**GitChat** (POC Page 3)

https://github.com/arunkv1/GitChat

Arun Krishna Vajjala, Ajay Krishna Vajjala, Deval Parikh

**Problem and Motivation:**

Many companies in the industry have large software codebases that can be challenging and time consuming to navigate for engineers, non-technical team members, and leadership. These current practices require developers to get training and familiarize themselves with the codebases to make meaningful contributions. This can consume a significant amount of the company’s time and resources.

To tackle these challenges, we introduce GitChat—an interactive AI Developer tool designed to comprehend codebases and enable developers to ask questions about it using natural language in plain English. This give developers a means to interact directly with large codebases without tediously navigating the documentation and technical components within it. This tool has the potential to significantly reduce the time it takes for developers to familiarize themselves with the source code. It can also enhance the productivity of developers at any level by enabling them to directly "chat" with the codebase. GitChat eliminates the need for laborious exploration to comprehend the complex logic and architectures within the code.

This increase in productivity can potentially result in developer output and performance since they no longer need to spend time manually searching the code. This allows developers to focus their efforts on more creative problem solving and innovation. Companies can benefit from this heightened level of productivity to achieve their goals.

**Use Case**

* Case 1: Team Members of Any Technical Background
  + In cases where there are multidisciplinary teams, there may be team members who may not understand the codebase but need to understand the overarching functionality of features within the product. GitChat will not only help those with an advanced technical background but provide a natural language explanation of features within the codebase for those with a less-technical expertise. This improves productivity within teams by providing a seamless knowledge transfer between team members of varied technical experience.
* Case 2: Onboarding Process
  + New team members within the company or team must familiarize themselves with the codebase in order to make meaningful contributions. This process can take weeks to months of the company’s time. By providing new hires with GitChat, they can simply “chat with the codebase” to understand it. These queries can be as simple as asking questions to understand functions or as complex as asking how certain large-level systems function.
* Case 3: Experienced Developers
  + Experienced developers are well versed in the code base and provide creative solutions to problems that may arise. These problems often require a search of the documentation to ensure that they can make the necessary changes. GitChat allows these high-skilled developers to simply query the codebase for design details and architecture information that they may need in order to implement the solution.

**Technical Overview**

Built with: Python, OpenAI GPT API, LangChain, GitHub Python Package

1. Index the Codebase: Duplicate the target repository, load all contained files, divide the files, and initiate the indexing procedure. Alternatively, you can bypass this step and use a pre-indexed dataset.

2. Store Embeddings and the Code: Code segments are embedded using a code-aware embedding model and saved in the Deep Lake VectorStore. This is done via LangChain. Code chunks are limited to 1000 characters and are encoded using OpenAi’s embeddings

3. Assemble the Retriever: Conversational Retriever Chain searches the VectorStore to find a specific query's most relevant code segments using cosine similarity. It uses context-aware filtering and ranking to figure out which code snippets and info are most relevant. Also done via LangChain

4. Build the Conversational Chain: Customize retriever settings and define any user-defined filters as necessary. We used a filter to filter for programming and documentation files, ignoring git/build files for maximum code context.

5. Pose Questions: Create a list of questions about the codebase, then use the Conversational Retrieval Chain to produce context-sensitive responses. The LLM (GPT-4, in this case) should now generate detailed, context-aware answers based on the retrieved code segments and conversation history.

**Proof of Concept**

GitChat asks if you want to use a pre-existing dataset. GitChat stores previously cloned GitHub Repos so that it does not have to clone them again

**A screenshot of a computer

Description automatically generated**

It lists the repos that have previously been cloned and allows you to select a repo to query

**A screenshot of a computer

Description automatically generated**

Once the repo is ready, the user can ask a question regarding the codebase

**A screenshot of a computer

Description automatically generated**

The answer is then presented to the user. In this case, it gives an explanation and function of the nav bar in the given application which was in the repo that we provided the link for.

**A screenshot of a computer

Description automatically generated**

We can pose more program related questions as well, such as changing colors of objects in the code. GitChat uses GPT to generate responses, so it is capable of generating code snippets.

**A screenshot of a computer

Description automatically generated**

Here we see that GitChat provides the substitute solution for the code in the repo

**A screenshot of a computer

Description automatically generated**