der Hoek. 2015. CodeExchange: Supporting reformulation of Internet-scale code queries in context (T). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE'15).* 24–35.

[178] Lee Martie and André van der Hoek. 2015. Sameness: An experiment in code search. In *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE, Los Alamitos, CA, 76–87.

[179] Max Howell. 2022. Homebrew. Retrieved June 26, 2023 from https://brew.sh.

[180] C. McMillan, M. Grechanik, D. Poshyvanyk, Chen Fu, and Qing Xie. 2012. Exemplar: A source code search engine for finding highly relevant applications. *IEEE Transactions on Software Engineering* 38, 5 (2012), 1069–1087.

[181] Collin McMillan, Mark Grechanik, Denys Poshyvanyk, Qing Xie, and Chen Fu. 2011. Portfolio: Finding relevant functions and their usage. In *Proceeding of the 33rd International Conference on Software Engineering*. ACM, New York, NY, 111–120.

[182] C. McMillan, N. Hariri, D. Poshyvanyk, J. Cleland-Huang, and B. Mobasher. 2012. Recommending source code for use in rapid software prototypes. In *Proceedings of the 2012 34th International Conference on Software Engineering (ICSE'12).* 848–858.

[183] Collin McMillan, Denys Poshyvanyk, and Mark Grechanik. 2010. Recommending source code examples via API call usages and documentation. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*. ACM, New York, NY, 21–25.

[184] Collin Mcmillan, Denys Poshyvanyk, Mark Grechanik, Qing Xie, and Chen Fu. 2013. Portfolio: Searching for relevant functions and their usages in millions of lines of code. *ACM Transactions on Software Engineering and Methodology* 22, 4 (2013), Article 37, 30 pages.

[185] Microsoft. 2022. Microsoft Bing. Retrieved June 26, 2023 from https://www.bing.com.

[186] Alon Mishne, Sharon Shoham, and Eran Yahav. 2012. Typestate-based semantic code search over partial programs. In *Proceedings of the ACM International Conference on Object-Oriented Programming Systems Languages and Applications.* ACM, New York, NY, 997–1016.

[187] A. Mockus. 2009. Amassing and indexing a large sample of version control systems: Towards the census of public source code history. In *Proceedings of the 2009 6th International Working Conference on Mining Software Repositories.* 11–20.

[188] Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Andrian Marcus. 2015. How can I use this method? In *Proceedings of the 37th International Conference on Software Engineering*, Vol. 1. IEEE, Los Alamitos, CA, 880–890.

[189] Ibrahim Jameel Mujhid, Joanna C. S. Santos, Raghuram Gopalakrishnan, and Mehdi Mirakhorli. 2017. A search engine for finding and reusing architecturally significant code. *Journal of Systems and Software* 130 (Aug. 2017), 81–93.

[190] Rohan Mukherjee, Swarat Chaudhuri, and Chris Jermaine. 2020. Searching a database of source codes using contextualized code search. *arXiv:2001.03277* (2020).

[191] Naoya Murakami and Hidehiko Masuhara. 2012. Optimizing a search-based code recommendation system. In *Proceedings of the 3rd International Workshop on Recommendation Systems for Software Engineering*. IEEE, Los Alamitos, CA, 68–72.

[192] Naoya Murakami, Hidehiko Masuhara, and Tomoyuki Aotani. 2014. Code recommendation based on a degree-of-interest model. In *Proceedings of the 4th International Workshop on Recommendation Systems for Software Engineering.* ACM, New York, NY, 28–29.

[193] Takuma Murakami, Zhenjiang Hu, Shingo Nishioka, Akihiko Takano, and Masato Takeichi. 2004. An algebraic interface for GETA search engine. In *Proceedings of the Program and Programming Language Workshop*.

[194] Anh Tuan Nguyen, Michael Hilton, Mihai Codoban, Hoan Anh Nguyen, Lily Mast, Eli Rademacher, Tien N. Nguyen, and Danny Dig. 2016. API code recommendation using statistical learning from fine-grained changes. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, New York, NY, 511–522.

[195] Anh Tuan Nguyen, Tung Thanh Nguyen, Jafar Al-Kofahi, Hung Viet Nguyen, and Tien N. Nguyen. 2011. A topic-based approach for narrowing the search space of buggy files from a bug report. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE'11)*. IEEE, Los Alamitos, CA, 263–272.

[196] T. Nguyen, P. Vu, and T. Nguyen. 2019. Personalized code recommendation. In *Proceedings of the 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME'19)*. 313–317.

[197] T. Nguyen, P. Vu, and T. Nguyen. 2019. Recommending exception handling code. In *Proceedings of the 2019 IEEE International Conference on Software Maintenance and Evolution (ICSME'19)*. 390–393.

[198] Tam The Nguyen, Phong Minh Vu, and Tung Thanh Nguyen. 2019. Code search on bytecode for mobile APP development. In *Proceedings of the 2019 ACM Southeast Conference*. ACM, New York, NY, 253–256.

[199] Tam The Nguyen, Phong Minh Vu, and Tung Thanh Nguyen. 2019. Recommendation of exception handling code in mobile APP development. *arXiv:1908.06567* (2019).

[200] T. Van Nguyen, A. T. Nguyen, H. D. Phan, T. D. Nguyen, and T. N. Nguyen. 2017. Combining Word2Vec with revised vector space model for better code retrieval. In *Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C'17)*. 183–185.

[201] L. Nie, H. Jiang, Z. Ren, Z. Sun, and X. Li. 2016. Query expansion based on crowd knowledge for code search. *IEEE Transactions on Services Computing* 9 (Sept. 2016), 771–783.

[202] H. Niu. 2015. *Improving Code Search Using Learning-to-Rank and Query Reformulation Techniques*. Master's thesis. Queen's University, Canada.

[203] Haoran Niu, Iman Keivanloo, and Ying Zou. 2017. Learning to rank code examples for code search engines. *Empirical Software Engineering* 22 (2017), 1–33.

[204] OpenHub. 2022. Synopsis/Black Duck Open Hub. Retrieved June 26, 2023 from https://www.openhub.net.

[205] Oracle. 2022. Java Platform, Standard Edition 7, API Specification. Retrieved June 26, 2023 from https://docs.oracle.com/javase/7/docs/api/.

[206] Ali Ouni, Raula Gaikovina Kula, Marouane Kessentini, Takashi Ishio, Daniel M. German, and Katsuro Inoue. 2017. Search-based software library recommendation using multi-objective optimization. *Information and Software Technology* 83, Suppl. C (2017), 55–75.

[207] Yoann Padioleau, Julia Lawall, René Rydhof Hansen, and Gilles Muller. 2008. Documenting and automating collateral evolutions in Linux device drivers. *ACM SIGOPS Operating Systems Review* 42, 4 (2008), 247–260.

[208] P. Pathak, M. Gordon, and Weiguo Fan. 2000. Effective information retrieval using genetic algorithms based matching functions adaptation. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*.

[209] Raphael Pham, Yauheni Stoliar, and Kurt Schneider. 2015. Automatically recommending test code examples to inexperienced developers. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, New York, NY, 890–893.

[210] Nina Phan, Peter Bailey, and Ross Wilkinson. 2007. Understanding the relationship of information need specificity to search query length. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, 709–710.

[211] Iasonas Polakis, Georgios Kontaxis, Spiros Antonatos, Eleni Gessiou, Thanasis Petsas, and Evangelos P. Markatos. 2010. Using social networks to harvest email addresses. In *Proceedings of the 9th Annual ACM Workshop on Privacy in the Electronic Society*. 11–20.

[212] Varot Premtoon, James Koppel, and Armando Solar-Lezama. 2020. Semantic code search via equational reasoning. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, New York, NY, 1066–1082.

[213] Ruchir Puri, David S. Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladmir Zolotov, Julian Dolby, et al. 2021. Project CodeNet: A large-scale ai for code dataset for learning a diversity of coding tasks. *arXiv preprint arXiv:2015.12655* (2021).

[214] M. Raghothaman, Y. Wei, and Y. Hamadi. 2016. SWIM: Synthesizing what I mean—Code search and idiomatic snippet synthesis. In *Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE'16)*. 357–367.

[215] C. Ragkhitwetsagul. 2016. Measuring code similarity in large-scaled code corpora. In *Proceedings of the 2016 International Conference on Software Maintenance and Evolution (ICSME'16)*. 626–630.

[216] Mohammad Masudur Rahman and Chanchal Roy. 2018. Effective reformulation of query for code search using crowdsourced knowledge and extra-large data analytics. In *Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME'18).* 473–484.

[217] Mohammad Masudur Rahman and Chanchal Roy. 2018. NLP2API: Query reformulation for code search using crowdsourced knowledge and extra-large data analytics. In *Proceedings of the 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME'18).* 714–714.

[218] M. M. Rahman and C. K. Roy. 2014. On the use of context in recommending exception handling code examples. In *Proceedings of the 2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation.* 285–294.

[219] Mohammad Masudur Rahman and Chanchal K. Roy. 2021. A systematic literature review of automated query reformulations in source code search. *arXiv preprint arXiv:2108.09646* (2021).

[220] Mohammad M. Rahman, Chanchal K. Roy, and David Lo. 2019. Automatic query reformulation for code search using crowdsourced knowledge. *Empirical Software Engineering* 24, 4 (2019), 1869–1924.

[221] M. M. Rahman, C. K. Roy, and D. Lo. RACK: Code search in the IDE using crowdsourced knowledge. In *Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C'17).* 51–54.

[222] Karthik Ram. 2013. Git can facilitate greater reproducibility and increased transparency in science. *Source Code for Biology and Medicine* 8 (Feb. 2013), 7.

[223] Steven P. Reiss. 2009. Semantics-based code search. In *Proceedings of the 31st International Conference on Software Engineering.* IEEE, Los Alamitos, CA, 243–253.

[224] S. P. Reiss. 2009. Semantics-based code search demonstration proposal. In *Proceedings of the 2009 IEEE International Conference on Software Maintenance.* 385–386.

[225] S. P. Reiss. 2009. Specifying what to search for. In *Proceedings of the Tools and Evaluation 2009 ICSE Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation.* 41–44.

[226] S. P. Reiss. 2013. Integrating S6 code search and Code Bubbles. In *Proceedings of the 2013 3rd International Workshop on Developing Tools as Plug-Ins (TOPI'13).* 25–30.

[227] Steven P. Reiss, Yun Miao, and Qi Xin. 2018. Seeking the user interface. *Automated Software Engineering* 25, 1 (2018), 157–193.

[228] Leiming Ren, Shinmin Shan, Kai Wang, and Kun Xue. 2020. CSDA: A novel attention-based LSTM approach for code search. *Journal of Physics: Conference Series* 1544 (2020), 012056.

[229] Romain Robbes and Michele Lanza. 2010. Improving code completion with program history. *Automated Software Engineering* 17, 2 (2010), 181–212.

[230] M. Roldan-Vega, G. Mallet, E. Hill, and J. A. Fails. 2013. CONQUER: A tool for NL-based query refinement and contextualizing code search results. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance.* 512–515.

[231] Chanchal Kumar Roy and James R. Cordy. 2007. A survey on software clone detection research. *Queen's School of Computing TR* 541, 115 (2007), 64–68.

[232] Julia Rubin and Marsha Chechik. 2013. A survey of feature location techniques. In *Domain Engineering.* Springer, 29–58.

[233] Saksham Sachdev, Hongyu Li, Sifei Luan, Seohyun Kim, Koushik Sen, and Satish Chandra. 2018. Retrieval on source code: A neural code search. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages.* ACM, New York, NY, 31–41.

[234] Caitlin Sadowski, Kathryn T. Stolee, and Sebastian Elbaum. 2015. How developers search for code: A case study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering.* ACM, New York, NY, 191–201.

[235] Tobias Sager, Abraham Bernstein, Martin Pinzger, and Christoph Kiefer. 2006. Detecting similar Java classes using tree algorithms. In *Proceedings of the 2006 International Workshop on Mining Software Repositories.* ACM, New York, NY, 65–71.

[236] Naiyana Sahavechaphan and Kajal Claypool. 2006. XSnippet: Mining for sample code. *ACM SIGPLAN Notices* 41, 10 (2006), 413–430.

[237] Hitesh Sajnani, Vaibhav Saini, Jeffrey Svajlenko, Chanchal K. Roy, and Cristina V. Lopes. 2016. SourcererCC: Scaling code clone detection to big-code. In *Proceedings of the 38th International Conference on Software Engineering.* ACM, New York, NY, 1157–1168.

[238] Gerard Salton, Anita Wong, and Chung-Shu Yang. 1975. A vector space model for automatic indexing. *Communications of the ACM* 18, 11 (1975), 613–620.

[239] Huascar Sanchez. 2013. SNIPR: Complementing code search with code retargeting capabilities. In *Proceedings of the 2013 International Conference on Software Engineering.* IEEE, Los Alamitos, CA, 1423–1426.

[240] Abdus Satter, M. G. Muntaqeem, Nadia Nahar, and Kazi Sakib. 2017. Retrieving self-executable and functionally correct code to improve source code search. In *Proceedings of the 2017 24th Asia-Pacific Software Engineering Conference (APSEC'17).* 749–750.

[241] A. Satter and K. Sakib. A search log mining based query expansion technique to improve effectiveness in code search. In *Proceedings of the 2016 19th International Conference on Computer and Information Technology (ICCIT'16).* 586–591.

[242] Abdus Satter and Kazi Sakib. A similarity-based method retrieval technique to improve effectiveness in code search. In *Companion to the 1st International Conference on the Art, Science, and Engineering of Programming.* ACM, New York, NY, 1–3.

[243] Max Eric Henry Schumacher, Kim Tuyen Le, and Artur Andrzejak. 2020. Improving code recommendations by combining neural and classical machine learning approaches. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops.* ACM, New York, NY, 476–482.

[244] Searchcode. 2022. Home Page. Retrieved June 26, 2023 from https://searchcode.com.

[245] Shailesh Kumar Shivakumar. 2021. A survey and taxonomy of intent-based code search. *International Journal of Software Innovation* 9, 1 (2021), 69–110.

[246] Jianhang Shuai, Ling Xu, Chao Liu, Meng Yan, Xin Xia, and Yan Lei. 2020. Improving code search with co-attentive representation learning. In *Proceedings of the 28th International Conference on Program Comprehension.* ACM, New York, NY, 196–207.

[247] Susan Elliott Sim, Megha Agarwala, and Medha Umarji. 2013. A controlled experiment on the process used by developers during Internet-scale code search. In *Finding Source Code on the Web for Remix and Reuse.* Springer, New York, NY, 53–77.

[248] S. E. Sim, C. L. A. Clarke, and R. C. Holt. 1998. Archetypal source code searches: A survey of software developers and maintainers. In *Proceedings of the 1998 6th International Workshop on Program Comprehension.* 180–187.

[249] Susan Elliott Sim, Medha Umarji, Sukanya Ratanotayanon, and Cristina V. Lopes. 2011. How well do search engines support code retrieval on the Web? *ACM Transactions on Software Engineering and Methodology* 21, 1 (2011), Article 4, 25 pages.

[250] Herbert A. Simon and Allen Newell. 1971. Human problem solving: The state of the theory in 1970. *American Psychologist* 26, 2 (1971), 145–159.

[251] Renuka Sindhgatta. 2006. Using an information retrieval system to retrieve source code samples. In *Proceedings of the 28th International Conference on Software Engineering.* ACM, New York, NY, 905–908.

[252] Janice Singer, Timothy Lethbridge, Norman Vinson, and Nicolas Anquetil. 1997. An examination of software engineering work practices. In *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research.* 21.

[253] Raphael Sirres, Tegawendé F. Bissyandé, Dongsun Kim, David Lo, Jacques Klein, Kisub Kim, and Yves Le Traon. 2018. Augmenting and structuring user queries to support efficient free-form code search. *Empirical Software Engineering* 23, 5 (2018), 2622–2654.

[254] Bunyamin Sisman and Avinash C. Kak. 2013. Assisting code search with automatic query reformulation for bug localization. In *Proceedings of the 2013 10th Working Conference on Mining Software Repositories.* 309–318.

[255] Aishwarya Sivaraman, Tianyi Zhang, Guy Van den Broeck, and Miryung Kim. 2019. Active inductive logic programming for code search. In *Proceedings of the 41st International Conference on Software Engineering.* IEEE, Los Alamitos, CA, 292–303.

[256] SourceForge. 2022. Home Page. Retrieved June 26, 2023 from https://sourceforge.net.

[257] Stack Overflow. 2022. Home Page. Retrieved June 26, 2023 from http://stackoverflow.com.

[258] Jamie Starke, Chris Luce, and Jonathan Sillito. 2009. Working with search results. In *Proceedings of the 2009 ICSE Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation.* IEEE, Los Alamitos, CA, 53–56.

[259] Kathryn Stolee and Sebastian Elbaum. 2012. *Solving the Search for Suitable Code: An Initial Implementation.* CSE Technical Reports. University of Nebraska–Lincoln.

[260] Kathryn T. Stolee. 2012. Finding suitable programs: Semantic search with incomplete and lightweight specifications. In *Proceedings of the 34th International Conference on Software Engineering.* IEEE, Los Alamitos, CA, 1571–1574.

[261] Kathryn T. Stolee and Sebastian Elbaum. 2012. Toward semantic search via SMT solver. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering.* ACM, New York, NY, Article 25, 4 pages.

[262] Kathryn T. Stolee, Sebastian Elbaum, and Daniel Dobos. 2014. Solving the search for source code. *ACM Transactions on Software Engineering and Methodology* 23, 3 (June 2014), Article 26, 45 pages.

[263] Kathryn T. Stolee, Sebastian Elbaum, and Matthew B. Dwyer. 2016. Code search with input/output queries. *Journal of Systems and Software* 116 (June 2016), 35–48.

[264] J. Stylos and B. A. Myers. 2006. Mica: A web-search tool for finding API components and examples. In *Proceedings of Visual Languages and Human-Centric Computing (VL/HCC '06).* 195–202.

[265] Fang-Hsiang Su, Jonathan Bell, Kenneth Harvey, Simha Sethumadhavan, Gail Kaiser, and Tony Jebara. 2016. Code relatives: Detecting similarly behaving software. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering.* ACM, New York, NY, 702–714.

[266] Siddharth Subramanian, Laura Inozemtseva, and Reid Holmes. 2014. Live API documentation. In *Proceedings of the 36th International Conference on Software Engineering.* ACM, New York, NY, 643–652.

[267] Zhensu Sun, Yan Liu, Chen Yang, and Yu Qian. 2020. PSCS: A path-based neural model for semantic code search. *arXiv:2008.03042* (2020).

[268] S. Surisetty. 2014. Behavior-based code search. In *Proceedings of the 2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'14).* 197–198.

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

[270] J. Svajlenko and C. K. Roy. 2015. Evaluating clone detection tools with BigCloneBench. In *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME'15).* 131–140.

[271] Peter P. Swire. 2005. A theory of disclosure for security and competitive reasons: Open source, proprietary software, and government systems. *Houston Law Review* 42 (2005), 1333.

[272] Watanabe Takuya and Hidehiko Masuhara. 2011. A spontaneous code recommendation tool based on associative search. In *Proceedings of the 3rd International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation.* ACM, New York, NY, 17–20.

[273] The Apache Software Foundation. 2022. asf–Revision 1910613. Retrieved June 26, 2023 from http://svn.apache.org/repos/asf/httpd/httpd/.

[274] Suresh Thummalapenta and Tao Xie. 2007. Parseweb: A programmer assistant for reusing open source code on the web. In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering.* ACM, New York, NY, 204–213.

[275] Sander Tichelaar. 1999. *FAMIX Java Language Plug-in 1.0.* Technical Report. University of Bern.

[276] Gabriel Valiente. 2002. *Algorithms on Trees and Graphs.* Springer Science & Business Media.

[277] C. Van Rijsbergen. 1979. Information retrieval: Theory and practice. In *Proceedings of the Joint IBM/University of Newcastle upon Tyne Seminar on Data Base Systems*, Vol. 79.

[278] Venkatesh Vinayakarao. 2015. Spotting familiar code snippet structures for program comprehension. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering.* ACM, New York, NY, 1054–1056.

[279] Venkatesh Vinayakarao, Anita Sarma, Rahul Purandare, Shuktika Jain, and Saumya Jain. 2017. ANNE: Improving source code search using entity retrieval approach. In *Proceedings of the 10th ACM International Conference on Web Search and Data Mining.* ACM, New York, NY, 211–220.

[280] Yao Wan, Jingdong Shu, Yulei Sui, Guandong Xu, Zhou Zhao, Jian Wu, and Philip Yu. 2019. Multi-modal attention network learning for semantic source code retrieval. In *Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE'19).* 13–25.

[281] Hao Wang, Jia Zhang, Yingce Xia, Jiang Bian, Chao Zhang, and Tie-Yan Liu. 2020. COSEA: Convolutional code search with layer-wise attention. *arXiv:2010.09520* (2020).

[282] Shaowei Wang, David Lo, and Lingxiao Jiang. 2016. AutoQuery: Automatic construction of dependency queries for code search. *Automated Software Engineering* 23, 3 (2016), 393–425.

[283] S. Wang, D. Lo, and L. Jiang. 2011. Code search via topic-enriched dependence graph matching. In *Proceedings of the 2011 18th Working Conference on Reverse Engineering.* 119–123.

[284] Shaowei Wang, David Lo, and Lingxiao Jiang. 2014. Active code search: Incorporating user feedback to improve code search relevance. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering.* ACM, New York, NY.

[285] Wenhua Wang, Yuqun Zhang, Zhengran Zeng, and Guandong Xu. 2020. TranS^3: A transformer-based framework for unifying code summarization and code search. *arXiv:2003.03238* (2020).

[286] Xiaoyin Wang, David Lo, Jiefeng Cheng, Lu Zhang, Hong Mei, and Jeffrey Xu Yu. 2010. Matching dependence-related queries in the system dependence graph. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE'10).* ACM, New York, NY, 457.

[287] Yuepeng Wang, Yu Feng, Ruben Martins, Arati Kaushik, Isil Dillig, and Steven P. Reiss. 2016. Hunter: Next-generation code reuse for Java. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering.* ACM, New York, NY, 1028–1032.

[288] Yuepeng Wang, Yu Feng, Ruben Martins, Arati Kaushik, Isil Dillig, and Steven P. Reiss. 2016. Hunter: Next-generation code reuse for Java. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering.* ACM, New York, NY, 1028–1032.

[289] GitHub. 2022. Home Page. Retrieved June 26, 2023 from https://www.github.com.

[290] Apache. 2022. Apache Lucene. Retrieved June 26, 2023 from https://lucene.apache.org.

[291] Markus Weimer, Alexandros Karatzoglou, and Marcel Bruch. 2009. Maximum margin matrix factorization for code recommendation. In *Proceedings of the 3rd ACM Conference on Recommender Systems.* ACM, New York, NY, 309–312.

[292] Doug Wightman, Zi Ye, Joel Brandt, and Roel Vertegaal. 2012. SnipMatch: Using source code context to enhance snippet retrieval and parameterization. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology.* ACM, New York, NY, 219–228.

[293] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering.* 1–10.

[294] Huaiguang Wu and Yang Yang. 2019. Code search based on alteration intent. *IEEE Access* 7 (2019), 56796–56802.

[295] Xin Xia, Lingfeng Bao, David Lo, Pavneet Singh Kochhar, Ahmed E. Hassan, and Zhenchang Xing. 2017. What do developers search for on the web? *Empirical Software Engineering* 22, 6 (Dec. 2017), 3149–3185.

[296] Tao Xie and Jian Pei. 2006. MAPO: Mining API usages from open source repositories. In *Proceedings of the 2006 International Workshop on Mining Software Repositories.* ACM, New York, NY, 54–57.

[297] Y. Xie, T. Lin, and H. Xu. 2019. User interface code retrieval: A novel visual-representation-aware approach. *IEEE Access* 7 (2019), 162756–162767.

[298] Jinxi Xu and W. Bruce Croft. 1996. Query expansion using local and global document analysis. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.* ACM, New York, NY, 4–11.

[299] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. 2017. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security.* ACM, New York, NY, 363–376.

[300] Yinxing Xue, Zhengzi Xu, Mahinthan Chandramohan, and Yang Liu. 2019. Accurate and scalable cross-architecture cross-OS binary code search with emulation. *IEEE Transactions on Software Engineering* 45, 11 (2019), 1125–1149.

[301] Shuhan Yan, Hang Yu, Yuting Chen, Beijun Shen, and Lingxiao Jiang. 2020. Are the code snippets what we are searching for? A benchmark and an empirical study on code search with natural-language queries. In *Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution, and Reengineering.* 344–354.

[302] Jinqiu Yang and Lin Tan. 2012. Inferring semantically related words from software context. In *Proceedings of the 2012 9th IEEE Working Conference on Mining Software Repositories.* 161–170.

[303] Jinqiu Yang and Lin Tan. 2013. SWordNet: Inferring semantically related words from software context. *Empirical Software Engineering* 19, 6 (2013), 1856–1886.

[304] Yangrui Yang and Qing Huang. 2017. IECS: Intent-enforced code search via extended Boolean model. *Journal of Intelligent & Fuzzy Systems* 33 (Jan. 2017), 2565–2576.

[305] Ziyu Yao, Jayavardhan Reddy Peddamail, and Huan Sun. 2019. CoaCor: Code annotation for code retrieval with reinforcement learning. In *Proceedings of the 2019 World Wide Web Conference.* ACM, New York, NY, 2203–2214.

[306] Ziyu Yao, Daniel S. Weld, Wei-Peng Chen, and Huan Sun. 2018. StaQC: A systematically mined question-code dataset from Stack Overflow. In *Proceedings of the 2018 World Wide Web Conference.* 1693–1703.

[307] Wei Ye, Rui Xie, Jinglei Zhang, Tianxiang Hu, Xiaoyin Wang, and Shikun Zhang. 2020. Leveraging code generation to improve code retrieval and summarization via dual learning. In *Proceedings of The Web Conference 2020.* ACM, New York, NY, 2309–2319.

[308] Xin Ye, Hui Shen, Xiao Ma, Razvan Bunescu, and Chang Liu. 2016. From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th International Conference on Software Engineering.* ACM, New York, NY, 404–415.

[309] Hang Yin, Zhiyu Sun, Yanchun Sun, and Wenpin Jiao. 2019. A question-driven source code recommendation service based on Stack Overflow. In *Proceedings of the 2019 IEEE World Congress on Services (SERVICES'19).* 358–359.

[310] Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to mine aligned code and natural language pairs from Stack Overflow. In *Proceedings of the 2018 IEEE/ACM 15th International Conference on Mining Software Repositories.* IEEE, Los Alamitos, CA, 476–486.

[311] Alexey Zagalsky, Ohad Barzilay, and Amiram Yehudai. 2012. Example Overflow: Using social media for code recommendation. In *Proceedings of the 2012 3rd International Workshop on Recommendation Systems for Software Engineering (RSSE'12).* 38–42.

[312] Amy Moormann Zaremski and Jeannette M. Wing. 1995. Signature matching: A tool for using software libraries. *ACM Transactions on Software Engineering and Methodology* 4, 2 (1995), 146–170.

[313] Feng Zhang, Haoran Niu, Iman Keivanloo, and Ying Zou. 2018. Expanding queries for code search using semantically related API class-names. *IEEE Transactions on Software Engineering* 44, 11 (2018), 1070–1082.

[314] Jie Zhao and Huan Sun. 2020. Adversarial training for code retrieval with question-description relevance regularization. *arXiv:2010.09803* (2020).

[315] Shufan Zhou, Beijun Shen, and Hao Zhong. 2019. Lancer: Your code tell me what you need. In *Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE'19).* 1202–1205.

[316] S. Zhou, H. Zhong, and B. Shen. 2018. SLAMPA: Recommending code snippets with statistical language model. In *Proceedings of the 2018 25th Asia-Pacific Software Engineering Conference (APSEC'18).* 79–88.

[317] Qun Zou and Changquan Zhang. 2020. Query expansion via learning change sequences. *International Journal of Knowledge-Based and Intelligent Engineering Systems* 24, 2 (2020), 95–105.

[318] Yanzhen Zou, Chunyang Ling, Zeqi Lin, and Bing Xie. 2018. Graph embedding based code search in software project. In *Proceedings of the 10th Asia-Pacific Symposium on Internetware.* ACM, New York, NY, 1–10.

Received 26 April 2022; revised 21 March 2023; accepted 13 April 2023