An Empirical Study of Code Search in Intelligent Coding Assistant: Perceptions, Expectations, and Directions

Chao Liu*
Chongqing University
Chongqing, China
liu.chao@cqu.edu.cn

Xindong Zhang
Alibaba Cloud
Hangzhou, China
zxd139932@alibaba-inc.com

Hongyu Zhang
Chongqing University
Chongqing, China
hyzhang@cqu.edu.cn

Zhiyuan Wan
Zhejiang University
Hangzhou, China
wanzhiyuan@zju.edu.cn

Zhan Huang
Chongqing University
Chongqing, China
huangzhan@cqu.edu.cn

Meng Yan
Chongqing University
Chongqing, China
mengy@cqu.edu.cn

ABSTRACT

Code search plays an important role in enhancing the productivity of software developers. Throughout the years, numerous code search tools have been developed and widely utilized. Many researchers have conducted empirical studies to understand the practical challenges in using web search engines, like Google and Koders, for code search. To understand the latest industrial practice, we conducted a comprehensive empirical investigation into the code search capability of TONGYI Lingma (short for Lingma), an IDE-based coding assistant recently developed by Alibaba Cloud and available to users worldwide. The investigation involved 146,893 code search events from 24,543 users who consented for recording. The quantitative analysis revealed that developers occasionally perform code search as needed, an effective tool should consistently deliver useful results in practice. To gain deeper insights into developers' perceptions and expectations, we surveyed 53 users and interviewed 7 respondents in person. This study yielded many significant findings, such as developers' expectations for a smarter code search tool capable of understanding their search intents within the local programming context in IDE. Based on the findings, we suggest practical directions for code search researchers and practitioners.

CCS CONCEPTS

• **Software and its engineering** → *Software maintenance tools*.

KEYWORDS

Empirical Study, Code Search, Coding Assistant

ACM Reference Format:
Chao Liu, Xindong Zhang, Hongyu Zhang, Zhiyuan Wan, Zhan Huang, and Meng Yan. 2024. An Empirical Study of Code Search in Intelligent Coding Assistant: Perceptions, Expectations, and Directions. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE Companion '24), July 15–19, 2024, Porto de Galinhas, Brazil*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3663529.3663848

*Corresponding author.

FSE Companion '24, July 15–19, 2024, Porto de Galinhas, Brazil

ACM ISBN 979-8-4007-0658-5/24/07
https://doi.org/10.1145/3663529.3663848

1 INTRODUCTION

In modern software development, code search is one of the most frequent activities [9, 31]. The objective of code search tools is to retrieve source code from a large-scale codebase that aligns to the intent of developers' search queries [24, 25]. By utilizing code search tools, developers enhance their development productivity through the reuse of existing code [7, 8, 21, 42].

Over the years, researchers conducted many empirical studies on code search tools [2, 31, 41]. For example, Bajracharya et al. [2] analyzed developers' motivations using the traditional code search engine Koders. Sadowski et al. [31] investigated developers' code search activities at Google with an internal code search engine. Zhang et al. [45] developed a tool named Bing Developer Assistant and provided users' feedback. This tool searches and recommends code example for users to improve their programming productivity. Xia et al. [41] explored the challenges associated with the code search using web search engines like Google. To investigate emerging industrial practices, in this work, we have conducted a comprehensive empirical study on the code search capability of TONGYI Lingma (short for Lingma hereafter), an IDE-based coding assistant recently developed by Alibaba Cloud[1].

This tool, Lingma[2], was developed to improve software development productivity at Alibaba[3], a multinational technology company with over 220k full-time employees. Lingma has been free to users worldwide. The tool provides code search feature and the codebase was sourced from the open-source community GitHub and the question-and-answer (Q&A) communities (e.g., Stack Overflow and Alibaba cloud Documents), where only permissively licensed data was collected. Depending on the type of query, this tool supports two types of code search. One is semantic search, which retrieves code functionally relevant to a given query written in natural language [13, 15, 25, 27]. The other is API search, which takes an API name as input and returns representative code examples [3, 14]. It is noteworthy that code search events can also be triggered by the

[1] Alibaba Cloud website: https://www.alibabacloud.com
[2] Lingma official website: https://tongyi.aliyun.com/lingma
[3] Alibaba website: https://www.alibaba.com

selected programming objects (i.e., code or comments) in the IDE editor. More details can be found in Section 2.1.

To examine the practical use of the code search tool, we have conducted a quantitative analysis on one year of recorded code search activities, obtained with user consent. These recordings involve 24,543 users and 146,893 code search events. The statistical results revealed several key insights:

- Semantic search is more frequently used within the IDE, compared to API search. The majority of developers perform code search as needed, an effective tool should consistently yield useful results in practice.
- User queries tend to be short and concise, and recommending relevant queries based on common user inputs is useful to pinpoint the desired queries. Developers occasionally submit lengthy queries by copying runtime logs for debugging purposes. For semantic search queries, inclusion of local language (e.g., Chinese) proves advantageous for non-English speaking developers.
- Developers commonly restrict their explorations to top rank (less than 3) of the first page of code search results. This behavior underscores the critical need for code search tools to deliver highly relevant results.

Moreover, we developed a survey questionnaire to investigate users' perceptions and future expectations regarding code search. The questionnaire was drafted based on the understanding of quantitative analysis and recent systematic reviews of code search studies [9, 24]. It underwent refinement through a pilot study involving two senior developers at Alibaba. Additionally, we conducted one-on-one interviews with seven respondents to delve deeper into their survey responses and gather additional insights. Some major findings include:

- Experienced developers may not perform frequent code search; but when they do, they expect the tool to be highly effective. Despite their experience, writing precise queries remains a challenge for semantic search users.
- In practice, developers search code for diverse purposes (e.g., program comprehension). They anticipate future tools to incorporate codebases with multiple programming languages (e.g., Go and Javascript) from various sources (e.g., API documents). These tools should support search at different code granularity levels (e.g., fragment and class) and offer additional quality improvement of the search results (e.g., bug detection and programming style checking).
- Developers expect code search tools to possess enhanced query understanding capabilities like ChatGPT. They expect search results to be succinct, representative, readily comprehensible, and adaptable to the programming context with minimal or no modifications.

In summary, our study makes the following major contributions:

- We performed a comprehensive quantitative analysis of a recent coding assistant to analyze developers' code search activities. Subsequently, we conducted surveys and interviews to uncover valuable insights.

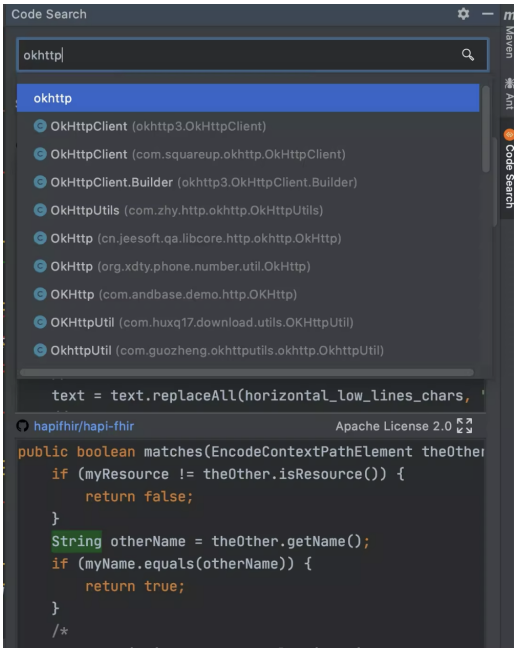

Figure 1: The interface of the code search feature in Lingma.

- We presented future directions for researchers concerning code search requirements, evaluation, and improvement. Additionally, we showed various user expectations that can assist practitioners in developing better tools.

2 METHODOLOGY

Section 2.1 describes the study subject of this paper. Section 2.2 presents the overview of our research methodology in two stages, each of which is elaborated in Sections 2.3 and 2.4, respectively.

2.1 Study Subject

Lingma is an AI coding assistant developed by Alibaba Cloud, providing plugins to JetBrains IDEs, Visual Studio Code, etc. Originally published on Oct 10th, 2021, under the name Alibaba Cloud AI Coding Assistant[4], the tool underwent a name change to Lingma on Oct 25th, 2023 with additional features such as code comment generation [16, 17], code generation [6, 38], test case generation [20, 39], bug fixing [36, 44], etc. Primarily, the code search feature of this tool leverages the pre-trained language model GPT-2 [30] and the distributed full-text search engine ElasticSearch [10].

This tool enhances software developers' productivity by providing code search capabilities directly within the integrated development environment (IDE), eliminating the need to switch browsers when encountering programming challenges. The user interface of the code search tool, depicted in Fig. 1, allows users to input textual descriptions or API names in the search box at the top. Additionally, the tool offers recommendations for candidate queries (e.g., "OkHttpClient") based on users' input (e.g., "okhttp"), facilitating

[4]Alibaba Cloud AI Coding Assistant website: https://github.com/alibaba-cloud-toolkit/cosy

more accurate queries. Upon entering a query, the tool retrieves a ranked list of relevant Java source code from a codebase curated by the Lingma development group. If users finds a result of interest, they can click on it to view contextual information associated with the code snippet.

The tool involves two types of code search methods depending on the query form: semantic search [13] and API search [14]. Semantic search involves queries described in natural language, such as "read file". Researchers indicated that the challenge of aligning the semantic gap between the natural language queries and source code [13, 24]. To address this issue, previous studies investigated IR-based techniques, which offer fast results but rely heavily on user keywords for accuracy [25, 27]. More recent studies have focused on the DL-based techniques, which enhance query understanding but sacrifice speed [5, 12, 13, 15, 37]. This tool addressed the challenge by integrating the strengths of both techniques.

The API search operates like the example depicted in Fig. 1. Users can input queries in the format of "package name + class/interface name + method name" (e.g., java.lang. StringBuilder) or "package name + enumeration name + attribute name" (e.g., Calendar.MONTH). Partial API names (e.g., java.lang or StringBuilder) are also acceptable as input. API search assists developers in finding relevant examples for a given API name. However, direct matching may yield numerous irrelevant and duplicate results [14, 24]. This area has been extensively explored by researchers [3, 4, 19, 29, 40]. To address this challenge, the tool leverages techniques such as result clustering and ranking following the strategies proposed in prior studies.

It is worth noting that the tool offers code search functionality within the programming context. Users can select any code snippet in the editor and access the code search option from the right-click menu. The tool automatically recognizes programming objects, such as comments, function names, etc. If users select a comment, the tool will perform a semantic search based on the natural language description. Otherwise, it will conduct an API search.

2.2 Overview

Fig. 2 illustrates an overview of our research methodology, which comprises two stages. **Stage I** involves quantitative analysis. We collected historical data from Lingma users who granted the recordings and investigated the code search activities in practice. This analysis informed the survey design in Stage II. More details can be found in Section 2.3. **Stage II** encompasses survey and interview. We crafted a survey questionnaire to investigate users' perceptions and future expectations regarding code search. Moreover, we conducted interviews with some respondents to clarify their survey responses as illustrated in Section 2.4.

2.3 Qualitative Analysis of Code Search in Practice

2.3.1 Process. Lingma records code search interactions for users who consent to the data collection, storing the information in a database. The second author, an Alibaba employee, is allowed to collect one year of code search events from April 14, 2022 to April 14, 2023. Subsequently, the data underwent preprocessing and key statistics were summarized. Following enterprise regulations, these

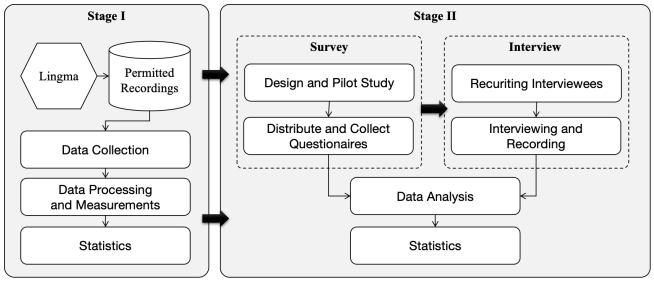

Figure 2: Overview of the methodology.

statistics and results can be shared with other authors and made publicly available after being scrutinized by the enterprise.

2.3.2 Data Collection. The data collected for each code search event, formed with query and search results, comprised several elements. Firstly, we captured the user ID (i.e., anonymous machine ID) from the database to identify the user initiating the query. Secondly, we analyzed the language employed in the query, such as English and Chinese. Thirdly, we counted the query words, excluding punctuation and other symbols. Additionally, we recorded the type of code search performed, namely semantic search or API search. For the corresponding search result, we tracked the number of pages viewed by users and recorded the rank of results that users found interesting, namely the instances where they clicked to view details or selected code for use (e.g., copy and paste). Detailed statistics derived from this data can be found in Sections 3.1.1-3.1.3, respectively.

2.3.3 Measurements. To investigate users preferences, we calculated three key metrics for all code search events as detailed in Table 5. Firstly, the Success Rate represents the percentage of events that users found interesting. By comparing semantic search and API search, we can know which type is more commonly favored. Secondly, the FRank measures the first result that captured users' interest in the result list, indicating users' effort in scanning the candidate list from top to bottom [18]. A lower FRank indicates fewer efforts and better effectiveness of a code search type, namely semantic or API search. Lastly, the MRR (Mean Reciprocal Rank) calculates the average of the reciprocal ranks for all queries, where the reciprocal rank is the inverse of FRank [25]. A larger MRR signifies a higher ranking for the first interested result.

2.3.4 Research Questions. Based on the above data and measurements, the quantitative analysis aims to investigate the following three research questions (RQs):

- *RQ1: Which code search task is frequently used?* This question seeks to determine developers' preferences regarding code search types: semantic search or API search.
- *RQ2: What are the characteristics of queries?* This question examines user queries, focusing on word count and language, and analyzes the difference between two code search types.
- *RQ3: How many search results are commonly viewed by developers?* This question aims to quantify the average number

of search results that developers reviewed and evaluate the effectiveness of different code search types.

2.4 Survey and Interview on Users' Perceptions and Expectations

2.4.1 Process. Our goal is to bridge the gap between research and industrial communities and enhance their comprehension on code search users' perceptions and expectations in practice. To reach the goal, the first author, a code search researcher, drafted a survey comprising initial questions based on two systematic reviews on code search [9, 24]. These questions underwent review and refinement by the fourth and last authors.

Subsequently, the second author invited two Alibaba users to conduct a pilot study. These user are experienced developers with over 12 years of experience in the software industry. Drawing from their insights, we refined the questionnaire by merging redundant questions and options, clarifying statements, and re-ordering the list. Following their suggestions, we removed all open-ended questions concerning respondents' rationales for selecting specific option(s) for each question. However, we retained one open-ended question regarding the comparison between the code search and ChatGPT[5], a globally recognized tool that generates code examples based on user descriptions, provided by OpenAI. This comparison is valuable for refining code search practices from users' perspective, even though ChatGPT does not directly search code [22]. This open-ended question was placed at last and made optional for submission, allowing respondents who did not use ChatGPT or preferred not to answer. The final survey, comprising 12 questions, is presented in Section 2.4.2.

The second author distributed an online survey link within an internal chat group comprising 199 users. These individuals are active Lingma users from various departments within Alibaba. This group was selected due to its representation of the tool's primary user base at Alibaba and their active involvement in providing feedback and suggestions for Lingma. Since the tool does not collect users' contact information for privacy considerations, this group served as the most accessible population for our study. Totally, 53 respondents, including the two participants from the pilot study, completed the survey.

To gain deeper insights into respondents' perceptions and expectations, we conducted interviews. Specifically, the second author enlisted 7 respondents from Alibaba, including the two developers involved in the pilot study. To save interviewees' time, the second author engaged them in one-on-one discussions, inviting them to elaborate on their survey responses and provide additional insights. These interviews aimed to make up for the removed open-ended questions in the survey and required approximately 20 minutes per participant.

2.4.2 Questionaire and Research Questions. Table 1 presents the 12 questions (Q1-Q12) from our survey, categorized into two parts. The first part (Q1-Q5) pertains to users' code search practices, while the second part (Q6-Q12) addresses their expectations. Questions 1 and 2 inquire about developers' demographics. Among these questions, the 1st, 3rd, and 5th are single-choice questions, the last is an open question optional to answer, and the rest are multi-choice questions. These questions correspond to the following research questions (RQs).

- *RQ4: What are developers' perceptions on the code search in practice?* Assessing users' opinion on the practical performance of code search (Q3-Q5).

[5]ChatGPT provided by OpenAI: https://chat.openai.com

Table 1: Survey questionnaire.

Note: 1, 3, and 5 are single-choice questions; 12 is an open question; the others are multi-choice questions.

Part I: Code Search Practice.

1. How many years have you been software industry? _____
(a) 0~3; (b) 3~5; (c) 5~10; (d) >10.

2. What is the role in your team? _____
(a) developer; (b) tester; (c) architect; (d) program manager; (e) team leader; (f) other: _____.

3. When you are programming software, what is the average number of code search activities you need per day?_____
(a) 0~2; (b) 3~5; (c) 6~10; (d) 11~15; (e) >15

4. Which code search type in Lingma do you often use? _____
(a) Semantic search: using keywords to search related code implementation; (b) API search: using API name to search related code examples.

5. How do you feel about the usefulness of two types of code search? Semantic Search: _____; API search: _____
(a) Very High, often returns many useful results; (b) High, often returns some useful results; (c) Moderate, often returns one or two useful results; (d) Low, occasionally returns one or two useful results; (e) Very Low, difficult to find useful results.

Part II: Code Search Expectations

6. If you have an ideal code search tool, what would be your purpose of code search? _____
(a) understanding code; (b) implementing requirement; (c) refactoring code; (e) optimizing code; (f) fixing bugs; (g) removing vulnerability issues; (h) not sure; (i) other: _____

7. For the ideal code search tool, what additional features do you expect it to have? _____
(a) voice search: searching code according to human voice; (b) sketch search: searching code for a sketched user interface figure; (c) I/O search: searching code with its input and output examples; (d) finding similar code: searching a code with similar functionality for a given code; (e) not sure; (f) other: _____

8. Which programming language(s) do you expect to add for the code search other than Java? _____
(a) Python; (b) JavaScript/TypeScript; (c) C/C++; (d) Go; (e) PHP; (f) Ruby; (g) C#; (h) Rust; (i) not sure; (j) other: _____

9. For the ideal tool, which data source do you expect to add? _____
(a) open source community (e.g., SourceForge and Gitee); (b) App Community (e.g., FDroid); (c) Q&A community (e.g., Stack Overflow); (d) web search engine (e.g., Google and being); (e) official API documents; (f) programming videos (e.g., Youtube and Bilibili); (g) vulnerability database (e.g., CVE); (h) local repository; (i) not sure; (j) other: _____

10. Which code granularity or target do you expect to search?_____
(a) code snippet (or fragment); (b) code function (or method); (c) class; (d) file; (e) repository; (f) not sure; (g) other: _____

11. Which aspect(s) do you expect to improve the quality of the retrieved code?_____
(a) improving programming style; (b) checking code defects; (c) checking vulnerability issues; (e) checking sensitive info (e.g., email and secret key); (f) associating relating test case(s); (g) calculating test coverage; (h) providing testing failure info; (i) not sure; (j) other: _____

12. Do you think what are the pros and cons of ChatGPT compared to the code search feature in Lingma? (Optional)
_____

- *RQ5: What are developers' purposes of code search?* Understanding the practical reasons behind developers' utilization of code search tools (Q6).
- *RQ6: What are developers' expected code search inputs?* Investigating the preferred formats for code search inputs (Q7).
- *RQ7: What are developers' expected extensions on the codebase?* Providing insights into potential expansions of the codebase regarding programming language (Q8) and data source (Q9).
- *RQ8: What are developers' expected search results?* Understanding the desired formats (Q10) and enhancements (Q11) of the searched results.
- *RQ9: Code Search vs. ChatGPT.* Analyzing the distinctions between the surveyed code search tool and the widely-used ChatGPT (Q12) as a basis for future research.

2.4.3 Data Analysis. After collecting responses to the open question (Q12) and interviews, we proceeded to organize these descriptions via open and axial coding [1]. Specifically, the first and second authors divided the responses into distinct points with corresponding codes and connected the codes into categories. We discussed the results together with the questionnaires in Section 3.2. As per enterprise regulations, the statistics and conclusive results can be disclosed to the remaining authors and made public after being scrutinized by the enterprise.

2.4.4 Demographics of Respondents. Our survey included a total of 53 respondents. Fig. 3 shows that 8 respondents (15.09%) possess less than 3 years of experience, while 15 respondents (28.30%) are relatively new software engineering practitioners with 3~5 years of experience. Besides, 23 respondents (43.40%) report having 5~10 years of experience, and 7 respondents (13.21%) claim over 10 years of software engineering experience. In summary, 84.91% of the respondents are experienced practitioners with over three years of experience in the software industry.

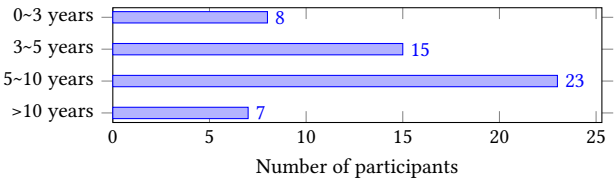

Figure 3: (Q1) Number of participants with different years of experience in software industry.

Fig. 4 summarized the respondents' role in their software development teams. Among these respondents, 50 individuals (94.34%) identified themselves as developers, because code search users should *"do some programming work more or less"*. Moreover, 35 respondents (66.04%) chose the role of the architect as they are commonly *"responsible for a software component and its architecture design"*. Besides, 11 respondents were project managers (13.21%) or team leaders (7.55%). The interviewees indicated that these two roles are *"part of jobs assigned by the team"* and their prominent roles are still developer or architect. Additionally, only 3 respondents (5.66%) recognized themselves as testers because *"developers are responsible for testing their own code"* as explained by the interviewees.

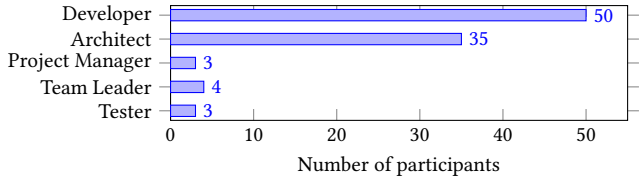

Figure 4: (Q2) Number of participants played different role(s) in their software development teams.

3 RESULTS AND DISCUSSION

We provided the quantitative analysis on code search in practice in Section 3.1, presented the results of survey and interview in Section 3.2, and finally discussed the future directions in Section 3.3.

3.1 Code Search in Practices

3.1.1 RQ1: Which code search task is frequently used? Table 2 presents insights from 24,543 users who consented to the recording of their code search activities. Of these users, 20,861 (85.0%) engaged in semantic search while 13,413 (54.7%) users utilized API search to find code examples. Notably, 9,731 (39.6%) users participated in both types of code search. These results indicated that many users would like to use two types of code search in a mixed manner with a preference for semantic search over API search. These users collectively generated 146,893 code search events, with 104,614 (71.2%) events attributed to semantic search and 42,279 (18.8%) to API search. On average, there were 402.45 code search requests per day, with 286.61 and 115.83 events dedicated to semantic and API search, respectively. However, for each user, only 0.02 code search events occurred per day. This suggests that while semantic search is more frequently used, users utilize the code search tool sporadically. When the tool fails to meet their needs, they would turn to alternative methods such as web search engine [41]. Therefore, developers would perceive a code search tool as valuable when it consistently resolves their issues in practice.

Table 2: Statistics of code search events for semantic search and API search tasks, where "#Users" and "#Events" indicate the number of users and code search events, respectively.

Statistics	Semantic Search	API Search	Total
#Users	20,861	13,413	24,543
#Events	104,614	42,279	146,893
#Events per day	286.61	115.83	402.45
#Events per day per user	0.01	0.01	0.02

Finding 1: *Semantic search shows a higher usage rate compared to API search, with 40% of users utilizing both together. As developers conduct code search as needed, an effective tool should consistently yield useful results in practice.*

3.1.2 RQ2: What are the characteristics of queries? Table 3 illustrates that the mean word count across all code search queries is

48.21. Specifically, the mean word count for semantic search queries surpasses that of API search (52.96 vs. 37.91). The word count distributions for both tasks exhibit significant difference (p-value<0.05) as determined by the two-sample Kolmogorov-Smirnov test [28] at a 5% significance level. However, the median counts for semantic search and API search show an inverse size relationship (16 vs. 36). These results imply that, in most cases, semantic search queries tend to be shorter than API search queries. As shown in Fig. 1, API search queries tend to be longer because developers often opt for candidate APIs with fully qualified names recommended by the tool. The recommendation is helpful for users to pinpoint target APIs. Additionally, it is widely acknowledged that semantic search queries are concise [2, 41], often composed of verb phrases [25]. We observed that developers may input lengthy queries by selecting recommended candidate queries from the tool or by copying runtime logs to find the code example for debugging purposes. Consequently, developers not only perceive the semantic search as a coding assistant, as per existing studies [18], but also expect it to serve as a debugging assistant.

Table 3: Word count statistics for users' queries in both semantic search and API search tasks.

Word Count	Semantic Search	API Search	Total
Mean	52.96	37.91	48.21
Minimum	1	1	1
Maximum	16,779	481	16,779
Median	16	36	22

Table 4 shows that 74.7% of queries are written in English, while 13.0% consist solely of local language (i.e., Chinese) words, and 12.3% involve a combination of both languages. It is common for API search users to input API names directly in English. In contrast, the semantic search involved 19,124 Chinese queries and 17,978 queries with a mix of languages. This indicates that local language may serve as auxiliary when users struggle to articulate their requirements clearly in English.

Table 4: The linguistic composition of code search queries used in semantic search and API search tasks.

Language	Semantic Search	API Search	Total
English	67,512 (64.5%)	42,248 (99.9%)	109,760 (74.7%)
Local	19,124 (18.3%)	2 (0.0%)	19,126 (13.0%)
Combined	17,978 (17.2%)	29 (0.1%)	18,007 (12.3%)
Total	104,614	42,279	146,893

Finding 2: *User queries tend to be short and concise; however, users might submit lengthy queries by opting for those recommended by the tool or by copying runtime logs for debugging purposes; in semantic search, the local language (e.g., Chinese) could be beneficial for users who are not proficient in English.*

3.1.3 RQ3: How many search results are commonly viewed by developers? Table 5 reveals a low success rate of code search (0.21). Both semantic and API search show similar outcomes without statistical differences (p-value<0.05), as per the two-sample Kolmogorov-Smirnov test [28] at a 5% significance level. This is primarily because developers typically review the top search results only (#Viewed<14), and the number of results they find interesting is often less than 2 (#Interested<2). These results corroborate existing assumption in code search studies [9, 13, 18, 34]. In our study, the performance of the code search tool is low with an MRR of 0.16. There is no statistical difference (p-values<0.05) between semantic and API search in terms of the two-sample Kolmogorov-Smirnov test [28]. However, for search results that users find interesting, the MRR is notably higher at 0.77 with a mean FRank of 2.29. These results suggest that developers are inclined to click on search results they find interesting, especially if they appear in the top three ranks, to explore details or select them for copying and pasting. However, in many cases, search results cannot be directly applied to meet users' implementation requirements. Additionally, developers may choose to write code based on the useful search results or express interest in certain code without taking further action. Thus, measuring the actual performance of code search remains challenging.

Table 5: Performance of the code search tool, where "#Viewed" and "#Interested" represent the number of results viewed by users or the ones that users find interesting, respectively; "MRR/Mean FRank for Interested" reflects the tool's performance specifically on the search results that users find interesting.

Result	Semantic Search	API Search	Total
Success Rate	0.20	0.21	0.21
#Viewed	13.97	13.06	13.45
#Interested	1.39	1.31	1.34
MRR	0.17	0.15	0.16
MRR for Interested	0.78	0.77	0.77
Mean FRank for Interested	2.34	2.25	2.29

Finding 3: *Users typically limit their exploration to the first page of code search results, with interesting findings often located around the second rank.*

3.2 Users' Perceptions and Expectations

3.2.1 RQ4: What are developers' perceptions on the code search in practice? Fig. 5 illustrates that 9 (16.98%) participants performed code search infrequently (0~2 times), while 28 (52.83%) participants demonstrated a medium frequency conducting code search 3~5 times. The remaining 3 participants conducted code search with a high frequency (11~15 times) or even greater (>15 times). According to the interviewees' feedback, they *"used code search tools only when they needed"* and they are *"very familiar with their coding context"*. This trend may be attributed to the respondents' extensive experience as highlighted in Section 3.1. Consequently, the average code search events per day per user remains low as discussed in Section

3.1.1. These results suggest that novice developers may generate more code search events, while experienced developers in software enterprises prioritize their code search needs and satisfaction levels over frequency.

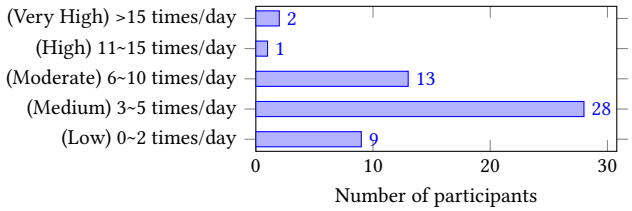

Figure 5: (Q3) Number of participants engaging in code search with different frequencies per day, which are categorized into five levels: low, medium, moderate, high, and very high.

The statistical results from Q4 show that 48 (90.57%) respondents frequently utilized the semantic search, while about 64.15% of respondents employed API search. These results suggest a significant preference among users for both types of code search in software programming, which is also confirmed by the quantitative analysis in Section 3.1 and feedback received from interviewees.

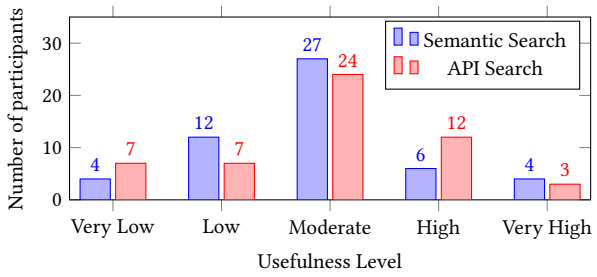

Figure 6: (Q5) Number of participants who scored the usefulness levels (i.e., very low, low, moderate, high, and very high) for semantic search and API search.

Fig. 6 lists the usefulness levels as reported by respondents. It indicates that 27 respondents (50.94%) perceived semantic search to be moderately useful, while 12 respondents (22.64%) rated its usefulness as low. Additionally, 10 respondents found semantic search to be highly useful. The remaining 4 respondents (7.55%) claimed a very low usefulness. In API search, the majority of respondents (45.28%) also rated its usefulness as moderate, with 13 respondents indicating high usefulness. However, 14 respondents (26.42%) rated API search as low or very low in usefulness. These results suggest that API search is perceived as more useful than semantic search, despite the latter being utilized by more users, as described in Section 3.2.1. Several interviewees noted that *"describing query accurately is difficult for semantic search compared to the API search"*.

Finding 4: *Experienced developers may not require frequent code search; however, when they do, a tool should prove highly effective. Semantic search, although more frequently utilized than API search, shows a lower level of usefulness due to the challenge of generating accurate queries.*

3.2.2 RQ5: What are developers' purposes of code search? Fig. 7 provided several purposes of code search. The primary purpose, chosen by 47 (88.68%) respondents, is implementing requirements. Many interviewees indicated that *"although some search results are useful"* they *"commonly write the code"* by themselves. This feedback suggest a gap between code search and developers' coding processes, potentially contributing to the moderate usefulness level reported in Section 3.2.1 and explaining why many users take no further actions for the searched results as described in Section 3.1.3. Moreover, 45 respondents (84.91%) anticipate understanding code examples using the tool, such as learning how to use specific APIs. However, one interviewee noted that *"sometimes a code is very long it is difficult to know which part could be useful, thus this result could be skipped"*. Therefore, returning concise and representative code examples is crucial for the effectiveness of the code search tool.

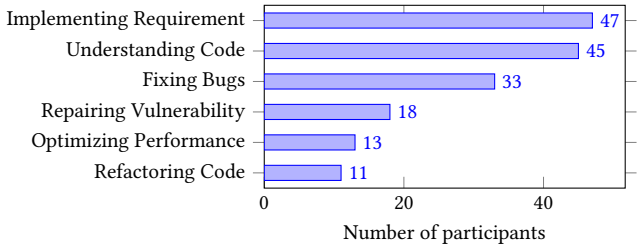

Figure 7: (Q6) Number of participants who search code for various purposes.

Subsequently, 33 respondents (62.26%) indicated a preference for using a code search tool to find solutions for programming bugs. According to interviewees' feedback, when encountering programming errors, they typically *"copy error information or related source code to the search engine and try to find some solutions"*. This practice may contribute to the lengthy search queries, as analyzed in Section 3.1.2. However, existing code search tools may not always be suitable for this type of query. Besides, 18 respondents (33.96%) expected code search results to help them repair vulnerabilities. Lastly, 13 and 11 respondents anticipated that an ideal code search tool could assist them in optimizing (24.53%) or refactoring (20.75%) the returned code. Nonetheless, some experienced interviewees pointed out that *"I would like to do the optimization or refactoring by myself"* This is because 1) *"they are what I am good at"*; 2) and *"I will decide whether these tasks are urgently required for now"*.

Finding 5: *Developers want to leverage code search tools to aid in implementing functional requirements, understanding programming examples, and fixing bugs or vulnerabilities. Throughout their search process, they anticipated that the retrieved code*

will be concise and representative, directly applicable to their programming context.

3.2.3 RQ6: What are developers' expected code search inputs? Fig. 8 lists five types of code search methods not included in Lingma. It reveals that 33 respondents (62.26%) express interest in code clone search, which involves inputting a code snippet to retrieve similar code within a codebase. Additionally, 22 respondents (37.74%) desire to search a target source code based on a given test case. Besides, 13 respondents (24.53%) anticipate searching code using a sketch of the user interface. The remaining 18 respondents hope for an ideal code search tool that works based on a pair of input and output examples (15.09%) or human voice (5.66%). According to interviewees, code clone search is valuable because it enables them to *"find some duplicated code snippets for the same modifications"*. However, many interviewees raised doubts on the usefulness of other code search methods because they *"cannot come up with any cases to use them"*.

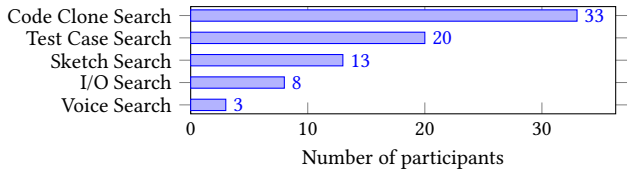

Figure 8: (Q7) Number of participants who expected other code search methods.

Finding 6: *Developers favor code clone search as an additional method alongside the semantic search and API search in Lingma.*

3.2.4 RQ7: What are developers' expected extensions on the codebase? Fig. 9 illustrates that many respondents advocate for the expansion of Java in code search to include three other programming languages: Go (77.4%), JS/TS (62.62%), and Python (75.5%). There are 13 votes each for C/C++, Ruby, and C#. Moreover, PHP and Rust received 13 other selections. These results underscore the diverse usage of programming languages by developers in practice. All interviewees emphasized the necessity of this extension because they *"need to use it"*. Furthermore, one interviewee highlighted the usefulness of extending Rust because *"this programming language is new to them"*.

Table 10 shows that over 33 respondents express interest in incorporating additional search sources from other open source communities (86.79%), Q&A community (66.04%), and official API documents (62.62%). These sources align with developers' common search practices as described by interviewees. Additionally, 20 respondents (37.74%) selected web search, with interviewees noting that *"the web search involved many other sources"*. 14 respondents (26.42%) found code search in local repository useful. One interviewee indicated that *"I want to know how a requirement is implemented in local repositories at enterprise if I have the authority"*. Moreover, 10, 3, and 2 respondents (18.87%) opted for vulnerability database

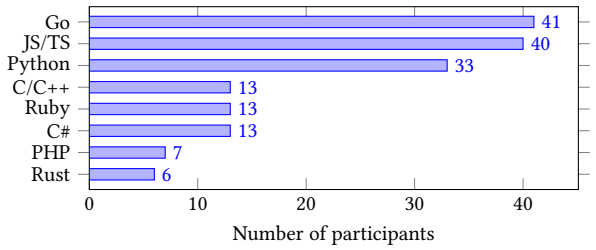

Figure 9: (Q8) Number of participants anticipating tool support for programming languages beyond Java.

(18.87%), APP community (5.66%), and programming video websites (3.77%) as additional sources, respectively.

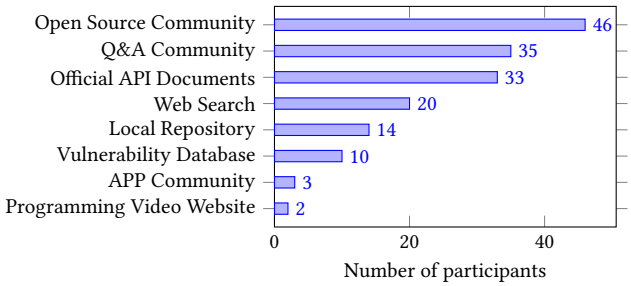

Figure 10: (Q9) Number of participants who expected other search sources.

Finding 7: *Developers anticipate expanding the current code search tool to including additional programming languages such as Go, JS/TS, and Python, as well as integrating more search sources such as other open source communities, Q&A community, official API documents, and local repositories.*

3.2.5 RQ8: What are developers' expected search results? Fig. 11 indicates that the majority of respondents (>88%) express interest in searching code functions or fragments. Interviewees noted the usefulness of these two types because their *"requirements are often small and specific"* and they *"would like to decompose the requirement"* by themselves. Besides, 27 respondents (50.94%) found searching classes useful, while 14 other respondents preferred searching repositories (22.64%) or files (3.77%). These results underscore the developers' expectations for searching higher granularity of code targets.

Fig. 12 ranked 7 types of quality improvements expected for code search. Ranked in descending order, 47 respondents (88.68%) express a need for a tool capable of checking bugs from the returned code within the developers' programming context. Additionally, 39 respondents (73.58%) require the returned code to conform to their expected programming style. Moreover, 24 respondents (37.74%) seek related test cases in the search space for the retrieved source code. 15 respondents (28.30%) are concerned about sensitive information in code and wish to remove them, such as secret key and

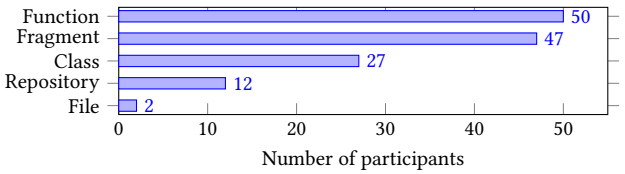

Figure 11: (Q10) Number of participants anticipating additional search granularities.

email address. Another 13 respondents (24.53%) require the future tool to present the testing coverage for the searched code, while an equal number of respondents (24.53%) want to know related testing failure information. According to interviewees, the ranking is natural. They emphasized that 1) detecting bug is crucial because *"code search can help write code but the code may not work in IDE"* and they prefer not to *"debug the searched code and save effort"*; 2) *"the code is required to follow the programming style at enterprise"*; 3) *"the others look useful but they are not the first priority"*.

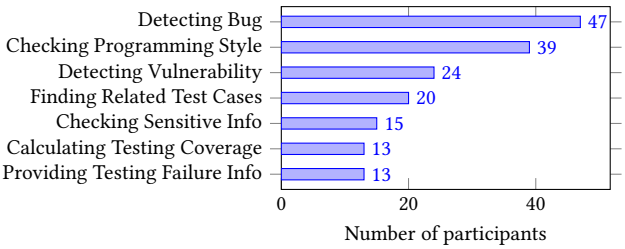

Figure 12: (Q11) Number of participants anticipating additional quality improvements for the retrieved code.

> **Finding 8:** *Developers prefer breaking down their requirements into specific tasks and search for code at the level of functions or fragments. They anticipate that an ideal tool can enhance code quality by detecting potential bugs and vulnerabilities, transforming programming style, etc.*

3.2.6 RQ9: Code Search vs. ChatGPT. Table 6 presents respondents' positive and negative comments comparing the code search feature in Lingma and the well-known AI language model ChatGPT. In summary, the primary advantages of code search include its fast response time and the retrieval of human-written code, in contrast to ChatGPT. However, code search faces challenges in addressing complex queries. Consequently, developers often need to refine their queries based on search results, which can be tedious. Moreover, assessing the relevance of expected results is cumbersome due to the presence of many duplicate results in the search list. These observations are consistent with feedback from interviewees. Besides, one interviewee mentioned that *"they used the code search tool when they knew the right keywords and had the confidence in finding the expected code"*.

On the contrary, ChatGPT excels in understanding queries and the generated code is often satisfactory in two key aspects: 1) it produces well-organized code with clear logic, which typically requires minimal modifications for usability; 2) it also generate comments to help understand the functionality of code. However, ChatGPT's performance is influenced by two factors. Firstly, its knowledge boundary limits the its ability to consistently generate accurate answers, as some respondents noted that ChatGPT cannot always provide the correct responses. The interviewees assumed the reason could be that *"the scope of training data is limited"*. Secondly, ChatGPT's performance depends on the prompt provided as input. The generated code may be sensitive to the prompt description or include unexpected APIs. Overall, interviewees favored ChatGPT over code search, acknowledging its strengths. However, they also commented that: 1) *"ChatGPT cannot work on internal enterprise data"* because a company may write code using the customized framework or APIs; 2) *"ChatGPT are not responsible for each line of code"* so that developers still need to review code carefully.

Table 6: (Q12) Comparison of the code search (CS) feature in Lingma and the well-known AI language model ChatGPT

Type	Aspect	Respondents' Comments
CS	Pros	1. Returning the real code written by humans. 2. Searching code is fast.
	Cons	1. Difficult to understand complex semantic queries. 2. Frequently reformulating queries according to search results. 3. Too many duplicated search results. 4. The search just narrows down the scope. 5. It is tedious to filter out the expected results.
ChatGPT	Pros	1. Capable of understanding programming context. 2. The generated code can be directly used with little modifications. 3. It generated detailed comments on the code for easier understanding. 4. The generated answer is well-organized with clear logic.
	Cons	1. Limited in asking questions related to some special resources. 2. The generation is sensitive to the input. 3. Some minor changes to input still lead to the same generation. 4. The generated API could be wrong which requires more re-generations.

> **Finding 9:** *Code search offers quick responses to user queries and help developers find human-written code within a vast codebase, but users often need to refine their queries. ChatGPT is user-friendly as it adeptly grasps user intent and produce better results, but its generation is confined by input prompts and the available training data.*

3.3 Implications for Future Directions

Based on our empirical findings, we present the following potential directions for both researchers and practitioners.

For Researchers. The semantic search has gained significant attention in recent years [9, 25], with numerous proposed models [11, 32, 33]. The CodeSearchNet benchmark [18] is commonly used for evaluation, focusing on the functional relevance between

searched code and semantic queries. However, our observations in Sections 3.1.1 and 3.2.2 indicated diverse code search purposes. Investigating how the code search contributes to the entire software development lifecycle is crucial, as functional relevance alone may not suffice for all practical search requirements, as demonstrated in Section 3.1.3. By understanding specific search requirements, researchers can develop better benchmarks and models to bridge academia and industry. Additionally, existing benchmarks are often small in size [24] because manually determining all the relevancy of ground truth is impractical. This necessitates the design of appropriate test cases for automated validation like the HumanEval benchmark for code generation [6]. Furthermore, our study reveals that search results are often unsuitable for direct coding use, prompting exploration into the ways of adapting search results to expected coding context, as noted in Section 3.2.6.

For Practitioners. Our empirical findings showed that developers have many expectations for advancing code search tools in practice, encompassing support for additional code search types like code clone search (Section 3.2.3), other programming languages (Section 3.2.4), quality improvements of the searched code (Section 3.2.5), and etc. Addressing these diverse expectations entails substantial engineering efforts. This study offers a framework framework and prioritization of various expectations, aiding practitioners in the industry in effectively meeting their needs.

4 RELATED WORK

Many empirical studies investigate the effectiveness of existing code search tools and analyze the influential factors. Previously, Sim et al. [35] examined the support provided by five websites (including Google, Koders, etc.) for developers' code search needs. They found that code-specific search engines performed better in searches for subsystems, but Google was more effective in searching code blocks. Bajracharya et al. [2] analyzed patterns in search engine logs from the traditional code search engine Koders to gain insights into programmers' methods and motivations during code search activities. Sadowski et al. [31] investigated developers' behaviors and needs when searching code on Google using search logs and surveys. They found that code search has become deeply entrenched in software development, with developers frequently conducting searches primarily focused on familiar code and specific code locations. Almost a third of searches involved incremental query reformulation. Most queries are scoped to a subset of the code repository, and search sessions are typically short. Subsequently, Xia et al. [41] investigated developers' code search activities using Google and analyzed their difficulties in code search. Liu et al. [23] analyzed the code search intents for deep learning programming.

Yan et al. [43] constructed a benchmark with search ground truth to evaluate six code search methods. The empirical findings demonstrated the benchmark's utility and revealed that deep learning (DL) methods are more effective for queries focused on reusing code, whereas information retrieval (IR) methods excel for queries focused on resolving bugs and learning API usage. Zhang et al. [46] compared two IR-based methods using the CodeSearchNet corpus [18], demonstrating that IR-based methods outperform several pre-BERT neural models in multiple aspects. Liu et al. [26] conducted a survey involving over 100 software developers, revealing that a considerable percentage of developers had never used code search engines and were unaware of their existence. Different from prior studies, we conducted an empirical study on the code search feature of a recent intelligent coding assistant Lingma, developed by Alibaba Cloud. Additionally we conducted survey and interview with tool users to gain insights into their perceptions and expectations. This investigation also provides guidance for future research directions in code search studies.

5 THREATS TO VALIDITY

The major threats to this empirical study are the quantity and quality of the relevant data, survey, and interview. Specifically, the dataset spans one year of historical data, which may not fully capture users' intentions due to limited recorded activities. Furthermore, the survey's scope is restricted to the code search users at Alibaba instead of all users as their contact information is not recorded by the tool. The clarity of the survey questions could impact its quality. To mitigate this issue, we conducted a pilot study with two experienced developers at Alibaba as described in Section 2.4. While many developers at Alibaba participated in the survey, the limited number of users may not fully represent the entire population. Additionally, as Alibaba Cloud continues to refine the code search feature in Lingma based on user feedback and this study, the findings from quantitative analysis in Section 3 may not apply to future versions. Consequently, the statistical results may not be generalizable to users of different code search tools or at different companies. However, despite these limitations, the insights and implications generated from this study are likely applicable to other similar tools. Additionally, due to the enterprise regulations, other data cannot be shared outside the enterprise. Hence, only final scrutinized statistics and results are presented, as described in Section 2. However, the Lingma development team has acknowledged the usefulness of this empirical study.

6 CONCLUSION AND FUTURE WORK

Code search serves as an important task for boosting developers' productivity. This paper reports the findings of our empirical study on the code search tool Lingma developed by Alibaba Group. We first performed quantitative analysis on the user-permitted recordings of code search activities to analyze users' preferences. We then designed questionnaires for the tool users at Alibaba, aiming to understand their perceptions and expectations. Afterward, we interviewed seven users to clarify their answers to the survey. Based on the results, we presented future research directions for researchers and practitioners. In our future work, we plan to include more companies of different sizes to investigate the generalizability of our findings. We also plan to improve the existing code search tool following the suggested future directions.

7 ACKNOWLEDGEMENTS

This research is supported by the National Nature Science Foundation of China (62202074 and 62372071), China Postdoctoral Science Foundation (2022M710519), the Postdoc Foundation of Chongqing (2021LY23), and Chongqing Technology Innovation and Application Development Project (CSTB2023TIAD-STX0015 and CSTB2022TIAD -KPX0068).

REFERENCES

[1] David F Bacon, Yiling Chen, David Parkes, and Malvika Rao. 2009. A market-based approach to software evolution. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*. 973–980.

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

[3] Celeste Barnaby, Koushik Sen, Tianyi Zhang, Elena Glassman, and Satish Chandra. 2020. Exempla Gratis (EG): Code examples for free. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1353–1364.

[4] Raymond PL Buse and Westley Weimer. 2012. Synthesizing API usage examples. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 782–792.

[5] Yitian Chai, Hongyu Zhang, Beijun Shen, and Xiaodong Gu. 2022. Cross-Domain Deep Code Search with Meta Learning. (2022).

[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[7] Zhongyang Deng, Ling Xu, Chao Liu, Luwen Huangfu, and Meng Yan. 2024. Code semantic enrichment for deep code search. *Journal of Systems and Software* 207 (2024), 111856.

[8] Zhongyang Deng, Ling Xu, Chao Liu, Meng Yan, Zhou Xu, and Yan Lei. 2022. Fine-grained Co-Attentive Representation Learning for Semantic Code Search. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 396–407.

[9] Luca Di Grazia and Michael Pradel. 2023. Code search: A survey of techniques for finding code. *Comput. Surveys* 55, 11 (2023), 1–31.

[10] BV Elasticsearch. 2018. Elasticsearch. *software], version* 6, 1 (2018).

[11] Guodong Fan, Shizhan Chen, Cuiyun Gao, Jianmao Xiao, Tao Zhang, and Zhiyong Feng. 2024. Rapid: Zero-shot Domain Adaptation for Code Search with Pre-trained Models. *ACM Transactions on Software Engineering and Methodology* (2024).

[12] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[13] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2018. Deep code search. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 933–944.

[14] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. 2019. Codekernel: A graph kernel based approach to the selection of API usage examples. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 590–601.

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

[16] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2018. Deep code comment generation. In *Proceedings of the 26th conference on program comprehension*. 200–210.

[17] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. 2020. Deep code comment generation with hybrid lexical and syntactical information. *Empirical Software Engineering* 25 (2020), 2179–2217.

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

[19] 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 40th International Conference on Software Engineering*. 946–957.

[20] Caroline Lemieux, Jeevana Priya Inala, Shuvendu K Lahiri, and Siddhartha Sen. 2023. Codamosa: Escaping coverage plateaus in test generation with pre-trained large language models. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 919–931.

[21] Chao Liu, Xuanlin Bao, Xin Xia, Meng Yan, David Lo, and Ting Zhang. 2022. CodeMatcher: a tool for large-scale code search based on query semantics matching. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1642–1646.

[22] Chao Liu, Xuanlin Bao, Hongyu Zhang, Neng Zhang, Haibo Hu, Xiaohong Zhang, and Meng Yan. 2023. Improving chatgpt prompt for code generation. *arXiv preprint arXiv:2305.08360* (2023).

[23] Chao Liu, Runfeng Cai, Yiqun Zhou, Xin Chen, Haibo Hu, and Meng Yan. 2024. Understanding the implementation issues when using deep learning frameworks. *Information and Software Technology* 166 (2024), 107367.

[24] Chao Liu, Xin Xia, David Lo, Cuiyun Gao, Xiaohu Yang, and John Grundy. 2021. Opportunities and challenges in code search tools. *ACM Computing Surveys (CSUR)* 54, 9 (2021), 1–40.

[25] Chao Liu, Xin Xia, David Lo, Zhiwe Liu, Ahmed E Hassan, and Shanping Li. 2021. CodeMatcher: Searching Code Based on Sequential Semantics of Important Query Words. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 1 (2021), 1–37.

[26] Yin Liu, Shuangyi Li, and Eli Tilevich. 2022. Toward a Better Alignment Between the Research and Practice of Code Search Engines. In *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 219–228.

[27] 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 (e). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 260–270.

[28] Frank J Massey Jr. 1951. The Kolmogorov-Smirnov test for goodness of fit. *J. Amer. Statist. Assoc.* 46, 253 (1951), 68–78.

[29] Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Andrian Marcus. 2015. How can I use this method?. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 880–890.

[30] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[31] 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*. 191–201.

[32] Ensheng Shi, Yanlin Wang, Wenchao Gu, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2023. Cocosoda: Effective contrastive learning for code search. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2198–2210.

[33] Zejian Shi, Yun Xiong, Yao Zhang, Zhijie Jiang, Jinjing Zhao, Lei Wang, and Shanshan Li. 2023. Improving Code Search with Multi-Modal Momentum Contrastive Learning. In *2023 IEEE/ACM 31st International Conference on Program Comprehension (ICPC)*. IEEE, 280–291.

[34] 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*. 196–207.

[35] 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 (TOSEM)* 21, 1 (2011), 1–25.

[36] Dominik Sobania, Martin Briesch, Carol Hanna, and Justyna Petke. 2023. An analysis of the automatic bug fixing performance of chatgpt. In *2023 IEEE/ACM International Workshop on Automated Program Repair (APR)*. IEEE, 23–30.

[37] Weisong Sun, Chunrong Fang, Yuchen Chen, Guanhong Tao, Tingxu Han, and Quanjun Zhang. 2022. Code Search based on Context-aware Code Translation. *arXiv preprint arXiv:2202.08029* (2022).

[38] Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. 2020. Intellicode compose: Code generation using transformer. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1433–1443.

[39] Yutian Tang, Zhijie Liu, Zhichao Zhou, and Xiapu Luo. 2024. Chatgpt vs sbst: A comparative assessment of unit test suite generation. *IEEE Transactions on Software Engineering* (2024).

[40] Jue Wang, Yingnong Dang, Hongyu Zhang, Kai Chen, Tao Xie, and Dongmei Zhang. 2013. Mining succinct and high-coverage API usage patterns from source code. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 319–328.

[41] 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 (2017), 3149–3185.

[42] Ling Xu, Huanhuan Yang, Chao Liu, Jianhang Shuai, Meng Yan, Yan Lei, and Zhou Xu. 2021. Two-stage attention-based model for code search with textual and structural features. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 342–353.

[43] 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 *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 344–354.

[44] Feng Zhang, Foutse Khomh, Ying Zou, and Ahmed E Hassan. 2012. An empirical study on factors impacting bug fixing time. In *2012 19th Working conference on reverse engineering*. IEEE, 225–234.

[45] Hongyu Zhang, Anuj Jain, Gaurav Khandelwal, Chandrashekhar Kaushik, Scott Ge, and Wenxiang Hu. 2016. Bing developer assistant: improving developer productivity by recommending sample code. In *Proceedings of the 2016 24th acm sigsoft international symposium on foundations of software engineering*. 956–961.

[46] Xinyu Zhang, Ji Xin, Andrew Yates, and Jimmy Lin. 2021. Bag-of-Words Baselines for Semantic Code Search. In *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021)*. 88–94.