**IIIT Pune Computer Science**

**2023-2027**

**PROJECT REPORT for Internship**

**A Python-based tool that integrates ChatGPT (versions 3.5 or 4) and GitHub for the purpose of automated codebase analysis and improvement suggestions**

**Khushi Garg**

**IIIT Pune Comp Sc**

A screenshot of a computer

Description automatically generated

**Index**

|  |  |
| --- | --- |
| **Serial Number** | **Topic** |
| 1. | Overview of LangChain, OpenAI |
| 2. | Project Synopsis |
| 3. | System Requirements |
| 5. | Program Code |
| 6. | Program Output |

**Overview of OpenAI**

The OpenAI API can be applied to virtually any task. It offers a range of [models](https://platform.openai.com/docs/models) with different capabilities and price points, as well as the ability to [fine-tune](https://platform.openai.com/docs/guides/fine-tuning) custom models.

## [Key concepts](https://platform.openai.com/docs/introduction/key-concepts)

### [Text generation models](https://platform.openai.com/docs/introduction/text-generation-models)

OpenAI's text generation models (often referred to as generative pre-trained transformers or "GPT" models for short), like GPT-4 and GPT-3.5, have been trained to understand natural and formal language. Models like GPT-4 allows text outputs in response to their inputs. The inputs to these models are also referred to as "prompts". Designing a prompt is essentially how you "program" a model like GPT-4, usually by providing instructions or some examples of how to successfully complete a task. Models like GPT-4 can be used across a great variety of tasks including content or code generation, summarization, conversation, creative writing, and more. Read more in our introductory [text generation guide](https://platform.openai.com/docs/guides/text-generation) and in our [prompt engineering guide](https://platform.openai.com/docs/guides/prompt-engineering).

### [Assistants](https://platform.openai.com/docs/introduction/assistants)

Assistants refer to entities, which in the case of the OpenAI API are powered by large language models like GPT-4, that are capable of performing tasks for users. These assistants operate based on the instructions embedded within the context window of the model. They also usually have access to tools which allows the assistants to perform more complex tasks like running code or retrieving information from a file. Read more about assistants in our [Assistants API Overview](https://platform.openai.com/docs/assistants).

### [Embeddings](https://platform.openai.com/docs/introduction/embeddings)

An embedding is a vector representation of a piece of data (e.g. some text) that is meant to preserve aspects of its content and/or its meaning. Chunks of data that are similar in some way will tend to have embeddings that are closer together than unrelated data. OpenAI offers text embedding models that take as input a text string and produce as output an embedding vector. Embeddings are useful for search, clustering, recommendations, anomaly detection, classification, and more. Read more about embeddings in our [embeddings guide](https://platform.openai.com/docs/guides/embeddings).

### [Tokens](https://platform.openai.com/docs/introduction/tokens)

Text generation and embeddings models process text in chunks called tokens. Tokens represent commonly occurring sequences of characters. For example, the string " tokenization" is decomposed as " token" and "ization", while a short and common word like " the" is represented as a single token. Note that in a sentence, the first token of each word typically starts with a space character. Check out our [tokenizer tool](https://platform.openai.com/tokenizer) to test specific strings and see how they are translated into tokens. As a rough rule of thumb, 1 token is approximately 4 characters or 0.75 words for English text.

One limitation to keep in mind is that for a text generation model the prompt and the generated output combined must be no more than the model's maximum context length. For embeddings models (which do not output tokens), the input must be shorter than the model's maximum context length. The maximum context lengths for each text generation and embeddings model can be found in the [model index](https://platform.openai.com/docs/models/overview).

**LangChain** is a framework for developing applications powered by language models. It enables applications that:

* **Are context-aware**: connect a language model to sources of context (prompt instructions, few shot examples, content to ground its response in, etc.)
* **Reason**: rely on a language model to reason (about how to answer based on provided context, what actions to take, etc.)

This framework consists of several parts.

* **LangChain Libraries**: The Python and JavaScript libraries. Contains interfaces and integrations for a myriad of components, a basic run time for combining these components into chains and agents, and off-the-shelf implementations of chains and agents.
* [**LangChain Templates**](https://python.langchain.com/docs/templates): A collection of easily deployable reference architectures for a wide variety of tasks.
* [**LangServe**](https://python.langchain.com/docs/langserve): A library for deploying LangChain chains as a REST API.
* [**LangSmith**](https://python.langchain.com/docs/langsmith): A developer platform that lets you debug, test, evaluate, and monitor chains built on any LLM framework and seamlessly integrates with LangChain.

**Streamlit** is a free and open-source framework to rapidly build and share beautiful machine learning and data science web apps. It is a Python-based library specifically designed for machine learning engineers. Data scientists or machine learning engineers are not web developers and they're not interested in spending weeks learning to use these frameworks to build web apps. Instead, they want a tool that is easier to learn and to use, as long as it can display data and collect needed parameters for modeling. Streamlit allows you to create a stunning-looking application with only a few lines of code.

**Project Synopsis**

Develop a Python-based tool that integrates ChatGPT (versions 3.5 or 4) and GitHub for the purpose of automated codebase analysis and improvement suggestions.

Detailed Requirements

1. Python Script Development Task: Develop a robust Python script.

Purpose: To serve as a bridge between GitHub and ChatGPT for code analysis.

Skills Required: Proficiency in Python, understanding of API integration.

2. Token Utilization Task: Implement secure usage of ChatGPT and GitHub tokens.

Purpose: To authenticate and enable interactions between ChatGPT and GitHub.

Skills Required: Understanding of OAuth protocols, API authentication, and security best practices.

3. GitHub Repository Fetching Task: Enable the script to fetch and access code from a specified GitHub repository.

Purpose: To analyze the repository's codebase using ChatGPT or other LLMs.

Skills Required: Familiarity with GitHub APIs, Git commands, and repository management.

4. Codebase Analysis with LLMs Task: Integrate LLM (such as ChatGPT) for code analysis.

Purpose: To provide insights and suggestions on the codebase. ML Engineer Intern 2

Skills Required: Understanding of machine learning models, particularly large language models like ChatGPT.

5. Model Analysis and Suggestions Code Improvement: Objective: Enhance code quality and maintainability. Expectation: Suggestions for code refactoring and improvements.

Code Optimization:

Objective: Improve code performance.

Expectation: Identifying areas for efficiency increase, such as reducing time complexity.

Test Case Development:

Objective: Strengthen testing and validation.

Expectation: Suggest additional test cases for better coverage.

Bug Identification and Resolution:

Objective: Identify and fix potential bugs.

Expectation: Pinpointing bugs with possible solutions or preventive measures.

6. Documentation and Reporting Task: Ensure output documentation of findings and suggestions.

Purpose: To provide a clear, actionable report for developers.

Skills Required: Ability to present data and suggestions in a user-friendly format.

**System Requirement**

streamlit

langchain

langchain-community

dotenv

GitPython

**Program Code**

import os

import shutil

from git import Repo

import streamlit as st

from langchain.text\_splitter import RecursiveCharacterTextSplitter, Language

from langchain.document\_loaders.generic import GenericLoader

from langchain.document\_loaders.parsers import LanguageParser

from langchain\_openai import OpenAIEmbeddings, ChatOpenAI

from langchain\_community.vectorstores import Chroma

from langchain.memory import ConversationSummaryMemory

from langchain.chains import ConversationalRetrievalChain

from PIL import Image

from dotenv import load\_dotenv

# Load environment variables

load\_dotenv()

OPENAI\_API\_KEY = os.getenv("OPENAI\_API\_KEY")

# Repo path in local directory

REPO\_PATH = "./Cloned\_repos"

# Utility Functions

def delete\_local\_repo(path):

    """Delete a local repository."""

    if os.path.exists(path):

        shutil.rmtree(path)

def create\_local\_repo(path):

    """Create a local repository directory."""

    os.makedirs(path, exist\_ok=True)

def clone\_repository(repo\_url, to\_path):

    """Clone a repository from a given URL."""

    delete\_local\_repo(to\_path)

    create\_local\_repo(to\_path)

    Repo.clone\_from(repo\_url, to\_path=to\_path)

def process\_documents(path, sub\_directory=None):

    """Process documents in the given path."""

    # If sub\_directory is not provided, use the base path

    process\_path = path if not sub\_directory else os.path.join(path, sub\_directory)

    loader = GenericLoader.from\_filesystem(process\_path, glob="\*\*/\*",

                                        suffixes=[".py", ".java", ".cpp", ".hpp", ".c", ".h", ".js", ".md", ".sh",".bat"],

                                        parser=LanguageParser(language=Language.PYTHON, parser\_threshold=500))

    documents = loader.load()

    documents\_splitter = RecursiveCharacterTextSplitter.from\_language(language=Language.PYTHON,

                                                                    chunk\_size=2000, chunk\_overlap=200)

    texts = documents\_splitter.split\_documents(documents)

    return texts

def setup\_vector\_db(texts):

    """Setup vector database for document processing."""

    embeddings = OpenAIEmbeddings(disallowed\_special=())

    vectordb = Chroma.from\_documents(texts, embedding=embeddings, persist\_directory='./data')

    vectordb.persist()

    return vectordb

def setup\_conversational\_chain():

    """Setup the conversational retrieval chain."""

    llm = ChatOpenAI(model\_name='gpt-3.5-turbo')

    memory = ConversationSummaryMemory(llm=llm, memory\_key="chat\_history", return\_messages=True)

    qa\_chain = ConversationalRetrievalChain.from\_llm(llm, retriever=vectordb.as\_retriever(search\_type="mmr", search\_kwargs={"k":8}), memory=memory)

    return qa\_chain

# Streamlit Interface

st.title("GitHub Repository Code Analyzer")

# Sidebar

st.sidebar.title("About")

st.sidebar.markdown("## Made by: Khushi Garg 👋")

st.sidebar.text("Additional Info 💻 ")

st.sidebar.markdown("[Project code ](https://github.com/garg-khushi/openai)")

st.sidebar.text("Mail-id : khushi-garg123@gmail.com")

repo\_url = st.text\_input("Enter the Repository URL", "Github Repository url")

sub\_directory = st.text\_input("Enter the subdirectory (optional)")

question = st.text\_input("Enter your question")

if st.button("Analyze"):

    if repo\_url and question:

        try:

            clone\_repository(repo\_url, REPO\_PATH)

            texts = process\_documents(REPO\_PATH, sub\_directory)

            vectordb = setup\_vector\_db(texts)

            qa\_chain = setup\_conversational\_chain()

            result = qa\_chain.invoke(question)

            st.write(result['answer'])

        except ValueError as ve:

            st.error(str(ve))

        except Exception as e:

            st.error(f"An unexpected error occurred: {e}")

    else:

        st.error("Please enter the repository URL and your question.")

**Program Output**

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Text

Description automatically generated

**Graphical user interface, text

Description automatically generated**

Text

Description automatically generated