IE University

School of Sciences & Technology

Bachelor of Computer Science and Artificial Intelligence

AI: Chatbots and Recommendation Engines

CodeCompass

Final Report

Luca Cuneo

lcuneo.ieu2021@student.ie.edu

Gabriel de Olaguibel

gdeolaguibel.ieu2021@student.ie.edu

Keti Sulamanidze

ksulamanidze.ieu2021@student.ie.edu

Miranda Drummond

mdrummond.ieu2021@student.ie.edu

Felix Gomez-Guillamon

fgomezguilla.ieu2021@student.ie.edu

Maud Helen Hovland

mhovland.ieu2021@student.ie.edu

07.04.2024

This document provides a comprehensive overview of the completion of CodeCompass, a Github recommendation and chatbot project. The project team has worked diligently across various fronts, including data engineering, testing, EDA, feature engineering, and model development, culminating in the final front-end interface. Below is a detailed account of our completed work.

Product Vision

CodeCompass aims to enhance the GitHub experience for developers and learners alike by providing a personalized interface to discover and interact with repositories. Our goal is to make learning and discovery in GitHub intuitive, informative, and tailored to each user's interests and needs.

Solution

CodeCompass integrates two core features to realize our vision:

- **Personalized Recommendations**: Leveraging user-specific data, CodeCompass delivers a curated recommendation engine for GitHub projects. These suggestions are driven by an analysis of the user's past interactions, preferences, and potential areas of interest, ensuring that each recommendation is relevant and engaging.
- **Interactive Exploration and Learning**: Our chatbot is designed to enrich the user's exploration of GitHub repositories. It acts as an interactive guide, offering insights into repository contents, technologies used, and context information. This feature encourages users to engage deeply with recommended projects, fostering a learning environment.

Project Management and Development Model

Backlog Management

- Completion Status: We achieved 100% completion of all items in our GitHub project backlog, marking a significant achievement towards our project vision completion and Demo date of April 3rd, 2024

Development Practices

Branch Management

- Approach: We maintained high code quality and facilitated collaborative development by employing short-lived branches for all development activities. Direct pushes to the main branch were strictly prohibited to ensure thorough review processes, and branches were deleted following their merge, keeping our repository clean and manageable.

Pull Requests (PRs)

- Code Review Process: Our team implemented a rigorous review protocol, necessitating at least one member's review and commentary on each pull request. Additionally, PRs were required to pass automated workflow tests prior to merging, ensuring code integrity and functionality.

Issues

- Collaboration and Transparency: By seamlessly integrating issues with our GitHub project backlog, we promoted an environment of open discussion. This approach enabled efficient collaboration on task resolutions and enhancements, furthering our project's development.

Data Engineering

Recommender GitHub API Integration

The *codecompasslib/API* directory houses scripts that utilize the GitHub API to gather in-depth repository and user profile information.

Data Retrieval and Preparation

- *get_bulk_data.py:* targets the acquisition of comprehensive data features, subsequently saving this information in CSV format for analysis.
- *helper_functions.py:* Provides critical utility functions supporting authentication token management and efficient data storage practices.

Chatbot GitHub API Retrieval

Within *codecompasslib/chatbot*, scripts employ the AskTheCode API—a wrapper around the GitHub API—for precise repository data retrieval based on user queries.

- *api_utilities.py:* Manages post requests and filters the received data to include only necessary JSON keys, optimizing data handling
- *repo_info.py:* Empowers the chatbot with capabilities to fetch a diverse set of repository details, enhancing user interaction with targeted information.

External Data Storage

To facilitate accessible and centralized data management, extracted information is stored using the Google Drive API. This approach ensures that Data Scientists and team members have uninterrupted access to a well-structured dataset ready for modeling and analysis.

- *drive_operations.py:* Implements utility code that enables seamless data access and integration within the project's external data storage framework.

Recommendation Model

Exploratory Data Analysis and Feature Engineering

Data Cleaning

- Utility Functions: The *codecompasslib/models/utilities/clean_data.py* script houses functions for refining the dataset by removing unnecessary columns, cleaning text data (e.g., stopwords, special characters), handling missing values, and saving the processed data into a new CSV file.

Data Exploration (EDA)

- Visualization and Analysis: *codecompasslib/models/utilities/eda.ipynb* a Python notebook, dives into data exploration and visualization. It utilizes RandomForestRegressor for determining feature importance, aiding the subsequent model development phase.

Feature Engineering

Text Encoding

Located in codecompasslib/embeddings:

- Embedding Generation: *generate_embedded_dataset.py* leverages OpenAI text embeddings 3-large for creating text embeddings from repository descriptions, which are then stored externally.
- Utility Scripts: *embeddings_helper_functions.py* offers utility functions for experimenting with various embedding techniques (e.g., Word2Vec, CodeLlama, Roberta-base, OpenAI), enhancing the dataset with rich text features.

Non-Text Data Preparation

- As of now, models work mostly based on textual data such as name, description, language (which we label encode). However, in a case when it uses a numerical column, such as 'stars', any row with missing values is simply dropped. This does not affect the size of the dataset in any significant way.

Recommendation Models

LightGBM Model

- **Production Implemented:** *models/lightgbm_model.py* introduces a LightGBM model predicting the likelihood of a user 'starring' a repository. Chosen for its efficiency, speed, gradient-boosting capabilities, and adept handling of categorical features.

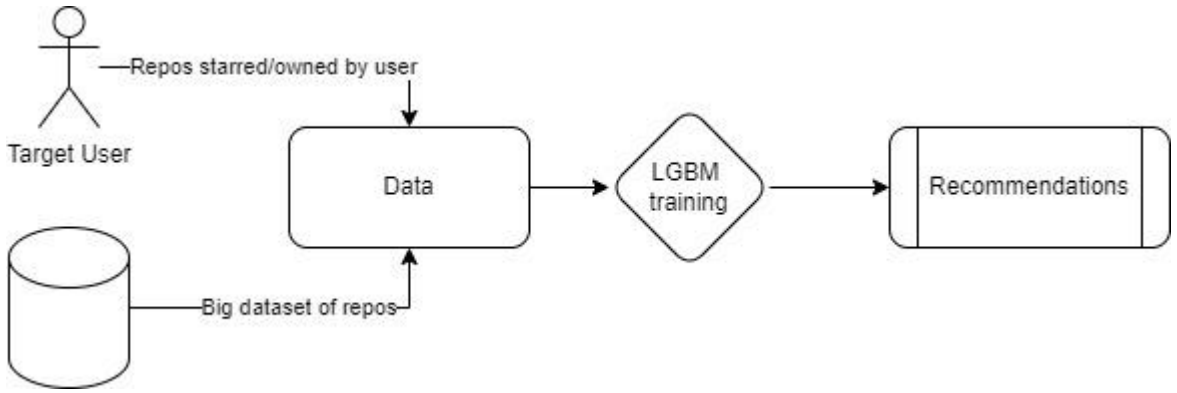
Cosine Similarity

- Trial Application: *models/cosine_similarity_model.py* employs cosine similarity to align repository recommendations with user language preferences and textual data insights.

Differentiating Repositories Model

- Trial Application: *models/model_diff_repos.py*: identifies similar users and distinguishes unique projects among them, enhancing personalized recommendations.

Recommendation System Design (For LightGBM Model)



Repository Retrieval Chatbot

Management and Initialization

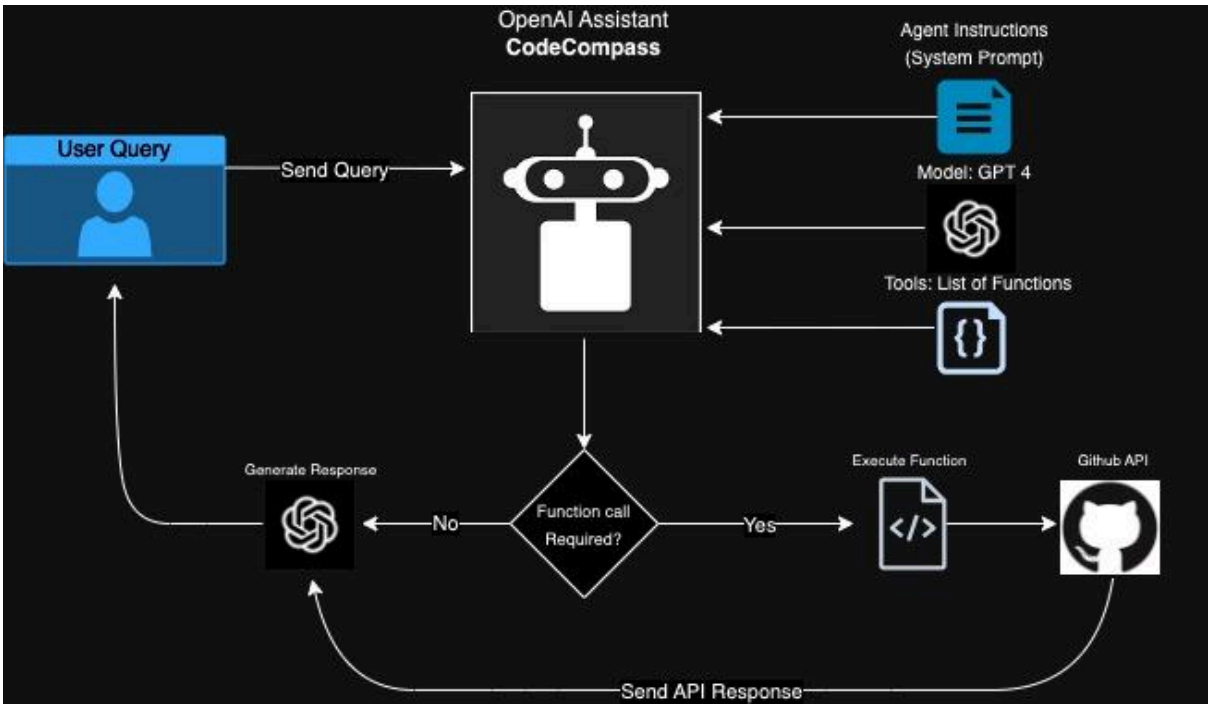
OpenAI Assistant

- Utility Functions: The script *codecompasslib/chatbot/chatbot_management.py* is equipped with a suite of utility functions essential for the chatbot's operational framework.
    - Functions such as *initialize_client*, *create_assistant*, *retrieve_assistant*, *create_message_and_run*, and *load_thread* play critical roles in initializing, retrieving, and managing interactions with the chatbot.

Function Calling and API Interaction

- API Call Execution: Within the same *chatbot_management.py* file, a series of utility functions (*load_tools*, *get_function_details*, *submit_tool_outputs*, *create_function_executor*, and *execute_function_call*) are delineated. These functions collectively facilitate the chatbot's ability to:
    - Present the assistant with a catalog of executable functions alongside their respective instructions and descriptions.
    - Execute the called function as directed by the assistant's input.
    - Relay the API response outputs back to the assistant for interpretation and user feedback.

Chatbot System Design



MLOps

Testing Framework

Located in the test/ directory

Data Engineering

- Functionality Tests: The *test_functional.py* file, targeting the codecompasslib/api library, incorporates 10 functional tests. These tests are crucial for verifying the correct extraction and management of repository and user data.

Storage Tests

- Google Drive Integration: With 3 functional tests encapsulated in *test_drive.py*, the project's interaction with external storage solutions, specifically Google Drive, is validated to ensure seamless data transfer and accessibility.

Chatbot Tests

- Chatbot Functionality: The *test_chatbot.py* script encompasses 13 tests designed to scrutinize various aspects of the chatbot's operational integrity within *codecompasslib/chatbot*, including:
    - API post-request executions and response management.
    - Initialization processes for the OpenAI Assistant.
    - Execution of function calls by the assistant.
    - Secure handling of sensitive information like secrets, API URLs, and system prompts.

Model Tests

- Recommender Model Validation: The addition of *tests/test_model.py* introduces critical testing for the LightGBM recommendation model, embedded within *codecompasslib/models/lightgbm_model*. Key tests include:
    - Data Preprocessing Verification: Validates the preprocessing pipeline ensuring data merging, label assignment, and filtering based on user interactions are performed correctly. Mock functions simulate API responses for user repositories and starred repositories to test the data preparation logic.
    - Recommendation Generation: Tests the generation of personalized repository recommendations, ensuring that the model properly recommends repositories not already owned or starred by the user. Mock functions are utilized to control the testing environment and focus on the model's recommendation capabilities.

Deployment

- Deployed live chatbot application on: https://render.com/
- Will not advertise on the repository due to usage limitations - operating in free plan which deactivates live application every few hours: making the reboot very slow (over 7 mins).
- Easy to fix - would only need to upgrade to a paid plan, but this is outside the project scope.

Workflow Automation

PR Gate Workflow

- Automated Testing Pipeline: Defined in *.github/workflows/pr_gate.yml* this workflow triggers automatically with each pull request and subsequent code merge into the ‘main’ branch. Key features include:
    - Compatibility testing across two Python versions (3.9 and 3.11) to accommodate the diverse development environments of team members.
    - Verification of project requirements against the current codebase.
    - Comprehensive execution of all project tests, covering Data Engineering, Storage, Chatbot functionality, and Model validation.

Frontend

Repository Recommender Interface

- Streamlit Application: Located at *frontend/recommender/app.py*, this Streamlit interface is crafted to enhance user interaction by accepting a 'target user' input. Upon submission, it dynamically displays repositories recommended by our LightGBM model, tailored to the user's preferences and interests.

Chatbot Interface

- Streamlit Application: The *frontend/chatbot/app.py* serves as a separate Streamlit application. This interface allows users to initiate chat sessions, providing a platform for interactive exploration and understanding of repository contents and functionalities.

Libraries and Repository Design

Code Organization

- **Structured Repository Design**: Our repository adopts a structured design approach, with an emphasis on clarity and accessibility. Inspired by Miguel Fierro's "*Project Template*," our project's architecture ensures that notebooks, scripts, and library calls are systematically organized. This methodical arrangement facilitates ease of navigation and enhances collaborative development efforts.

CodeCompass Library

- **Internal Functionalities**: Central to our project is the *codecompasslib* library, which encapsulates a suite of internal functions critical to the operation of CodeCompass. This library includes all the necessary tools and utilities, supporting everything from data retrieval and processing to the implementation of the recommendation system and chatbot functionalities.

Code Release Strategy

- Versioned Deployments: CodeCompass has undergone several developmental milestones, marked by versioned releases to document progress and updates:
    - v1.0.0 (pre-release) - Corresponds to our Midterm Review, signifying a significant phase of development where basic core functionalities were presented.
    - v2.0.0 (official first release)- Corresponds with our final product

Demonstration and Usage

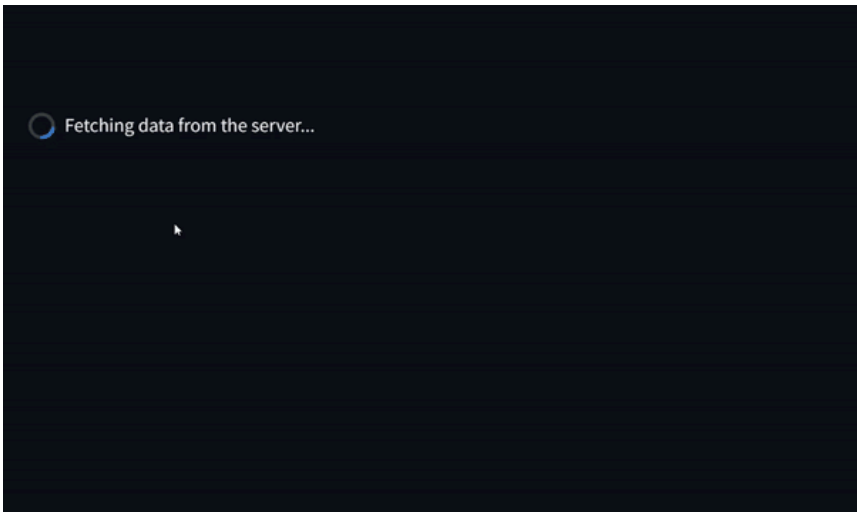
Setup

1. **Clone the repo**: git clone *https://github.com/RecandChat/CodeCompass*
2. **Change to the project directory**: *cd CodeCompass*
3. **Create and activate virtual environment**: *python -m venv venv source venv/bin/activate*
4. **Install dependencies**: *pip install -r requirements.txt*
5. **Create Secrets Directory at the root (*secrets/*)**
    a. askthecode_api: Your API URL wrapper for the chatbot to make requests.
    b. github_token: Your GitHub API Token.
    c. instructions: Your chatbot system prompt.
    d. openAI_key: Your OpenAI API key.

Recommender

Run: *streamlit run frontend/recommender/app.py*

- Full Demo Video and Explanation
- Gif link



Chatbot

Run: streamlit run frontend/chatbot/app.py

- Full Demo Video
- Gif Link

Recommended Prompts

- Tell me about the following repository: https://github.com/recommenders-team/recommenders
- Summarize and tell me about what the code in the 'lightgbm_utils.py' file does
  Where in the repo can I find information about collaborative filtering?
- What are the current branches in the repo?
- What are some commits related to 'Documentation'?
- Find me similar repositories related to "recommenders" or "recommendation engines"