RFC 699 REVIEW: Building Google for code

Author: Rok Novosel

Date: May 23, 2022

Approvers: Loïc Guychard Ben Venker Beyang Liu Petri Last

Status: Review

Motivation

Sourcegraph's ultimate goal is to make it so everyone can code, and there are multiple ways we can accomplish that. We have a good set of medium-term goals that center around use cases. If we execute them correctly, that gets Sourcegraph one incremental step closer to achieving our ultimate goal. On the other hand, there are larger steps we could take in that direction. But there is an inherent risk with large steps, as we could miss our goal and spend a lot of resources on a useless exploration or get stuck in a local maximum, not really knowing where to go next (if you think of it as an optimization problem).

Concretely, a large step I would like to take is building a Google-for-code. We could argue that Sourcegraph already accomplishes that to a certain extent. In my mind, two key things are missing: natural language queries and multiple sources of results. Sourcegraph has an excellent query language that operates on code and artifacts derived from code (symbols, code intel). That in itself makes it a powerful tool in many scenarios. Still, to get the most out of it, you already have to be comfortable with programming and familiar with the subject you are researching (syntax-wise). Many programmers still tend to turn to Google when they are stuck learning a new language, learning a new tech stack, or exploring a new topic.

A piece of anecdotal evidence that resonated with me is a Reddit thread about a junior developer having a hard time in his first programming job out of college. The relevant quote from the thread is:

“I find even asking coworkers questions hard because it takes me literally an hour or two sometimes to formulate my question in a way that makes sense.”

What if it didn’t have to be that way? What if we could provide tools to the junior engineers of the world where they can help themselves? They would feel empowered by their ability to find answers and, in turn, reduce the burden on their senior colleagues. Again, Sourcegraph could already alleviate some of the pain through the developer onboarding use case and being an excellent code search engine. But I want to take Sourcegraph a step further. I want to give them a knowledge engine capable of finding the most relevant StackOverflow posts and GitHub issues alongside corresponding code snippets from real-world projects. Code doesn’t exist in a vacuum, and one usage example can be worth 1000 pages of docs.

Junior engineers are not the only ones getting stuck. Everyone gets stuck eventually, no matter their skill level. Maybe you’re exploring a new topic or forgot how to throw an exception in Python after programming in Go for so long. And at that point, you don’t want to read a whole Python tutorial just to learn how to throw an exception or read through an entire blog post comparing exceptions in different languages. You already know what an exception is. You only need one line to keep you moving.

Google is still by far the best place to get these kinds of answers. But depending on the query, the results can be a mixed bag. If we set aside the ad issues, we are still left with aggressive SEO manipulation and coding clone websites that slowly erode users' trust. The amount of information presented can sometimes be overwhelming, especially to beginners who have not yet learned what is safe to ignore and what will most likely answer their questions. More often than not, programmers end up on StackOverflow or wading through a long-winded blog post (again due to SEO) trying to extract the relevant piece of information. I found a blog post from a programmer in which he mostly corroborates my findings. He argues that Google is making him a better programmer, although not because of the quality of the search results. In fact, it’s the opposite. Low-quality results forced him to branch out and invest in better tooling, a personal knowledge base, and self-documentation.

A few alternative search engines started popping up in recent years that address some of Google's shortcomings. And they started explicitly catering to the programmer crowd, primarily by extracting relevant information from popular sites like StackOverflow and presenting them upfront on the results page. Here is a comparison between Google and newer competitors (you.com, Neeva, and Kagi). You.com even showcased YouCode, a search engine for coding.

But none of the mentioned search engines can deal with actual code. It's mainly because it's hard to bridge the gap between natural language and code. With recent advancements in machine learning techniques, especially transformers, it has become easier to train a model capable of semantic code search. It is not perfect, but it gives us a way to connect natural language meaning with code.

Proposal

To address the issues I outlined above, I propose that we build a Google-for-code-style web application. It could also be framed as a vertical search engine for code. It should accept natural language queries, return related code snippets from multiple sources, and present them side-by-side in an easily digestible manner. For the initial prototype, I'm targeting to include code functions gathered from GitHub and code answers from StackOverflow. Future sources could consist of doc sites, blog posts, and anything code-related.

There are several important considerations to take into account. The web application would not be integrated into the main Sourcegraph application because that allows us faster iteration and validation. The application should still be affiliated with Sourcegraph and use the appropriate branding, but, in essence, it would act as a separate product. Additionally, we would only consider publicly available code and data. This does not mean it could not be adapted as an on-premise solution. We would have to get strong interest from potential customers since deployment and indexing are far from trivial. Updating the index regularly is also not a primary concern for now.

As I alluded to earlier, the solution would be based on the transformers model. Transformers are a state-of-the-art natural language processing model used in many

applications (question-and-answering, bots, summarization). The onset of transformers is also one of the reasons why natural language code search has become a more exciting topic. Transformers are significantly easier to train, and they come with a considerable performance boost. Previously, you had to engineer massive neural network architectures that were hard to train and understand. Spotify, for example, has already put transformers to use for their natural language podcast search.

To demonstrate the feasibility of natural language code search, I built a proof of concept search application using the Huggingface transformers library. Here is a quick demo video of the application. It operates solely on code functions and doesn't have any additional sources. The prototype was also essential to evaluate the time, compute power, and monetary costs. Gathering the data, training the models, and building the indices took roughly 2 weeks. Subsequent updates are not as time-consuming because we can reuse the models trained in the first stage. The total price is dictated by the cost of GPUs on Google Cloud. I used 4 NVIDIA T4 GPUs, and the monthly price is $178 per GPU.

I omitted the detailed technical specs from this document since I want to focus on the motivation and the high-level proposal. I'll include the technical details in a separate document if the proposal is approved.

Timeline

- Gathering and cleaning the data (3-4 weeks)
    - Build parsers (using tree-sitter) to extract functions in multiple languages
    - Investigate the StackOverflow data dump and extract the code from the answers
    - Decide on the data model and how it will be stored
- Training the models (2-3 weeks)
    - This will mostly be an idle time while we wait for the models to train. We can start working on the web app in parallel.
- Building nearest neighbor search index and the corresponding database (1 week)
- Building the web application (2 weeks)

    - I like the Kagi Search model of a gradual rollout through private alpha and private beta stages with a landing page.
- Evaluation (2 weeks)

Definition of success

The future of the project will hinge on positive internal feedback. To get quantitative feedback, we can have a Google Form asking, “Would you use this tool day-to-day over Google?” and, if not, what improvements we would have to make. We should target overwhelmingly positive answers. Otherwise, it makes no sense to proceed with a public rollout. We should also perform user research sessions to evaluate the strong and weak points of the search engine. For example, we could have a set of tasks that the user must complete with Google and then compare them with our project. If we are happy with the internal feedback, we can move into a private beta phase and invite external users to evaluate the project. We can reach out to our existing customers if their developers would be interested in evaluating our project. Again, we can proceed with the same feedback structure as for internal users. Once we are confident that our project is on the right track, we can opt for public beta and share it through our social media channels. We should target an ambitious goal for our public beta link to be on the top half of the Hacker News front page.

Appendix A: Collection of research links

- Surge HQ: Google Search is falling behind
- Three areas where Google Search lags behind competitors: code, cooking, travel | Hacker News
- How Google search is making me a better programmer
- Google no longer producing high-quality search results in significant categories | Hacker News
- Google Search Is Dying | Hacker News
- Show HN: YouCode, a Search Engine for Coding | Hacker News

- Why Google is so unbearable (and how to fix it)
- We need more boutique search engines | Hacker News
- The future of search is boutique | Hacker News
- The Future of Search Is Boutique
- Introducing Natural Language Search for Podcast Episodes
- A rant about search engine results for large format laser printers [video] | Hacker News