УДК 004.89 DOI: https://doi.org/10.30837/IYF.IIS.2024.050

СТВОРЕННЯ АСИСТЕНТІВ НА БАЗІ ШТУЧНОГО ІНТЕЛЕКТУ

Талах В. О.
Науковий керівник – к.т.н., доц. Любченко В. А.
Харківський національний університет радіоелектроніки, каф. ІТМ
м. Харків, Україна
e-mail: vladyslav.talakh@nure.ua

The text explores the popular trend of creating AI assistants through generative artificial intelligence. It addresses challenges in updating neural network knowledge and introduces LangChain as a framework for assistant development. The process includes information retrieval, storage, search, and response generation. The example used for illustration involves developing an assistant for document search, providing insights into the overall development process and the versatility of LangChain.

Популярним в наш час напрямком є створення асистентів на основі штучного інтелекту. Такий асистент використовує генеративний штучний інтелект для надання допомоги в різних завданнях. Багато компаній вже розробляють своїх асистентів, які можуть бути окремими продуктами або інтегруватися в уже існуючі.

Нейронні мережі використовують статичні знання, а їх постійне оновлення може бути складним процесом та потребувати великої кількості ресурсів. Тому при створенні подібних асистентів можна зустрітись із проблемою обмеженості знань нейронних мереж особливо про приватну інформацію, малий бізнес чи просто не дуже популярну інформацію.

Через це постає задача в створенні системи, яка містить актуальну інформацію та не потребує постійного навчання моделі. Так як генеративний штучний інтелект може отримувати контекст, в якому користувач пише свій запит, можемо використовувати цей контекст як спосіб для надання моделі усієї потрібної інформації. Одним із засобів для розробки асистентів на базі штучного інтелекту є фреймворк LangChain. Розробка асистента включає п'ять основних кроків: отримання, збереження, пошук і генерація інформації. В якості прикладу буде розглядатися випадок створення асистента, який виконує пошук для працівників в документації компанії, що зберігається в системі Confluence. В якості моделі штучного інтелекту для прикладу буде використовуватись GPT від OpenAI.

Перший крок – отримання інформації. LangChain дозволяє отримати дані з різних ресурсів: від звичайних локальних файлів до інтернет ресурсів. В даній задачі цікавить інформація з Confluence. Для цього в LangChain присутній клас ConfluenceLoader. Він завантажує документи з Confluence та повертає їх звичайним текстом, залишаючи лише символи переносу строки.

Перед збереженням даних їх потрібно розділити. Тим самим модель буде отримувати менше інформації, яка не стосується заданого користувачем питання, що впливає на якість відповіді. Для збереження логічної цілісності даних, можна розділити, наприклад, по заголовкам, але, враховуючи формат отриманих даних, будемо використовувати RecursiveCharacterTextSplitter. Він розбиває текст по кількості символів, проте робить це до кінця слова, речення, абзацу, тощо (задається окремим параметром), що дозволяє зберегти цілісність даних. Для зберігання інформації використовуються векторні бази даних. Векторні БД дають можливість зберігати інформацію векторами, що в подальшому буде потрібно для пошуку інформації в них. Для цього, отримані в попередньому кроці, чанки потрібно токенізувати (перетворити на ембедінги). Використаємо модель ada v2 від OpenAI.

Наступним кроком буде пошук даних. Розглянемо алгоритм пошуку stuff-search. Він базується на підрахунку Евклідової відстані між запитом користувача та чанками в БД, з яких береться n найближчих для подальшої генерації. Перед пошуком запит користувача потрібно також перетворити на ембедінги. Цей алгоритм є найпростішим, який надає LangChain, проте може підійти для більшості задач.

Останній етап – генерація відповіді, включає створення темплейту, який визначає структуру відповіді та надає інструкції для генеративного штучного інтелекту. В інструкціях можна описати роль моделі, які дані вона отримає, що з цими даними потрібно зробити, що робити у випадку відсутності даних чи їх невідповідності запиту користувача, формат і манеру відповіді, та багато чого іншого. Інструкції допомагають корегувати модель та направляти її в правильне русло генерації.

В цілому алгоритм створення подібних асистентів доволі схожий. В залежності від задачі відрізняються лише методи отримання інформації, її розбиття, пошуку та темплейт. LangChain в свою чергу полегшує процес розробки, пропонуючи велику кількість заготовленого функціоналу, тим самим розробнику не потрібно замислюватись над імплементацію усіх кроків і дає можливість експериментувати на різних етапах роботи асистента. Проте варто зауважити, що він не є повним вирішенням усіх задач і може потребуватись написання власної логіки, хоча за потреби можна замінити деякі кроки своїми рішеннями, як то отримання інформації, її розбиття, пошук та інше.

Список використаних джерел:

1. Using langchain for Question Answering on Own Data. Medium. URL: https://medium.com/@onkarmishra/using-langchain-for-question-answering-on-own-data-3af0a82789ed (дата звернення: 23.11.2023).

2. LangChain documentation. URL: https://python.langchain.com/ docs/get_started (дата звернення: 25.11.2023).