Report from Dagstuhl Seminar 24172

Code Search

Satish Chandra*[1], Michael Pradel*[2], and Kathryn T. Stolee*[3]

1 **Google – Mountain View, US.** `schandra@acm.org`
2 **Universität Stuttgart, DE.** `michael@binaervarianz.de`
3 **North Carolina State University – Raleigh, US.** `ktstolee@ncsu.edu`

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar "Code Search" (24172). The seminar brought together researchers and practitioners working on techniques that enable software developers to find code and artifacts related to code. The participants discussed the state of the art in code search, identified open problems, and discussed future directions for research and practice. The seminar was structured with keynote talks, short talks, and breakout groups. Breakout groups identified how researchers can situate their code search research in terms of the targeted user groups, the access point for the developer, and the stage of software development that is most relevant to the code search tasks. Synergies between generative AI and Code Search were discussed, concluding that for some users and some tasks, generative AI can work with Code Search to enhance the developer experience and effectiveness. For other tasks, code search without generative AI would be more effective because of concerns regarding data provenance, update frequency, privacy, and the need for correctness.

Seminar April 21–24, 2024 – https://www.dagstuhl.de/24172
2012 ACM Subject Classification Software and its engineering → Integrated and visual development environments; Software and its engineering → Software development process management; Software and its engineering → Software evolution; Software and its engineering → Software libraries and repositories
Keywords and phrases code reuse, code search
Digital Object Identifier 10.4230/DagRep.14.4.108

1 Executive Summary

Kathryn T. Stolee (North Carolina State University – Raleigh, US)
Satish Chandra (Google – Mountain View, US)
Michael Pradel (Universität Stuttgart, DE)

The 3-day Dagstuhl Seminar on "Code Search" brought together leading experts from academia and industry to discuss and advance the field of code search. This seminar highlighted the critical role of code search in various software engineering activities, from locating where an error was thrown to learning new APIs or programming languages. It also emphasized the importance of search in automated software engineering tasks like automated program repair, code recommendation, and clone detection. The emergence of generative AI tools, which offer alternative methods for finding and reusing code, was also a significant topic of discussion.

* Editor / Organizer

Participants explored the implications of code search research on developer productivity, code quality, and software engineering ethics. They examined the diverse tools available for code search, ranging from internal company tools to open-source platforms like GitHub, and generative AI tools like ChatGPT. The seminar addressed various dimensions of code search, such as appropriate scope for search results, indexing methodologies, and combinations of code search and LLMs, e.g., in the form of retrieval-augmented generation.

In addition to talks and informal discussions, there were several break-out sessions during which participants discussed specific topics in smaller groups and eventually reported back to the other participants. Sections 4.1 provides an overview of the breakout sessions.

As a result of the seminar, several participants plan to launch various follow-up activities, such as joint publications and transferring promising ideas from academia to industry.

2 Table of Contents

3 Overview of Talks

3.1 Trustworthy Code Search: A Data-Centric Perspective

Bowen Xu (North Carolina State University – Raleigh, US)

Data quality plays an important role in LLMs' performance. For code search, there also exist several data quality issues from different aspects that may affect LLMs. For example, security, consistency, label correctness, etc. Regarding this, I presented an open-source library named SEEDGuard (https://seedguard.ai) I am currently developing with my students. SEEDGuard aims to generate higher-quality data for building LLM4Code.

3.2 Representations for (searching) (for? in? with?) spreadsheets

José Cambronero (Microsoft – Redmond, US)

Spreadsheet environments remain one of the most popular platforms for end-users (and non-professional programmers) to carry out computational tasks. In contrast to traditional programming environments, spreadsheets are inherently multimodal: they contain tabular (and non-tabular) data; code in the form of sheet formulas, recorded macros, and small data analysis programs in popular languages like Python; artifacts of analyses such as plots and formatted tables; and natural language in the form of table headers, comments, and values. To expand the applicability of code search to such environments, we must inherently tackle retrieval (and similarity and so on) across these different types of data. In this talk, I argue one possible way to do so is to leverage learned representations. However, for these representations to be effective we must incorporate domain-specific insights into the learning process. To illustrate this, I present an approach to learning spreadsheet formula representations that incorporates data curation, spreadsheet-specific tokenization, and pretraining objectives. Next, I provide an overview of a table representation learning approach that incorporates hierarchical position information and sheet-oriented pre-training objectives that enable these representations to be effective for the heterogeneity of tables in spreadsheets. Finally, I present some results showing that the effectiveness of LLMs at solving basic table tasks (such as value lookups) when using prompting-based approaches are not robust to the table serialization.

3.3 DiffSearch: A Scalable and Precise Search Engine for Code Changes

Luca Di Grazia (Universität Stuttgart, DE), Michael Pradel (Universität Stuttgart, DE)

Joint work of Luca Di Grazia, Paul Bredl, Michael Pradel
Main reference Luca Di Grazia, Paul Bredl, Michael Pradel: "DiffSearch: A Scalable and Precise Search Engine for Code Changes", IEEE Trans. Software Eng., Vol. 49(4), pp. 2366–2380, 2023.
URL https://doi.org/10.1109/TSE.2022.3218859

The source code of successful projects is evolving all the time, resulting in hundreds of thousands of code changes stored in source code repositories. This wealth of data can be useful, e.g., to find changes similar to a planned code change or examples of recurring code improvements. This paper presents DiffSearch, a search engine that, given a query that describes a code change, returns a set of changes that match the query. The approach is enabled by three key contributions. First, we present a query language that extends the underlying programming language with wildcards and placeholders, providing an intuitive way of formulating queries that is easy to adapt to different programming languages. Second, to ensure scalability, the approach indexes code changes in a one-time preprocessing step, mapping them into a feature space, and then performs an efficient search in the feature space for each query. Third, to guarantee precision, i.e., that any returned code change indeed matches the given query, we present a tree-based matching algorithm that checks whether a query can be expanded to a concrete code change. We present implementations for Java, JavaScript, and Python, and show that the approach responds within seconds to queries across one million code changes, has a recall of 80.7% for Java, 89.6% for Python, and 90.4% for JavaScript, enables users to find relevant code changes more effectively than a regular expression-based search and GitHub's search feature, and is helpful for gathering a large-scale dataset of real-world bug fixes.

You can try our online instance here: `http://diffsearch.software-lab.org/diffsearch`.

3.4 AI-Resilient Interfaces and the Value of Variation

Elena Leah Glassman (Harvard University – Allston, US)

Joint work of Elena Leah Glassman, Ziwei Gu, Jonathan Kummerfeld
Main reference Elena L. Glassman, Ziwei Gu, Jonathan K. Kummerfeld: "AI-Resilient Interfaces", CoRR, Vol. abs/2405.08447, 2024.
URL https://doi.org/10.48550/ARXIV.2405.08447

AI is powerful, but it can make both objective errors and contextually inappropriate choices. We need AI-resilient interfaces that help people be resilient to the AI choices that are not right, or not right for them. Existing human-AI interaction guidelines recommend that interfaces include user-facing features for efficient dismissal, modification, or otherwise efficient recovery from AI choices that the user does not like. However, users cannot decide to dismiss or modify AI choices that they have not noticed, and, without sufficient context, users may not realize that some of the noticed AI choices are wrong or inappropriate. In this talk, I discuss the challenges and benefits of designing AI-resilient interfaces for code search, and how two complementary theories of human concept learning – Variation Theory and Analogical Learning Theory – can provide design guidance.

3.5 Coccinelle for Rust

Julia Lawall (INRIA – Paris, FR)

Joint work of Julia Lawall, Tathagata Roy

Coccinelle is a tool for code search and transformation that has been under development since the mid 2000s. The main novelty of Coccinelle was to design the transformation language around the notion of a patch, familiar to source-code developers, and to extend this with metavariables and information about types and control-flow. Coccinelle was originally designed to support large-scale transformation in the Linux kernel, and has been extensively used by Linux kernel developers. Today, we are investigating whether the same approach can be successful for Rust code. This talk reviews some of the main design decisions of Coccinelle for C, including writing the parser and pretty printer from scratch and the design of the control-flow graph. We then consider how those design decisions have been adapted to Rust, including more reuse of existing Rust tools, and the potential implications of those decisions.

3.6 An Academic Perspective on Code Search and AI

Tobias Kiecker (HU Berlin, DE)

Joint work of Tobias Kiecker, Lars Grunske

This talk examines the impact of artificial intelligence (AI), especially large language models (LLMs), on software engineering research in general and on code search in particular. It starts with a recap on how LLMs have advanced or replaced other code generation techniques in recent years. The talk then goes on to our previous research on code search, addressing how LLMs have influenced this area and might shape it further in the future. It concludes with an open challenge, namely the relatively low visibility of code search in academic research and teaching, and advocates for the integration of this topic into software engineering curricula.

3.7 My Code Search: Then, Now

Dongsun Kim (Kyungpook National University – Daegu, KR)

My “Code Search” journey has two phases. At the first phase, I have focused on traditional code search techniques, which take query strings and return locations of source code in local or global code repositories. I figured out that many users of code search tools experienced the vocabulary mismatch problem. To address this problem, I proposed the “two-step” query translation approach based on StackOverflow posts. I built two code search tools based on this idea: CoCaBu (for free-form queries) and FaCoY (for code-to-code search). These tools are effective in searching for code locations against a given (free-form and code) queries.

The second phase explores applications of code search. First, I proposed an approach to detecting and repairing wrong inconsistent method names. This approach searches for similar methods by method names and bodies after embedding them into vectors. Then, the approach compares neighboring sets of a method name and body to figure out the inconsistency between them. This approach successfully detects inconsistent names and suggests better names. My recent applications include improving LLMs with code search techniques: Memorization discovery and membership inference attack.

References

1 Zhou Yang, Zhipeng Zhao, Chenyu Wang, Jieke SHI, Dongsun Kim, DongGyun Han, David Lo, "Unveiling Memorization in Code Models", in the Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE 2024), Lisbon, Portugal, April 14-20, 2024.

2 Zhou Yang, Zhipeng Zhao, Chenyu Wang, Jieke Shi, Dongsun Kim, DongGyun Han, David Lo: Gotcha! This Model Uses My Code! Evaluating Membership Leakage Risks in Code Models. CoRR abs/2310.01166 (2023)

3 Kui Liu, Dongsun Kim, Tegawendé F. Bissyandé, Taeyoung Kim, Kisub Kim, Anil Koyuncu, Suntae Kim and Yves Le Traon, "Learning to Spot and Refactor Inconsistent Method Names", in the Proceedings of the 41st International Conference on Software Engineering (ICSE 2019), Montréal, QC, Canada, May 25–31, 2019. Acceptance rate: 20.6% (109/529).

4 Kisub Kim, Dongsun Kim, Tegawendé F. Bissyandé, Eunjong Choi, Li Li, Jacques Klein, Yves Le Traon: FaCoY: a code-to-code search engine. ICSE 2018: 946-957

5 Raphael Sirres, Tegawendé F. Bissyandé, Dongsun Kim, David Lo, Jacques Klein, Kisub Kim, Yves Le Traon: Augmenting and structuring user queries to support efficient free-form code search. Empir. Softw. Eng. 23(5): 2622-2654 (2018)

3.8 A Journey through Searching Similar Code

Miryung Kim (University of California at Los Angeles, USA & Amazon Web Services – Palo Alto, USA)

This talk reflects on my group's research on searching similar code for the past 20 years, answering the following six questions: (1) What motivated us to research code search? (2) What were early attempts? (3) How serious is this problem? (4) How can we automate? (5) How can we examine variations at scale? (6) How to search similar code with a human in the loop?

We discuss that similar recurring updates motivated this line of work on searching similar code. As an early attempt, we created rule-based change abstractions and automatically inferred rules from diff-patches. We then quantified the effort needed to make similar changes in multiple contexts: co-evolution of forked projects, similar updates to clones, API evolution and ripple effects on client applications, and refactoring. We discussed our work on generalized patch synthesis to automate similar updates by learning from example patches. We realized the importance of examining search results at scale and designed a new visualization method of leveraging simultaneous overlay of similar code snippets. Then to enable construction of a search pattern with a human in the loop, we designed an active learning method that provides global distribution and what-if speculative analysis.

3.9 Code Search – Clone Search – Code Similarity

Jens Krinke (University College London, GB)

Joint work of Jens Krinke, Chaiyong Ragkhitwetsagul
Main reference Jens Krinke, Chaiyong Ragkhitwetsagul: "BigCloneBench Considered Harmful for Machine Learning", in Proc. of the 16th IEEE International Workshop on Software Clones, IWSC 2022, Limassol, Cyprus, October 2, 2022, pp. 1–7, IEEE, 2022.
URL https://doi.org/10.1109/IWSC55060.2022.00008

This talk presents the connection of code similarity to clone and code search. The application of clone search to investigate the provenance and quality of code on StackOverflow led to the development of a clone search engine evaluated with the often-used BigCloneBench dataset. However, this dataset is flawed due to how it has been constructed and the evaluation results are unreliable, particularly if the dataset is used to learn code similarity. Current work is on using LLMs to detect code similarity but another benchmark for functional similarity, CodeNet, is shown to be potentially illicit due to scraping copyrighted code. The talk concludes with preliminary results on using 36 LLMs to detect code similarity, which show that the LLMs are not yet ready for use as only six perform better than a random classifier.

3.10 Syntactic Code Search with Sequence-to-Tree Matching

Gabriel Matute (University of California – Berkeley, US)

Joint work of Gabriel Matute, Wode Ni, Titus Barik, Alvin Cheung, S. E. Chasins
Main reference Gabriel Matute, Wode Ni, Titus Barik, Alvin Cheung, Sarah E. Chasins: "Syntactic Code Search with Sequence-to-Tree Matching: Supporting Syntactic Search with Incomplete Code Fragments", Proc. ACM Program. Lang., Vol. 8(PLDI), pp. 2051–2072, 2024.
URL https://doi.org/10.1145/3656460

Lightweight syntactic analysis tools like Semgrep and Comby leverage the tree structure of code, making them more expressive than string and regex search. Unlike traditional language frameworks (e.g., ESLint) that analyze codebases via explicit syntax tree manipulations, these tools use query languages that closely resemble the source language. However, state-of-the-art matching techniques for these tools require queries to be complete and parsable snippets, which makes in-progress query specifications useless.

We propose a new search architecture that relies only on tokenizing (not parsing) a query. We introduce a novel language and matching algorithm to support tree-aware wildcards on this architecture by building on tree automata. We also present `stsearch`, a syntactic search tool leveraging our approach. In contrast to past work, our approach supports syntactic search *even for previously unparsable queries.* Our work offers evidence that lightweight syntactic code search can accept in-progress specifications, potentially improving support for interactive settings.

3.11 Scaling Embeddings for Github

Alexander Neubeck (GitHub – San Francisco, US)

There are a lot of papers and publications around RAG based systems. But most of the benchmarks, algorithms, and implementations focus on relatively small datasets (1-100 million embeddings) whereas at Github the scale is 100-1000x larger. At this scale, every tiny aspect of the RAG system must be revisited. Quality is now just one parameter in the equation, but no longer the most important one. One central part in RAG systems is the chunking strategy with the main focus to increase retrieval quality. However, at scale, stability and redundancy aspects become just as important. Picking a different strategy can decrease the cost easily by 10x and more. The talk shows for the various aspects of a RAG system which problems arise at scale and which questions need to be answered.

3.12 User Intent and Needs for Code Search

Nikitha Rao (Carnegie Mellon University – Pittsburgh, US)

Joint work of Vincent J. Hellendoorn, Martin Hirzel, Jason Tsay, Kiran Kate, Chetan Bansal, Thomas Zimmermann, Ahmed Hassan Awadallah, Nachiappan Nagappan, Joe Guan
Main reference Nikitha Rao, Jason Tsay, Kiran Kate, Vincent J. Hellendoorn, Martin Hirzel: "AI for Low-Code for AI", in Proc. of the 29th International Conference on Intelligent User Interfaces, IUI 2024, Greenville, SC, USA, March 18-21, 2024, pp. 837–852, ACM, 2024.
URL https://doi.org/10.1145/3640543.3645203
Main reference Nikitha Rao, Chetan Bansal, Thomas Zimmermann, Ahmed Hassan Awadallah, Nachiappan Nagappan: "Analyzing Web Search Behavior for Software Engineering Tasks", in Proc. of the 2020 IEEE International Conference on Big Data (IEEE BigData 2020), Atlanta, GA, USA, December 10-13, 2020, pp. 768–777, IEEE, 2020.
URL https://doi.org/10.1109/BIGDATA50022.2020.9378083
Main reference Nikitha Rao, Jason Tsay, Kiran Kate, Vincent J. Hellendoorn, Martin Hirzel: "AI for Low-Code for AI", in Proc. of the 29th International Conference on Intelligent User Interfaces, IUI 2024, Greenville, SC, USA, March 18-21, 2024, pp. 837–852, ACM, 2024.
URL https://doi.org/10.1145/3640543.3645203

Developers use search for various tasks such as finding code, documentation, debugging information, etc. First, we study user intents by conducting a large-scale analysis of web search behavior for software engineering tasks and propose a taxonomy of user intents. We then introduce a weak supervision based approach for detecting code search intent in search queries for C# and Java. Additionally, we present Search4Code, the first large-scale real-world dataset of code search queries mined from the Bing web search engine. Next, we extend our analysis beyond textual code and explore other forms of code representations such as low-code. We observe that different types of users (novices vs experts) may have different search needs, and demonstrate how LLMs can be useful in a visual (low-code) space, despite being trained on textual code, using LowCoder.

3.13 Code and Library Search

Christoph Treude (The University of Melbourne, AU)

License Creative Commons BY 4.0 International license
© Christoph Treude
Joint work of Brittany Reid, Marcelo d'Amorim, Markus Wagner, Christoph Treude
Main reference Brittany Reid, Marcelo d'Amorim, Markus Wagner, Christoph Treude: "NCQ: Code Reuse Support for Node.js Developers", IEEE Trans. Software Eng., Vol. 49(5), pp. 3205–3225, 2023.
URL https://doi.org/10.1109/TSE.2023.3248113

When developing software, finding the right pieces of code and the best libraries are important but challenging tasks. Code search lets developers find specific code snippets quickly, while library search helps them pick libraries that add more capabilities to their projects. To help, we've developed Node Code Query (NCQ), a tool that simplifies both tasks for Node.js developers. NCQ allows developers to search for NPM packages and code snippets, and it helps fix errors, set up testing environments quickly, and switch easily between searching and editing. Feedback from users shows that NCQ makes starting and finishing tasks faster, making it a valuable tool for Node.js developers. We've also started exploring methods to prioritize search results for diversity, ensuring users receive varied and useful results.

3.14 Querying code in Meta-SQL

Jan Van den Bussche (Hasselt University, BE)

License Creative Commons BY 4.0 International license
© Jan Van den Bussche
Joint work of Jan Van den Bussche, Stijn Vansummeren, Gottfried Vossen
Main reference Jan Van den Bussche, Stijn Vansummeren, Gottfried Vossen: "Towards practical meta-querying", Inf. Syst., Vol. 30(4), pp. 317–332, 2005.
URL https://doi.org/10.1016/J.IS.2004.04.001

We recall some ideas presented more than 20 years ago on meta-querying. Collections of database queries, e.g., SQL statements from database catalogs, or query logs from SPARQL endpoints on the semantic Web, are also datasets of code that we may want to query. We are interested in expressive querying, so we represent queries as trees. We go one step further and also want to be able to query the behavior of queries. We describe Meta-SQL, a prototype language that uses SQL/XML for querying and transforming the tree structures of code, and which includes an added EVAL function to dynamically execute queries.

3.15 Code Search Perspectives from (Startup) Industry

Rijnard van Tonder (Mysten Labs – Palo Alto, US)

License Creative Commons BY 4.0 International license
© Rijnard van Tonder
Main reference Rijnard van Tonder: "You Don't Know Search: Helping Users Find Code by Automatically Evaluating Alternative Queries", in Proc. of the 45th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, SEIP@ICSE 2023, Melbourne, Australia, May 14-20, 2023, pp. 270–280, IEEE, 2023.
URL https://doi.org/10.1109/ICSE-SEIP58684.2023.00030

The value and future of Code Search lies in the concrete benefits provided to ordinary developers. Developers use code search for simple tasks (finding a function) and complex ones (regular expressions to refactor parts of code). The wide spectrum of use cases imply the need for levels of code search expressivity that cater to both novice and advanced users. We pose the question of how to provide greater value to developers (e.g., efficiency and time-savings for completing software tasks) in terms of search query expressivity. For example, we find that in practice, the majority of users do not regularly use regular expressions in search queries. Providing value (i.e., greater efficiency) to developers through code search implies discovering methods and evaluating usage (e.g., click behavior on results sets) to discover a balance of expressivity in code search. We share our outlook on what continues to work well in practice (fast literal code search via indexing), what's changing (LLMs and prompt queries), and challenges that remain difficult (e.g., discovering user intent, especially across heterogeneous users and organizations who use code search).

3.16 Searching for code that doesn't exist

Cristina Videira Lopes (University of California – Irvine, US)

Joint work of Cristina Videira Lopes, Rakib Misu, Iris Ma, James Noble
Main reference Md Rakib Hossain Misu, Cristina V. Lopes, Iris Ma, James Noble: "Towards AI-Assisted Synthesis of Verified Dafny Methods", Proc. ACM Softw. Eng., Vol. 1(FSE), pp. 812–835, 2024.
URL https://doi.org/10.1145/3643763

Large Language Models don't know much about Dafny, because not much code exists written in Dafny. I explain some experiments we did that drastically improve the LLMs' ability to generate formally verified algorithms written in Dafny.

3.17 Code Search at Google

Tobias Welp (Google – München, DE)

Code Search enables fast search for tokens, files, filtering for languages, etc. across large code bases and browsing through code with semantic information and cross references. It requires continuous investment in advancing the technology to keep up with code base growth. In comparison to Code Search, LLMs provide the opportunity to cover for wider knowledge gaps of the user, potentially addressing some of their Code Search needs better, but provide less precision with higher latency.

3.18 Codesearch in developer journeys

Ciera Jaspan (Google – Mountain View, US)

When we extract logs from developer tools, we can usee that code search is a common task across nearly every common developer journey. It's used when trying to answer questions while coding, to share information with others, to review code, to debug production issues, and to identify security problems, among many many others. Codesearch is used by developers mfultiple times a day for all of these tasks. However, we are frequently missing two pieces of information when understanding these developer journeys. First, while we can see that engineers are doing these tasks, we can't determine their intent. We can't tell what question they are trying to answer or what the context is that they want an answer for. Second, we can't tell when a task is "successful"; there is no way to distinguish between "I found my answer" and "I gave up". Until we can see these differences, it's very hard to determine whether new features of codesearch, especially ones powered by LLMs, are actively helping developers achieve their goals or whether they are getting in the way.

3.19 Code Search + Code Review = ♡

Bogdan Vasilescu (Carnegie Mellon University – Pittsburgh, US) and Reid Holmes (University of British Columbia – Vancouver, CA)

Tools that search through code, the history of software repositories, and other software artifacts (hereafter just "code search tools") have a long history of development and deployment in industrial practice. The data generated by code search tools is especially relevant given the large-scale, quickly-evolving nature of modern systems. However, one common design challenge facing most code search implementations is that it is easy to overwhelm users with too much information. But there is hope! Large language models and the conversational agents that usually go with them tend to be particularly useful at summarizing large volumes of information. Could they also help with amplifying code review with search? In this work we outline a vision for augmenting code review with extra information from code search tools coupled with advanced LLM-based techniques for analyzing and summarizing these data into information a patch-writer or reviewer could use to improve a proposed patch.

4 Working groups

4.1 Overview of Breakout Sessions

Kathryn T. Stolee (North Carolina State University – Raleigh, US), Boris Bokowski (Google – München, DE), José Cambronero (Microsoft – Redmond, US), Satish Chandra (Google – Mountain View, US), Jürgen Cito (TU Wien, AT), Luca Di Grazia (Universität Stuttgart, DE), Elena Leah Glassman (Harvard University – Allston, US), Georgios Gousios (TU Delft, NL), Reid Holmes (University of British Columbia – Vancouver, CA), Ciera Jaspan (Google – Mountain View, US), Tobias Kiecker (HU Berlin, DE), Dongsun Kim (Kyungpook National University – Daegu, KR), Miryung Kim (University of California at Los Angeles, USA & Amazon Web Services – Palo Alto, USA), Jens Krinke (University College London, GB), Julia Lawall (INRIA – Paris, FR), Gabriel Matute (University of California – Berkeley, US), Alexander Neubeck (GitHub – San Francisco, US), Michael Pradel (Universität Stuttgart, DE), Nikitha Rao (Carnegie Mellon University – Pittsburgh, US), Christoph Treude (The University of Melbourne, AU), Jan Van den Bussche (Hasselt University, BE), Rijnard van Tonder (Mysten Labs – Palo Alto, US), Bogdan Vasilescu (Carnegie Mellon University – Pittsburgh, US), Cristina Videira Lopes (University of California – Irvine, US), Tobias Welp (Google – München, DE), Bowen Xu (North Carolina State University – Raleigh, US), and Svetlana Zemlyanskaya (JetBrains GmbH – München, DE)

4.2 Code Search Needs of Different User Groups

This breakout focused on different groups of users of code search tools. This includes professional developers, students, legacy developers, and hobbyists. The tasks they are trying to accomplish include navigation, search, information acquisition, observability (e.g., Do I understand this problem to know how many people will be impacted? How much resources will solving this require? What are the impacts of this security vulnerability?), and debugging.

When addressing the needs of a particular user group, it is important to understand their entry point into code search. Given a specific micro-intent/goal, how will they access code search? Is it within a document as in Ctrl+F? Is it as a separate browser tab? From there, how can we help developers retain and regain their mental context? Last, how easy or hard is it to consume the search results?

4.3 Impact of Generative AI Tools on Code Search

We had three breakout groups with the following prompt for discussion: *"Come up with two concrete examples of how code search and LLMs are good and two where they are bad."* Each group reported out, and we summarize the main points.

4.3.1 The Good of LLMs + Code Search

The power of the LLMs can be used to assist users with understanding complex queries and patterns (e.g., regexes). Similarly, the LLM could be used to summarize or analyze the results from code search to aid with code comprehension. Another interesting use case could be for finding clones, which may be more likely to be generated by a LLM than by a real person. LLMs may uncover better search / query heuristics. Analyzing tradeoffs between different decisions (e.g., choosing packages). Good at presenting results in a personalized way (e.g, personalized summarization / aggregation etc). Help less expert users act more like power users.

The ability to support fuzzy searches with embeddings for re-ranking search results would be a useful combination of LLMs and Code Search. LLMs can also go to answering the actual question instead of just code retrieval. Test code generation was mentioned in particular. LLMs fall short when fact extraction / code navigation is necessary (or at least unnecessary).

4.3.2 The Bad of LLMs + Code Search

On the flip side, often, we need results of a query or prompt to be correct. Additionally, checking for the absence of something is hard (e.g., have all references to a depreciated API been updated?). Complete results are needed (audits, security, etc.)

There could be legal issues or privacy concerns that prevent sending data to LLM. Also, there were provenance concerns and concerns about update frequency.

There were also concerns for education with respect to whether students will learn how to read code without writing it.

4.4 Code Search at Different Stages of Software Development

This breakout group discussed the different stages of software development and how code search can be used at each stage. The group discussed different stages where code search is relevant, such as learning a new language, debugging, and code reuse. The group also discussed the different tools and techniques that can be used at each stage, such as code search engines, code search in IDEs, and code search in documentation.

Participants

- Boris Bokowski
Google – München, DE
- José Cambronero
Microsoft – Redmond, US
- Satish Chandra
Google – Mountain View, US
- Jürgen Cito
TU Wien, AT
- Luca Di Grazia
Universität Stuttgart, DE
- Elena Leah Glassman
Harvard University – Allston, US
- Georgios Gousios
TU Delft, NL
- Reid Holmes
University of British Columbia – Vancouver, CA
- Ciera Jaspan
Google – Mountain View, US
- Tobias Kiecker
HU Berlin, DE
- Dongsun Kim
Kyungpook National University – Daegu, KR
- Miryung Kim
University of California at Los Angeles, USA & Amazon Web Services – Palo Alto, US
- Jens Krinke
University College London, GB
- Julia Lawall
INRIA – Paris, FR
- Gabriel Matute
University of California – Berkeley, US
- Alexander Neubeck
GitHub – San Francisco, US
- Michael Pradel
Universität Stuttgart, DE
- Nikitha Rao
Carnegie Mellon University – Pittsburgh, US
- Kathryn T. Stolee
North Carolina State University – Raleigh, US
- Christoph Treude
The University of Melbourne, AU
- Jan Van den Bussche
Hasselt University, BE
- Rijnard van Tonder
Mysten Labs – Palo Alto, US
- Bogdan Vasilescu
Carnegie Mellon University – Pittsburgh, US
- Cristina Videira Lopes
University of California – Irvine, US
- Tobias Welp
Google – München, DE
- Bowen Xu
North Carolina State University – Raleigh, US
- Svetlana Zemlyanskaya
JetBrains GmbH – München, DE