ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Побудова Чат-ботів з використання штучного інтелекту»

на здобуття освітнього ступеня Бакалавра
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Джозеф Ель-Кафарна
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти Джозеф Ель-Кафарна
група КНД-42

Керівник: к._т._н._доцент_Щербина_
науковий ступінь, вчене звання _Ірина_Сергіївна_

Рецензент:
науковий ступінь, вчене звання _____
(Ім'я, ПРІЗВИЩЕ)

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Бакалавр

Спеціальність Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Ель-Кафарна Джозеф Мухаммедович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Побудова Чат-ботів з використання штучного інтелекту»

керівник кваліфікаційної роботи Ірина Щербина к.т.н., доцент,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» 02.2024р. №36

2. Строк подання кваліфікаційної роботи «31» травня 2024р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, принципи створення чат-ботів з ШІ, Вибір технологій для побудування чат-ботів.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Принципи створення Чат-ботів зі Штучним Інтелектом
Дослідження та вибір технологій розробки Чат-ботів, що використовують великі мовні моделі
Розробка Чат-ботів з використанням великих мовних моделей LLAMA та ORCA

5. Перелік графічного матеріалу: *презентація*

1. Огляд штучного інтелекту та чат-ботів.
2. Типи чат-ботів.
3. Поняття LLM.
4. Мова програмування Python.
5. Docker.
6. Вибір LLM моделі.
6. Chainlit.
7. LangChain.
8. Демонстрація отриманого результату.

6. Дата видачі завдання «23» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Принципи створення Чат-ботів з ШІ	01.03-25.03.24	виконанно
2	Дослідження та вибір технологій розробки чат-ботів, що використовують великі мовні моделі	26.03-10.04.24	виконанно
3	Бібліотеки та Фреймворки для побудови Чат-Ботів , що використовують великі мовні моделі	11.04-24.04.24	виконанно
4	Розгортання середовища розробки та встановлення потрібних додатків	25.04-10.05.24	виконанно
5	Побудова великих мовних моделей	11.05-17.05.24	виконанно
6	Використання Chainlit для створення інтерфейсу чат-бота	18.05-22.05.24	виконанно
7	Огляд тестування програми	23.05-24.05.24	виконанно
8	Покращення та виправлення багів	24.05-25.05.24	виконанно
9	Оформлення роботи: висновки, реферат	26.05-30.05.24	виконанно

Здобувач вищої освіти _____ Джозеф Ель-Кафарна
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Ірина Щербина
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 68 стор., 2 табл., 25 рис., 12 джерел.

Наукове завдання – Розробка та побудування чат-ботів з використання ШІ за допомогою великі мовні моделі.

Мета роботи – розуміння та підвищення роботи при використанням чат-ботів.

Об'єкт дослідження – процедура побудови та розвитку чат-ботів з використанням штучного інтелекту.

Предмет дослідження – використання великі мовні моделі при побудови чат-ботів.

Короткий зміст роботи: Висновки цієї роботи стосуються еволюції та розвитку чат-ботів з використанням технологій ШІ та LLM.

Огляд Чат-GPT та Великі мовні моделі.

Вибір технологій для розробки чат-ботів.

Будування чат-бота за допомогою бібліотеки та фреймворки.

Тестування, виправлення та демонстрація результату завершоного проекту.

КЛЮЧОВІ СЛОВА: ЧАТ-БОТ, LLAMA, ORCA,CHAINLIT, LANGCHAIN, DOCKER, ЧАТ-GPT, PYTHON, ВЕЛИКІ МОВНІ МОДЕЛІ , VSCODE.

ABSTRACT

Text part of the master's qualification work: 68 pages, 25 pictures, 2 tables, 12 sources.

This work research is an overview of how AI developed and built chatbots as well as to enhance your understanding of their functionality while building chatbots.

Object of research – The process of building and developing chatbots using AI.

Subject of research – The usage of LLM in building and developing chatbots.

Summary of the work: Conclusion of this work is about the evolution and development of chatbots using AI and LLM technology.

Overview of ChatGPT and LLM

Choosing technologies for building chatbots.

Building Chatbot with the provided technologies.

Testing and demonstrating the result of the finished project.

KEYWORDS: CHATBOT, LLAMA, ORCA,CHAINLIT, LANGCHAIN, DOCKER, CHATGPT, PYTHON, LLM, VSCODE.

ЗМІСТ

ВСТУП

Створення чат-ботів з використанням ШІ стає дедалі популярнішим в останні роки завдяки прогресу в технології обробки природної мови. Чат-боти, або розмовні агенти, - це системи штучного інтелекту (ШІ), створені для імітації людського спілкування з користувачами. Чат-боти використовуються в різних галузях, включаючи обслуговування клієнтів, електронну комерцію, охорону здоров'я та інші, для надання швидких та ефективних відповідей на запити споживачів та покращення користувацького досвіду. У роботі розглянемо основні принципи побудови чат-ботів з використанням штучного інтелекту (ШІ), включаючи основні концепції, інструменти та методології, необхідні для розробки складних розмовних агентів, проаналізуємо багато категорій чат-ботів, включаючи чат- ботів на основі правил, чат-ботів на основі пошуку та генеративних чат-ботів, а також їхні індивідуальні переваги та недоліки, розглянемо фундаментальні елементи побудови чат-ботів, включаючи розуміння природної мови (NLU), генерацію природної мови (NLG), управління діалогом і відстеження контексту та Крім того, різноманітні фреймворки та платформи штучного інтелекту, доступні для створення чат-ботів, зокрема Dialogflow, Microsoft Bot Framework, IBM Watson та інші. Тому тема бакалаврської роботи є актуальною.

1 ПРИНЦИПИ СТВОРЕННЯ ЧАТ-БОТІВ ЗІ ШТУЧНИМ ІНТЕЛЕКТОМ

1.1 Еволюція розвитку та сфери застосування великих мовних моделей

У сучасному світі чат-боти стають все більш поширеними, оскільки вони дають компаніям нові методи спілкування зі своїми клієнтами. А завдяки їхній здатності відповідати на типові запити та використанню штучного інтелекту й обробки природної мови, щоб звучати як справжня людина, ми називаємо їх віртуальними агентами. Це не лише покращує клієнтський досвід, але й підвищує загальну ефективність бізнесу. Очевидно, що фірми зараз впроваджують чат-ботів, щоб зробити пріоритетом задоволення та залучення клієнтів. Вони починають усвідомлювати, що чат-боти мають потенціал для зменшення витрат за рахунок усунення потреби у звичайних операторах кол-центру або співробітниках служби підтримки. Мета цієї роботи - проаналізувати використання чат-ботів на ринку, дослідити технології та бізнес-стратегії, пов'язані з впровадженням чат-ботів, а також оцінити вплив розвитку штучного інтелекту на корпоративну продуктивність.

Загальна схема розвитку (Рисунок 1.1).

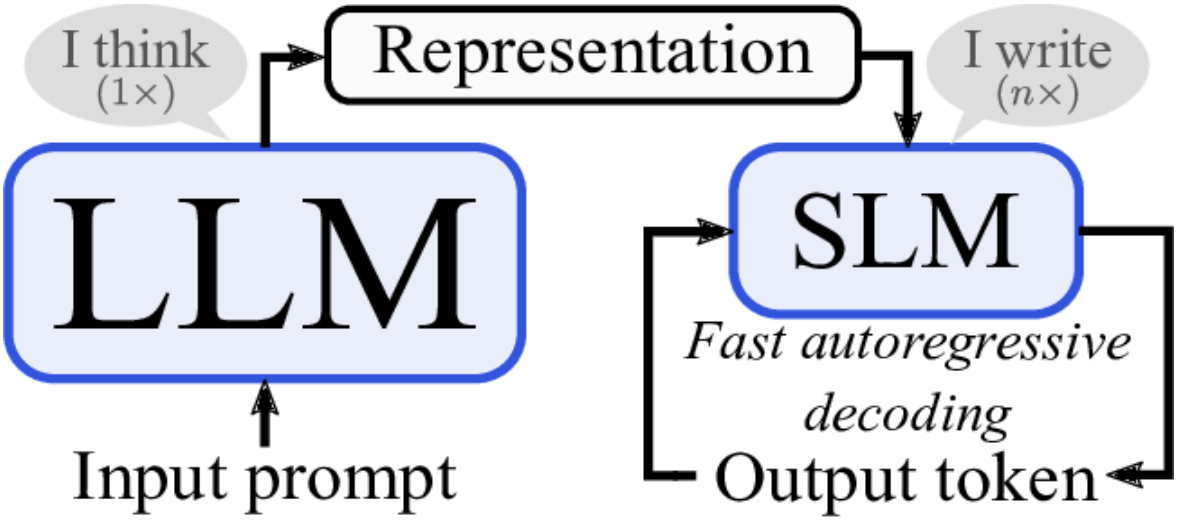


Рисунок 1.1 - Загальна схема роботи LLM.

Великі мовні моделі (LLM) є важливою частиною штучного інтелекту і різновидом чат-ботів, які можуть використовувати обробку природної мови для надання відповідей і більш динамічного та настроюваного обміну з користувачем. Останні технологічні досягнення дозволили чат-ботам LLMS швидко покращити свою здатність надавати користувачам найточніші та найактуальніші відповіді. Ці чат-боти можуть допомогти студентам дізнатися більше, виступаючи в ролі персональних помічників.Великі мовні моделі, такі як GPT, стали можливими завдяки тому, що LLM сильно змінилася з часом.

Нижче наведено хронологію розвитку технології LLM, а також перелік деяких важливих переломних моментів:

- 1950-ті - Ранні мовні моделі: Сфера мовного моделювання починається з ранніх спроб створити обчислювальні моделі мови.
- 1970-ті N-грамові моделі: Стають популярними N-грамові моделі, які використовують статистичні методи для передбачення наступного слова в послідовності на основі частоти його появи в навчальному корпусі.
- 1990-ті - Приховані марковські моделі (HMM) - використовуються для моделювання мови, включаючи контекстну інформацію та ймовірності для передбачення наступного слова.
- 2000s: Нейромережеві мовні моделі: Моделі на основі нейронних мереж змінюють спосіб моделювання мови, використовуючи глибоке навчання для більш точних прогнозів.
- У 2010-х роках рекурентні нейронні мережі (RNN) та мовні моделі на основі RNN, такі як Long Short-Term Memory (LSTM), стали відомими завдяки тому, що вони могли знаходити довготривалі зв'язки в тексті.
- 2017 GPT-1: Це підводить нас до серії GPT, яку вони розпочали з випуском GPT-1. Він використовує трансформаторну архітектуру і має найкращу швидкість у завданнях генерації мови.
- 2018 GPT-2: GPT-2 розвинув успіх GPT-1, додавши більшу модель з більшою кількістю факторів, щоб зробити генерацію мови ще більш логічною та усвідомленою в її оточенні.

- 2020 GPT-3: Маючи 175 мільярдів параметрів, GPT-3 є наступним великим кроком вперед у моделюванні великих мов. Він чудово справляється з такими завданнями обробки природної мови, як переклад, реферування та відповіді на запитання.
- 2022 GPT-4: Серія GPT продовжує вдосконалюватися. З виходом GPT-4 і наступних версій мовні моделі зможуть робити ще більше. Вони розширюють межі обробки природної мови і дають ще більш точні та релевантні відповіді.
- 2023 Інтеграція та спеціалізація: Впровадження GPT-4 та інших мовних моделей (LLM) у різні галузі швидко розвивається завдяки підвищенню ефективності моделей і вдосконаленню методологій їхнього налаштування. GPT-4 було адаптовано для таких цілей, як вивчення юридичних документів, діагностика медичних станів і прогнозування фінансових результатів. Покращені багатомовні можливості дозволяють моделям стабільно добре працювати з широким спектром мов і діалектів. Зростаюча важливість етичних питань і правил у використанні ШІ зумовлена необхідністю вирішувати проблеми, пов'язані з упередженістю, прозорістю та впливом великомасштабних моделей на навколишнє середовище.
- 2024 GPT-4o : Наразі GPT-4o перевершує всі інші моделі за здатністю розуміти та обговорювати надані вами фотографії. Як ілюстрація, зараз ви можете зробити знімок меню, написаного іноземною мовою, і вступити в розмову з GPT-4o, щоб перекласти його, отримати інформацію про історичну та культурну цінність страви та отримати пропозиції. У майбутньому вдосконалення уможливить покращений, безперебійний аудіо зв'язок у реальному часі та можливість брати участь у відео розмовах у реальному часі за допомогою ChatGPT.

Еволюція розвитку LLM (Рисунок 1.2).

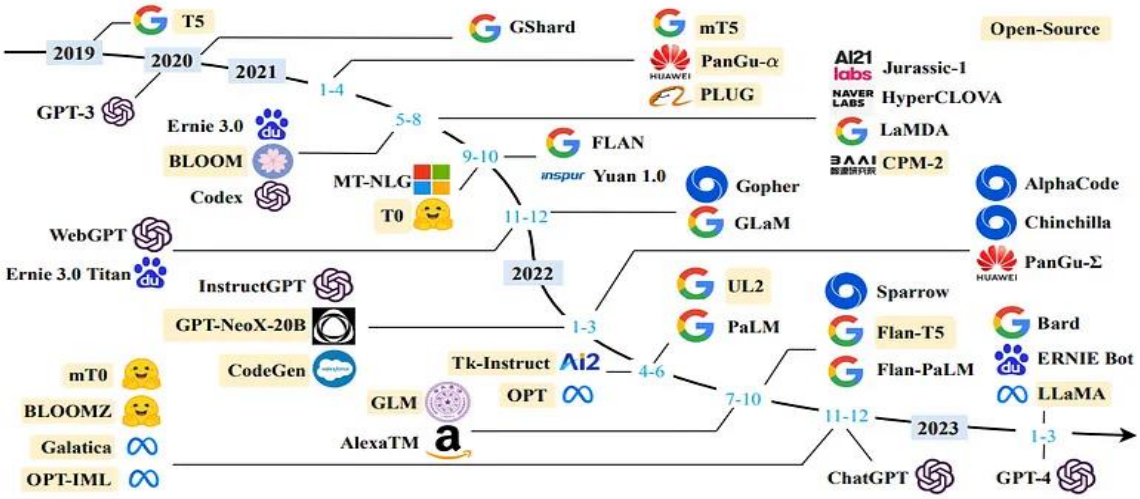


Рисунок 1.2 - Еволюція розвитку LLM.

З представлення моделі ми бачимо, що LLM справила великий вплив на світ інтелектуальних машин, змінивши те, що вони можуть робити і як вони можуть це робити. Наприклад, ChatGPT зробив сферу чат-ботів більш технологічною, дозволивши більш цікаві та схожі на людські взаємодії. Наприклад, він може розуміти і відтворювати людську мову з дивовижною точністю, що дозволяє чат-ботам розмовляти більш природно та інтуїтивно. Це підводить нас до того, що модель LLM зробила чат-ботів набагато кращими у спілкуванні, допомагаючи їм розуміти те, що говорять, і відповідати так, щоб це мало сенсу.

Одним із шляхів розвитку чат-ботів на базі LLM у майбутньому є можливість працювати з новими технологіями. Наприклад, додавання до чат-ботів технологій розпізнавання мови та розуміння природної мови може зробити їх простішими у використанні та надати користувачам більш інтерактивний і природний досвід. Ми розглянемо це більш детально в наступному розділі.

Новітні інструменти та комбінації LLM дали нам нові способи вдосконалення чат-ботів. Ось деякі з таких інтеграцій:

Додавання доповненої реальності (AR). Технологія доповненої реальності дала користувачам візуальний і динамічний досвід, що дозволило чат-ботам на базі

LLM показувати користувачам інформацію або розмовляти з ними в більш цікавий і реалістичний спосіб.

Додавання функцій віртуальної реальності (VR). Додавання чат-бота у віртуальний світ має величезні перспективи, оскільки створює реалістичний віртуальний світ, де користувач може спілкуватися з чат-ботом, що дуже круто.

Голосові асистенти та розумні пристрої: люди все частіше використовують ці помічники та пристрої для виконання повсякденних справ та зв'язку. Ці гаджети дозволяють людям спілкуватися з речами без використання рук. Це, зокрема, полегшило людям використання чат-ботів у різних місцях, як-от вдома, в машині чи на роботі.

Метою цього дослідження є визначення рівня, до якого ChatGPT може бути корисним у маркетингових стратегіях, а також викликів і можливостей, які він представляє для підвищення ефективності споживчої комунікації

Застосування чат GPT підвищує ефективність маркетингових ініціатив завдяки сегментації аудиторії, точному таргетингу та прогностичному аналізу майбутньої поведінки споживачів. GPT-чат має можливість класифікувати дані про клієнтів за кількома параметрами, що дозволяє розробляти маркетингові кампанії більш точно і цілеспрямовано. Крім того, використання чатів GPT дає змогу створювати точні стратегії таргетингу, що дозволяє успішно залучати певні сегменти клієнтів на основі їхніх уподобань та попередніх взаємодій.Розвиток і впровадження інструментів чат GPT в маркетингу можуть стати вагомою конкурентною перевагою для бізнесу у сучасному цифровому світі.

Основними сферами, в яких вони успішно застосовуються, є цифровий маркетинг, електронна комерція та контент-маркетинг:

Основні сфери застосування Чат GPT в електронній комерції (Рисунок 1.3).

Рисунок 1.3 - Основні сфери застосування Чат GPT в електронній комерції.

Цифрові маркетингові агенти можуть використовувати чат-GPT для створення цікавих чат-ботів, які привертають увагу людей. Ці боти також можуть бути використані для пошуку нових потенційних клієнтів. Дзвінки та електронні листи, які надходять від потенційних клієнтів, називаються лідами. Завдання бота - перетворити їх на покупців. Чат-боти можуть проводити розмовні опитування, розповідати людям про товари та послуги компанії та збирати інформацію про клієнтів. Це дає можливість знаходити нових потенційних клієнтів і ефективно комунікувати з ними. Маркетологи можуть використовувати чат-GPT для створення персоналізованих текстів і оголошень для соціальних мереж.

В електронній комерції чат GPT може використовуватися для оптимізації процесу покупок та оформлення замовлень. Він може надавати допомогу клієнтам у виборі товарів в онлайн-магазині, відповідати на запитання щодо наявності або доставки та значним чином спрощувати процес обробки замовлень.

1.2 Основні принципи функціонування ChatGPT

ChatGPT - це новий інструмент обробки природної мови, який змінив спосіб спілкування між людьми та комп'ютерами. ChatGPT створює розмови, які звучать дуже схоже на реальні розмови між людьми, використовуючи методи машинного та глибокого навчання. Він дуже добре знаходить закономірності в написаному, розуміє, що це означає, і відповідає так, ніби з вами розмовляє людина. Використовуючи методи глибокого навчання, він може записувати синтаксичні та семантичні патерни, які роблять результати більш реальними.

Гнучке навчання також допомагає йому постійно покращувати розуміння мови, піддаючи його різноманітним людським взаємодіям. Це свідчить про те, що ChatGPT є частиною змін у тому, як штучний інтелект впливає на багато сфер життя, включаючи бізнес, фінанси, військову сферу та безпеку.

ChatGPT Generative Pre-trained Transformer (GPT) - це генеративний попередньо навчений перетворювач. OpenAI створив GPT - сімейство моделей великих мов, які використовують глибоке навчання для створення тексту чату, який звучить так, ніби його написала людина.

OpenAI, компанія, яка розробила ChatGPT, нещодавно припинила вимагати від користувачів входити в систему, перш ніж вони зможуть нею користуватися. Тепер можна легко потрапити до ChatGPT, перейшовши на сайт chat.openAI.com, який є його веб-сайтом. І вони створили додаток для вашого iPhone або Android, який дозволяє вам потрапити на нього.

Багато людей побоюються, що робити зі штучним інтелектом замінять людей, або зроблять їх менш розумними. Наприклад, робот може швидко написати статтю на будь-яку тему, хоч і не завжди правильно. Це може означати, що людина-письменник не потрібна.

Робот також може написати повне есе за лічені секунди. Це полегшує учням можливість списувати або не навчитися писати правильно. Деякі шкільні округи навіть заблокували доступ до ChatGPT, коли він тільки з'явився, через це.

Зараз не тільки багато з цих шкіл вирішили відкрити технологію, але й деякі коледжі та університети почали включати в свої курси теми, пов'язані зі штучним інтелектом, щоб залучити більше студентів.

Ще одне занепокоєння щодо ШІ-робота полягає в тому, що він може поширювати неправдиву інформацію. Бот може видавати неправдиву інформацію, оскільки він не пов'язаний з інтернетом.

Бот каже: "Мої відповіді не повинні сприйматися як факт, і я завжди заохочую людей перевіряти будь-яку інформацію, яку вони отримують від мене або з будь-якого іншого джерела". OpenAI також повідомляє, що іноді ChatGPT пише "відповіді, які звучать правдоподібно, але є помилковими або не мають сенсу".

Нарешті, існують моральні проблеми з даними, на основі яких навчався ChatGPT, оскільки компанія перерила весь інтернет, щоб навчити робота.

Занепокоєння щодо конфіденційності викликає той факт, що він автоматично використовує те, як люди користуються безкоштовною версією робота, щоб ще більше тренувати свої моделі. Однак користувачі OpenAI можуть вимкнути налаштування навчання.

Незважаючи на те, що ChatGPT має багато чудових функцій, у нього все ще є деякі проблеми. Користувачам доводиться перефразовувати свої запитання кілька разів, щоб ChatGPT зрозумів, що вони мають на увазі. Ще більша проблема полягає в тому, що відповіді не завжди хороші. Вони можуть звучати розумно, але не завжди мають сенс або можуть бути занадто довгими.

Переваги та недоліки використання ChatGPT зведені в таблицю 1.1.

Таблиця 1.1. Переваги та недоліки використання ChatGPT у різних сферах.

Переваги	Недоліки
- Забезпечує доступність діалогових агентів 24/7.	- Потенціал для упереджених або неточних відповідей через навчальні дані.
- Забезпечує масштабованість для обробки великих обсягів запитів.	- Відсутність емоційного взаєморозуміння, що призводить до безособової взаємодії.
- Сприяє взаємодії природною мовою, покращуючи користувацький досвід.	- Обмежена здатність вирішувати складні проблеми або вести складні розмови.
- Пропонує узгоджені відповіді на всі взаємодії.	- Потребує постійного моніторингу та управління для забезпечення якості та адекватності реагування.

Ми бачимо, що ChatGPT, як і всі мовні моделі, має недоліки і може давати неправильні або дивні відповіді, тому важливо двічі перевіряти інформацію, яку він надає.

Google, Wolfram Alpha і ChatGPT дозволяють користувачам вводити текст в один рядок і дають їм текстові відповіді. Google повертає результати пошуку, які є

списком веб-сторінок та історій, які (сподіваємось) містять інформацію, що має відношення до пошукових запитів. Wolfram Alpha зазвичай дає відповіді, пов'язані з математикою та аналізом даних.

З іншого боку, ChatGPT відповідає на основі того, про що запитує користувач і як він це запитує. Ми не можемо попросити Google або Wolfram Alpha написати історію або програмний модуль, але ChatGPT може.

Нейрони, які використовуються для обробки вхідних даних природною мовою, називаються архітектурами-трансформерами. Нейронна мережа обробляє інформацію через шари вузлів, які пов'язані між собою. Це схоже на те, як працює мозок. Нейромережу можна порівняти з хокейною командою. Кожен гравець виконує певну роботу, але шайба передається туди-сюди між гравцями, щоб кожен міг допомогти забити гол.

Трансформаторний дизайн працює з групами слів, використовуючи "самоуважність", щоб зрозуміти, які слова є найбільш важливими для прогнозування. Самоуважність - це як повернення до попереднього рядка або абзацу в книзі, щоб отримати довідкову інформацію, необхідну для розуміння нового слова. Трансформер дивиться на всі слова поспіль, щоб зрозуміти, що вони означають і як пов'язані між собою.

Важливо також пам'ятати, що ці моделі можуть вивчати тенденції та упередження на основі навчальних даних, а це означає, що вони можуть створювати шкідливі або упереджені матеріали. Компанії, які використовують ці методи, намагаються забезпечити "захисні перила", але ці перила самі по собі можуть спричинити проблеми. Це тому, що кожна людина має свою точку зору, і якщо ви намагаєтеся зупинити упередженість, засновану на одній школі думки, хтось інший може сказати, що ви упереджені. Як наслідок, створити глобального робота важко, тому що люди надзвичайно складні.

Тепер давайте розділимо роботу ChatGPT на два простих кроки: навчання та відповідь на запитання. Навчання моделі ChatGPT складається з двох етапів:

Попереднє навчання: На цьому етапі ми використовуємо багато даних з Інтернету для навчання GPT-моделі, яка є трансформатором, що працює лише з декодером. Мета полягає в тому, щоб навчити модель вгадувати, які слова підуть

далі, на основі речення, яке є абсолютно правильним і має сенс, як і дані в Інтернеті. Після того, як модель пройшла етап попереднього навчання, вона може закінчувати задані слова, але не відповідати на запитання.

Точне налаштування: Цей етап складається з трьох кроків, які перетворюють вже вивчену модель на модель ChatGPT, яка може відповідати на запитання:

- Отримайте навчальні дані (запитання та відповіді) і використовувати їх для доопрацювання вже вивченої моделі.
- Отримайте більше інформації (запитання та кілька відповідей) і навчіть платіжну модель сортувати ці відповіді від найважливіших до найменш важливих.
- Налаштуйте модель за допомогою навчання з підкріпленням (PPO-оптимізація), щоб отримати більше правильних відповідей від моделі.

Щоб відповісти на підказку, потрібно зробити кілька кроків.

- Спочатку людина вводить питання повністю: "Поясніть, як працює алгоритм класифікації".
- Другий крок: Питання надсилається до частини, яка перевіряє інформацію. Ця частина перевіряє питання, щоб переконатися, що воно не порушує жодних правил безпеки, і видаляє неприйнятні питання.
- Третій крок. Якщо вхідні дані проходять фільтрацію контенту, вони надсилаються до моделі ChatGPT. Якщо вхідні дані не проходять фільтрацію контенту, вони одразу переходять до створення шаблонної відповіді.
- П'ятий і шостий крок: Відповідь знову надсилається до компонента контролю контенту після того, як вона була створена моделлю. Таким чином, створена відповідь гарантовано буде безпечною, нешкідливою, справедливою тощо.
- Останній крок: Користувач бачить вхідні дані, якщо вони пройшли фільтрацію контенту. Якщо вхідні дані не пройшли через фільтрацію контенту, користувач бачить шаблонну відповідь і переходить до створення шаблонної відповіді.

Алгоритм тренування Чат GPT (Рисунок 1.4).

Алгоритм відповіді на промпти (Рисунок 1.5).

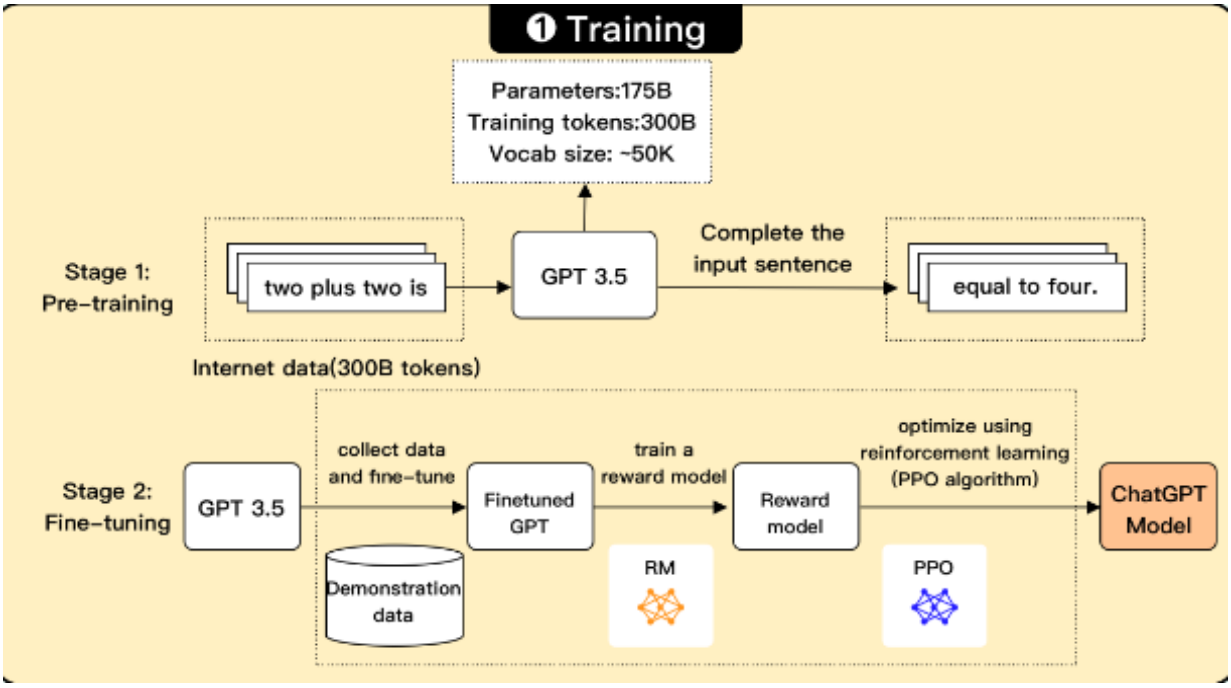


Рисунок 1.4 - Алгоритм тренування Чат GPT.

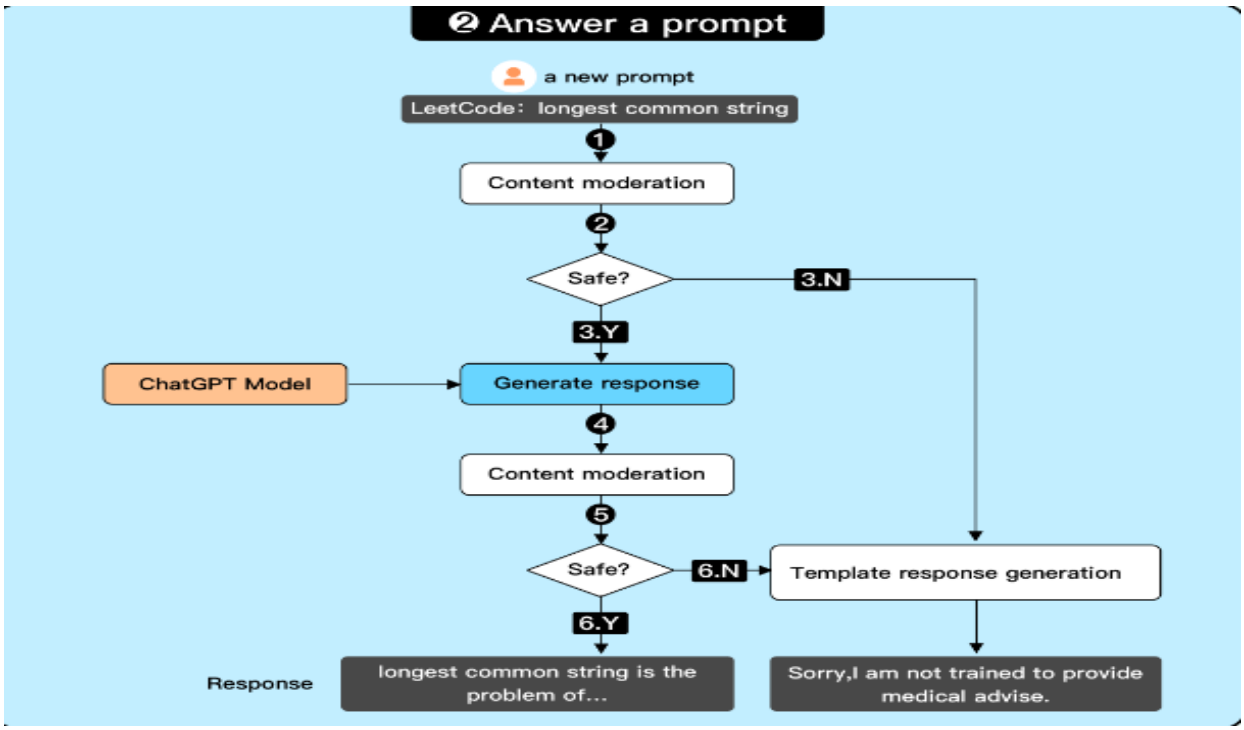


Рисунок 1.5 - Алгоритм відповіді на промпти.

1.3 Можливості використання великих мовних моделей при розробці чат- ботів

Останніми роками ШІ та LLM продемонстрували нам вражаючі можливості способу мислення про технологію та її ефективність у різних галузях та сферах, включаючи сферу систем чат-ботів.

На кожному веб-сайті чи додатку, який ми відвідуємо, є спливаюче вікно чату, яке допомагає нам вирішити будь-які питання чи проблеми, які можуть виникнути. Якщо ми озирнемося навколо, то побачимо, що чат-боти стали важливою частиною нашого повсякденного життя. Те, як компанії спілкуються з людьми, яких вони обслуговують, змінилося, оскільки вони пропонують швидку допомогу та унікальний досвід. Магістерська програма мала великий вплив на розвиток роботів, полегшивши їм розуміння того, що говорить користувач, і даючи кращу, більш задовільну відповідь. Завдяки цьому роботи стали дуже корисними і допомагають у багатьох сферах, таких як охорона здоров'я, обслуговування клієнтів і освіта.

У таблиці 1.2. наведено перелік різних сфер бізнесу, роль, яку відіграють чат-боти в кожній з них, а також деякі приклади того, що чат-боти роблять для своїх користувачів.

Таблиця 1.2. Індустрії, де застосовуються чат-боти.

Промисловість	Мета використання	Приклади та їх завдання
Служба підтримки клієнтів	- Пропонуючи цілодобову допомогу та вирішуючи запити з високою швидкістю та точністю.	- Чат-бот служби підтримки Apple, що надає технічну допомогу. - Усунення несправностей та інформацію про продукт.
Електронна комерція	- Покращення клієнтського досвіду завдяки рекомендаціям щодо продуктів, відстеженню замовлень та персоналізованому шопінгу.	- Чат-бот Amazon. - Допомога користувачам у пошуку товарів. - Пропонує рекомендації та відстеження замовлень.

Продовження таблиця 1.2.

Промисловість	Мета використання	Приклади та їх завдання
Охорона здоров'я	- Удосконалення медичної практики шляхом підвищення точності діагностики, прогнозування прогресування захворювання та підтримки прийняття клінічних рішень.	- Чат-бот HealthTap. - Надання медичних консультацій. - Перевірка симптомів - Зв'язок користувачів з лікарями.
Освіта	- Виступають у ролі персональних асистентів, пропонуючи студентам посилену підтримку та настанови.	- Чат-бот Duolingo - Допомога тим, хто вивчає мову, у виконанні практичних вправ, таких як словниковий запас - Підказки з граматики.
Медіа та розваги	- Надання персоналізованих рекомендацій, відповіді на запити користувачів та створення інтерактивного досвіду у сфері медіа та розваг.	- Чат-бот Netflix - Пропонувати фільми на основі історії переглядів користувача. - Відповідаємо на поширені запитання та організовуємо покази.

Продовження таблиця 1.2.

Промисловість	Мета використання	Приклади та їх завдання
Фінанси та банківська справа	- Допомагають клієнтам у фінансових операціях, надаємо індивідуальні консультації та відповіді на банківські запити.	- Чат-бот Bank of America - Допомога користувачам в управлінні акаунтами - Надають користувачам поради щодо планування бюджету та оплати рахунків.
Подорожі та гостинність	- Допомагають мандрівникам з бронюванням, плануванням маршрутів і пропонуємо локалізовані рекомендації щодо визначних пам'яток і ресторанів.	- Чат-бот Expedia - Сприяння у бронюванні авіаквитків та готелів. - Пропонування місцеві визначні пам'ятки та заклади харчування.

Використання ChatGPT підіймає низку мораальних питень Використання роботів на основі LLM змушує людей турбуватися про можливу упередженість, вторгнення в приватне життя та належне використання технологій штучного інтелекту. Важливо переконатися, що чат-боти на основі LLM навчаються на різноманітних і справедливих наборах даних, щоб вони не додавали і не посилювали упередження, які вже є в даних.

Занепокоєння щодо конфіденційності також виникає, коли чат-боти, керовані магістрами права, мають доступ до приватних або секретних даних. Ще одна проблема, яку потрібно виправити, - це те, наскільки надійні нинішні чат-боти на основі LLM. Роботи на основі LLM повинні бути сильними та надійними, щоб добре працювати в широкому діапазоні ситуацій. Завдяки інтерпретованості та

прозорості користувачі можуть розуміти, як чат-боти приймають рішення, і мати уявлення про процеси, що відбуваються в їхній роботі.

Крім того, для того, щоб чат-боти на рівні LLM могли постійно навчатися та адаптуватися до мінливих потреб користувачів, їхніх уподобань та контексту спілкування, дуже важливо впроваджувати постійні процеси навчання та перепідготовки. Це передбачає регулярне оновлення чат-ботів новими даними та взаємодіями, щоб вони залишалися актуальними та реагували на мінливі запити користувачів.

LLM-агенти - це передові системи штучного інтелекту, які поєднують потужність великих мовних моделей з іншими експертними моделями для розширення їх можливостей у різних завданнях, таких як розуміння зображень та семантичне міркування LLM-агенти складаються з декількох компонентів, включаючи інструменти, ядро агента та експертні моделі. Ці компоненти працюють разом для створення модульної та універсальної системи ШІ. Інструменти - це зовнішні ресурси, сервіси або API, які агент може використовувати для виконання певних завдань або розширення своїх можливостей. Інструменти можуть включати бази даних, бази знань і зовнішні моделі.

Наприклад, боти можуть використовувати конвеєр RAG, щоб робити відповіді, які відповідають ситуації. А як щодо ядра агента? Його завдання полягає в тому, щоб контролювати і координувати роботу LLM-агента в цілому. Це та частина, яка надбудовується над самою моделлю LLM. Наші цілі для LLM-агента, інструменти, які він буде використовувати, і його важливі спогади - все це закладено в його серці. Вона також надає агенту індивідуальності, використовуючи ретельно продумані підказки та вказівки, щоб керувати його діями та відповідями.

Агенти LLM відрізняються один від одного, і є багато з них зі своїми варіантами використання, такими як:

- Розмовні агенти. Ці агенти призначені для ведення розмов природною мовою з користувачами, надання інформації, відповідей на запитання та допомоги у виконанні різних завдань. Вони часто покладаються на

великі мовні моделі, щоб розуміти і генерувати відповіді, схожі на людські.

- Агенти, орієнтовані на завдання. Ці агенти зосереджені на виконанні конкретних завдань або досягненні заздалегідь визначених цілей. Вони взаємодіють з користувачами, щоб зрозуміти їхні потреби, а потім виконують дії для задоволення цих потреб. Прикладами є віртуальні асистенти та інструменти автоматизації завдань.
- Креативні агенти Ці агенти здатні створювати оригінальний і творчий контент, наприклад, твори мистецтва, музику чи літературу. Вони можуть використовувати LLM, щоб зрозуміти людські вподобання та мистецькі стилі, що дозволяє їм створювати контент, який резонує з аудиторією.

Розвиток технологій чет-ботів з великими мовними моделями (Рисунок 1.6).

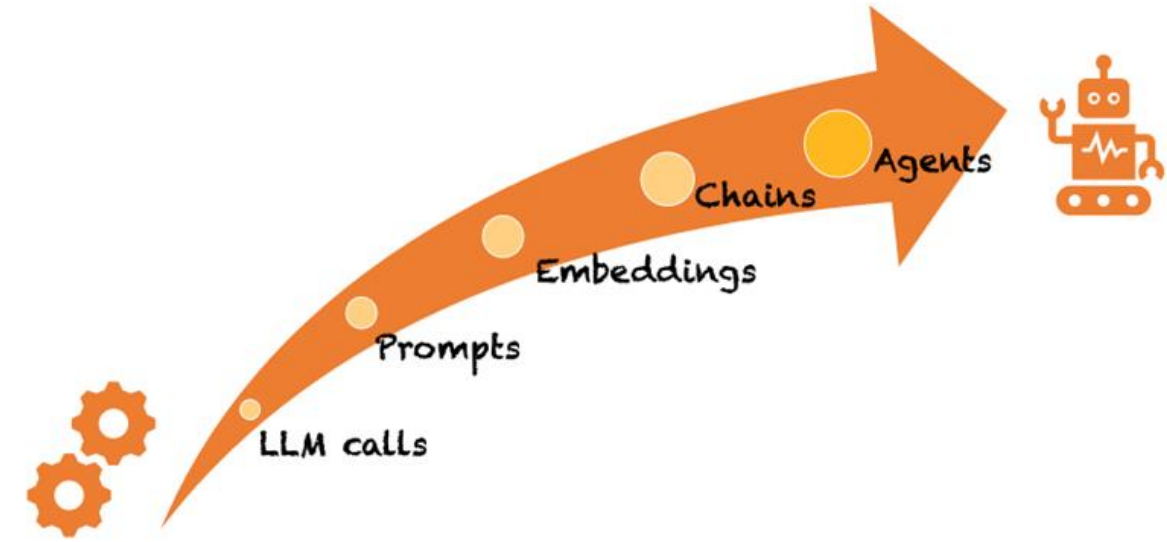


Рисунок 1.6 - Розвиток технологій чет-ботів з великими мовними моделями.

Можна навести приклад таких відомих агентів LLM, як:

– VisualGPT - VisualGPT пов'язує ChatGPT з серією моделей Visual Foundation, полегшуючи обмін зображеннями під час розмов.
– Lindy AI - Як особистий асистент зі штучним інтелектом.

- CensusGPT - дозволяє користувачам запитувати про теми, пов'язані з даними перепису.
- Hearth AI - управління агентськими відносинами.
- ChemCrow - ШІ-агент, орієнтований на завдання з хімії, що використовує великі мовні моделі для автономного планування та виконання таких процесів, як органічний синтез та відкриття ліків.

Як бачимо, вони пройшли шлях від простого виконання підказок до агента і мають різноманітне використання в багатьох сферах, таких як навчання та бізнес.

2 ДОСЛІДЖЕННЯ ТА ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ ЧАТ-БОТІВ, ЩО ВИКОРИСТОВУЮТЬ ВЕЛИКІ МОВНІ МОДЕЛІ

2.1 Мови програмування для чат-ботів з LLM

Сьогодні алгоритми та технології перетворилися на потужні, сучасні інструменти з широким спектром застосування. Створення чат-ботів - одна зі сфер, де ці технології стають дедалі популярнішими. І це дасть нам безліч варіантів для розробки чат-ботів.

Таким чином, у цьому розділі ми розглянемо технології та алгоритми, що використовуються при розробці чат-ботів. Чат-боти - це комп'ютерні програми, створені для імітації спілкування людини з користувачем. Вони можуть розуміти людську мову, виявляти закономірності та надавати релевантні відповіді, розшифровуючи вхідні дані користувача за допомогою алгоритмів обробки природної мови (NLP), підходів машинного навчання та технологій штучного інтелекту.

Вибір правильної комп'ютерної мови важливий для створення широкого спектру систем і додатків. Ось чому ми збираємося перерахувати кілька комп'ютерних мов, які можна використовувати для створення чат-ботів.

Python - це комп'ютерна мова високого рівня, яка є інтерпретованою та об'єктно- орієнтованою. Її значення змінюються з часом.

Високорівневі вбудовані структури даних, динамічна типізація та зв'язування роблять її хорошою мовою для швидкої розробки додатків. Її також можна використовувати як мову кодування або для з'єднання частин, які вже існують. Завдяки чіткому, інтуїтивно зрозумілому синтаксису. Python надає пріоритет читабельності, а отже, мінімізує витрати на підтримку програмного забезпечення.

Повторне використання коду та модульність програмного забезпечення заохочується підтримкою модулів та пакетів Python. Всі основні платформи пропонують безкоштовні вихідні коди або бінарні версії інтерпретатора Python та велику стандартну бібліотеку, якими можна ділитися без обмежень.

Java - це популярна мова програмування, яка відома своєю гнучкістю при створенні веб-додатків. Завдяки більш ніж 20-річному досвіду, мільйони додатків

сьогодні розробляються з використанням Java. Java широко визнана у сфері розробки програмного забезпечення завдяки своїй об'єктно-орієнтованій природі, мережево- орієнтованій архітектурі та свободі платформи.

Уявіть собі, що Java - це ретельно продумана робоча станція для лабораторії; вона надає розробникам безпечний простір для експериментів і творчості. Суворе дотримання правил кодування та модульна архітектура забезпечують повторюваність і надійність процесів розробки програмного забезпечення. Подібно до того, як науковці використовують сучасне обладнання для проведення досліджень у багатьох галузях, програмісти використовують можливості Java для створення веб- додатків, які є масштабованими та сумісними, що дозволяє їм процвітати в постійно мінливому цифровому середовищі.

Ruby - це динамічна, об'єктно-орієнтована комп'ютерна мова, яка відома своєю простотою у вивченні та розумінні.

При створенні програми для чат-бота найважливіше - зробити сам процес створення цікавим і легким. Велика бібліотека та зрозумілий стиль мови Ruby дозволяють авторам писати креативний та лаконічний код. Оскільки її можна використовувати з багатьма різними стилями написання коду, Ruby також корисна для веб-розробки, написання сценаріїв та автоматизації.

Незважаючи на те, що більшість з цих мов сумісні з відомими фреймворками глибокого навчання, в наступному розділі буде використано Python через його адаптивність та обмежений набір функцій, які сприяють спрощенню системи.

2.2 Бібліотеки та фреймворки для побудови чат-ботів з великими мовними моделями

Локальний запуск великих мовних моделей (LLM) є дуже поширеною темою зараз, і в наступному списку представлені відомі інструменти, бібліотеки для налаштування, розгортання та структурування чат-ботів на основі LLM.

Hugging Face та інші платформи перетворюються на головні майданчики для створення та обміну моделями, оскільки машинне навчання та штучний інтелект швидко розвиваються.

Hugging Face - це загальнодоступна платформа з відкритим вихідним кодом, яка пропонує величезну колекцію ресурсів для машинного навчання. Вона може похвалитися понад 300 000 моделей, 70 000 наборів даних і більш ніж 150 000 додатків. Ця онлайн-платформа полегшує співпрацю та розробку проектів, що робить її дуже затребуваним інструментом для дослідників і розробників у галузі машинного навчання.

Переваги:

- Це зручна бібліотека для розробників і дослідників, яка надає їм доступ до найсучасніших моделей в NLP без обчислювальних ресурсів.
- Hugging Face має широкий вибір попередньо навчених моделей, які можна швидко налаштувати для виконання конкретних завдань. Це робить їх дуже гнучкими та корисними.
- Hugging Face має власну потужну спільноту та систему підтримки, яка дозволяє користувачам працювати разом, обмінюватися моделями та допомагати робити бібліотеку кращою.
- В обіймах обличчям теж є кілька поганих речей:
- Це значною мірою залежить від попереднього навчання, а це означає, що успіх моделей залежить від того, наскільки добре і на якому обсязі даних вони навчені.
- Оскільки Hugging Face є інструментом з відкритим вихідним кодом, моделі можуть не бути послідовними або точними, оскільки вони надходять з різних місць.
- Але Hugging Face буде LLM з відкритим вихідним кодом, який ми будемо використовувати для вибору моделі для нашого робота.

Flake8 - це інструмент для лінтування коду Python, який допомагає дотримуватися стандартів кодування та виявляти потенційні помилки або стилістичні проблеми. Він поєднує в собі декілька окремих інструментів для лінкування коду Python, включаючи PyFlakes, pycodestyle (раніше відомий як pep8) та перевірку складності сценаріїв McCabe Неда Батчелдера. Він широко використовується в робочих процесах розробки та інтегрується в редактори коду

або конвеєри CI/CD, щоб забезпечити відповідність коду Python найкращим практикам.

Flake8 має такі можливості:

- Максимальна довжина рядка: цей параметр дозволяє вказати максимальну кількість символів, дозволену в рядку у вашому коді.
- Список ігнорованих кодів помилок: за допомогою цієї опції користувач може вказати через кому список кодів помилок, які програма має ігнорувати.
- Вибір плагінів: Flake8 дозволяє встановлювати додаткові плагіни, які забезпечують користувацькі перевірки або підтримку певних стандартів кодування.

Chainlit - це пакет Python, призначений для легкого створення LLM чат-програм і створення користувацького інтерфейсу, схожого на ChatGPT. LangChain легко інтегрується з Langflow.

- Деякі з основних речей, які пропонує нам Chainlit, є наступними:
- Швидка побудова. Він може бути швидко і легко інтегрований з уже існуючою кодовою базою, або ж ми можемо створити його з нуля за лічені хвилини.
- Можна написати свою логіку чат-бота і використовувати її скрізь.
- Персистентність даних. Збирає, відстежує та аналізує дані від користувачів.
- Візуалізація багатокрокових міркуваннь. Розуміє проміжний крок, який одразу дає результат.
- Режим підказки, щоб зрозуміти, де щось пішло не так, і покращити.

Chainlit, як бібліотека з відкритим вихідним кодом, має зростаючу спільноту розробників і користувачів, які роблять активний внесок у її розвиток. Розробники визнали переваги використання Chainlit для швидкого та легкого створення чат-ботів на основі LLM.

Якщо порівнювати Chainlit з іншими подібними бібліотеками, то її переваги стають очевидними. Вона виділяється своєю інтеграцією, як і LangChain, розширюючи свої можливості за допомогою чат-ботів, а головне - простотою роботи з нею.

LangChain - це фреймворк для розробки додатків на основі великих мовних моделей (LLM).

LangChain спрощує життєвий цикл LLM всього за 3 простими категоріями:

- Розробка: Ми використовуємо будівельні блоки та компоненти, надані LangChain, для створення нашого додатку.
- Виробнича реалізація: Для постійного вдосконалення і розгортання нашого проекту ми будемо використовувати LangSmith для перевірки, моніторингу та оцінки ланцюжків, наданих LangChain.
- Розгортання: Ми використовуємо LangServe від LangChain для перетворення будь-якого ланцюжка в API.

Причиною вибору LangChain над іншими альтернативами з відкритим вихідним кодом є його унікальна мова вираження.

Технологій для побудови чат ботів (Рисунок 2.1).

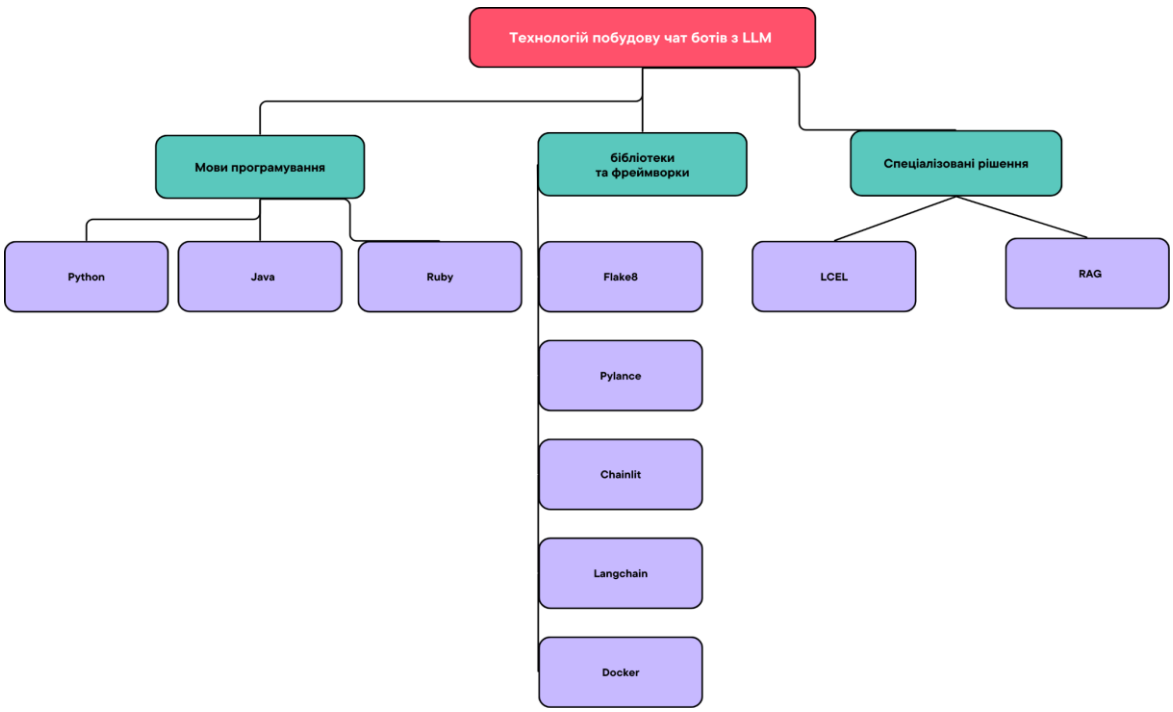


Рисунок 2.1 - Технологій для побудови чат-ботів.

2.3 Спеціалізовані мови програмування те open-source платформи для побудови чет-ботів з використанням великих мовних моделей

Мова виразів LangChain (LCEL) слугує основою для декількох компонентів LangChain і дозволяє створювати ланцюжки в декларативний спосіб. LCEL була навмисно розроблена, щоб полегшити перехід від прототипів до виробництва, не вимагаючи жодних модифікацій коду. Ця можливість поширюється від найпростішого ланцюжка "prompt та LLM" до найскладніших ланцюжків.

Пояснимо, переваги LCEL:

- Високоякісна потокова підтримка. Коли будуємо свої ланцюжки за допомогою LCEL, то досягаємо найвищого часу до першого токена, який означає тривалість до генерації першого виводу. У деяких мережах блокчейн це передбачає пряму передачу токенів від модуля нижнього рівня (LLM) до парсеру потокового виводу. Це дозволяє отримувати розібрані дані в інкрементному режимі, відповідно до швидкості, з якою постачальник LLM генерує сирі токени.
- Асинхронна підтримка. Ланцюжок, побудований за допомогою LCEL, можна викликати як за допомогою синхронного API (наприклад, у блокноті Jupyter під час створення прототипу), так і за допомогою асинхронного API (наприклад, на сервері LangServe).
- Ефективне паралельне виконання. Якщо LCEL-ланцюжки містять етапи, які можуть виконуватися одночасно (наприклад, отримання документів з різних пошукових систем), ми автоматично виконуємо їх паралельно. Це стосується як синхронних, так і асинхронних інтерфейсів, що призводить до мінімально можливої затримки.
- Спроби та інші опції. Налаштування механізми повторних спроб і резервного копіювання для кожного компонента вашого LCEL-ланцюга. Цей метод є чудовим підходом для підвищення надійності ваших ланцюжків при роботі у великих масштабах.
- Отримання проміжних результатів. При роботі зі складними послідовностями часто дуже корисно мати можливість отримати результати проміжних етапів ще до того, як буде створено кінцевий

результат. Це може бути використано для інформування кінцевих користувачів про поточні процеси або для усунення несправностей у вашій системі. Проміжні результати можуть передаватися в потоковому режимі і доступні на всіх екземплярах LangServe.

- Вхідні та вихідні структури. Схеми вводу та виводу надають схеми Pydantic та JSONSchema для кожного ланцюжка LCEL, які виводяться зі структури ланцюжка. Ця функція використовується для перевірки входів і виходів і є важливим компонентом LangServe.

Окрім своєї унікальної мови вираження, LangChain надає нам дружню екосистему, якою ми можемо користуватися. Існує три основні екосистеми:

- LangSmith. LangSmith допомагає вам відстежувати та оцінювати ваші додатки з мовними моделями та інтелектуальними агентами, щоб допомогти вам перейти від прототипу до виробництва.
- LangGraph. LangGraph - це бібліотека для побудови багатоакторних додатків з керуванням станом за допомогою LLM. Натхненна Pregel та Apache Beam, LangGraph дозволяє координувати та контролювати декілька ланцюжків (або акторів) через циклічні обчислювальні кроки за допомогою звичайних функцій Python (або JS). Загальнодоступний інтерфейс натхненний NetworkX.
- LangServe. LangServe допомагає розробникам розгортати виконувані модулі та ланцюжки LangChain як REST API. Ця бібліотека інтегрована з FastAPI і використовує pydantic для перевірки даних. Крім того, вона надає клієнт, який можна використовувати для виклику виконуваних модулів, розгорнутих на сервері. JavaScript-клієнт доступний у LangChain.js.

LangChain стає в нагоді з технікою, відомою як Retrieval Augmented Generation, або RAG.

RAG - це методика поглиблення знань LLM за допомогою додаткових даних.

Хоча їхня база знань обмежена публічними даними до певного моменту часу, які слугували їм навчальною базою, LLM здатні міркувати про більшість предметів. Для створення додатків ШІ, які можуть міркувати над приватними даними або даними, доданими після дати відсікання моделі, ви повинні додати точну інформацію, необхідну моделі для розширення її знань. Доповнена генерація на основі пошуку (RAG) - це процес отримання відповідних даних і вставки їх у запит моделі.

RAG складається з двох основних частин: генерації та індексації.

– Індексування: спосіб отримання інформації з джерела та її впорядкування.
– Пошук і генерація: це і є справжній ланцюжок RAG. Він приймає питання користувача під час виконання, отримує відповідні дані з індексу, а потім надсилає їх до моделі.

Це найпоширеніший повний шлях від сирої інформації до відповіді:

Індексація

- Завантаження: Спочатку ми повинні імпортувати наші дані. Для цього використовуються DocumentLoaders.
- Розділювачі тексту розбивають великі документи на менші сегменти. Ця функція корисна як для індексування даних, так і для введення даних у модель. Великі фрагменти даних складніше шукати, і вони не вміщуються в обмеженому контекстному вікні моделі.
- Сховище: Нам потрібне місце для зберігання та організації наших підрозділів, щоб у майбутньому їх можна було легко знайти. Зазвичай для цього використовується модель VectorStore та Embeddings.

Структура компоненту Індексування (Рисунок 2.2).

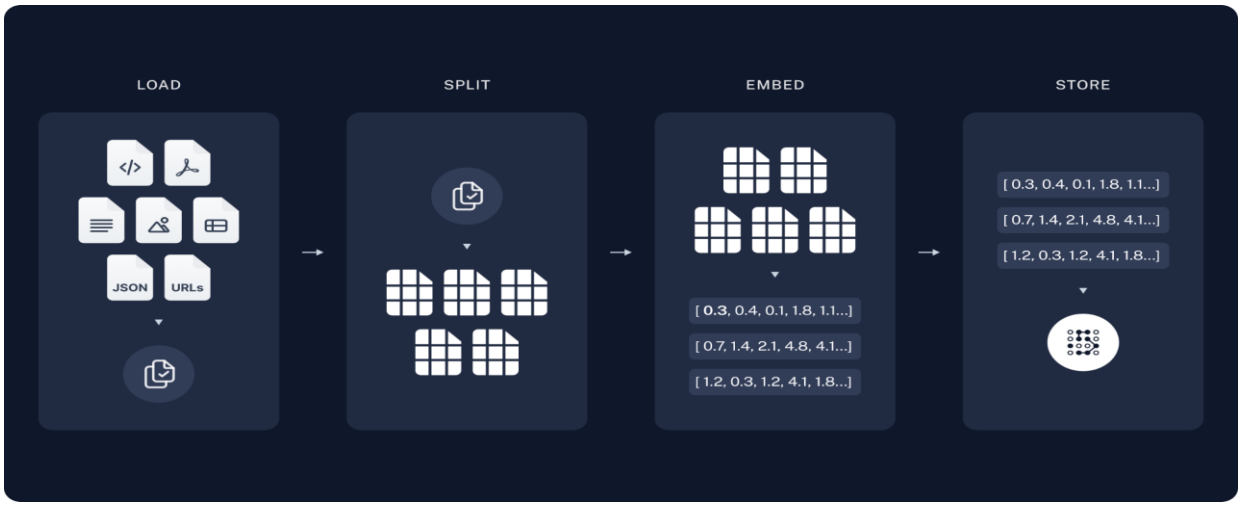


Рисунок 2.2 - Структура методу індексування.

Пошук інформації та генерація тексту:

– Отримати: Відповідні розбиття отримуються зі сховища за допомогою програми Retriever на основі даних, введених користувачем.
– Чат-модель, також відома як мовна модель, генерує відповідь, використовуючи підказку, яка складається із запиту та відповідних даних, які були отримані.

Метод пошуку інформації та генерації тексту (Рисунок 2.3).

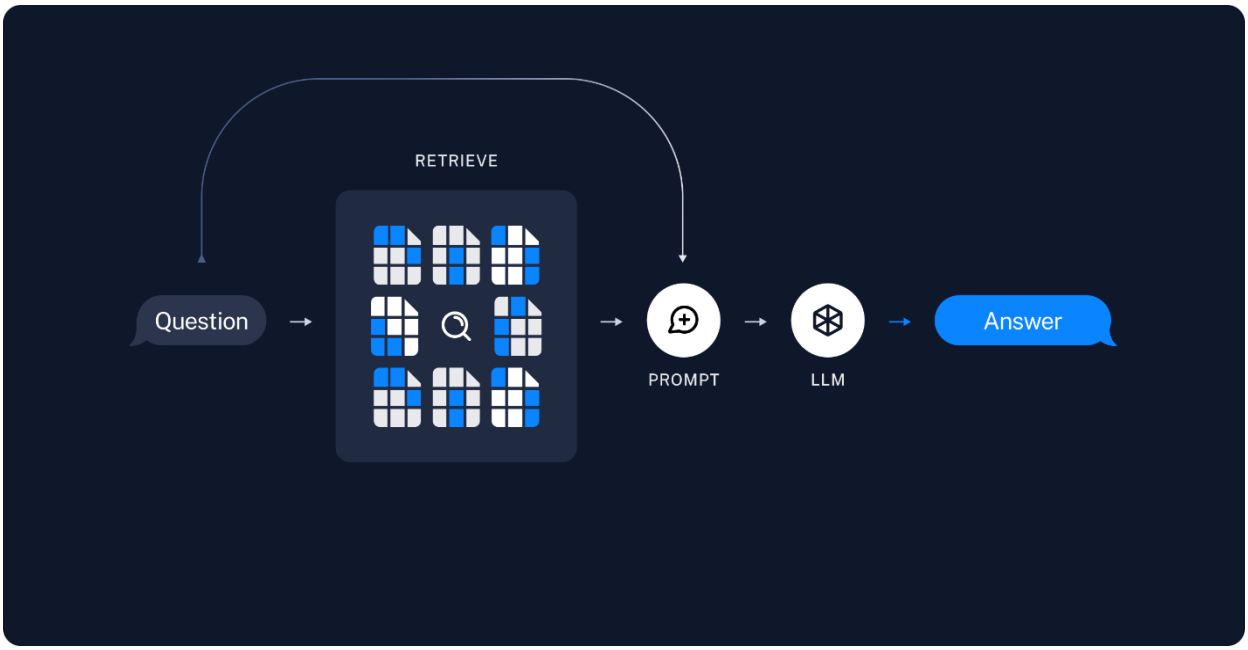


Рисунок 2.3 - Метод пошуку інформації та генерації тексту.

Завдяки цим можливостям і перевагам використання LangChain, я вважаю, що це буде найбезпечніший варіант для створення нашої системи в наступному розділі.

2.4 Великі мовні моделі для побудування чат-ботів

Зараз існує велика кількість великих мовних моделей, які дозволяють будувати чат-боти. Розглянемо найбільш відомі з них.

Llama - це мовна модель від компанія OpenAI, яка призначена для розуміння та створення текстів, подібних до людського письма. Це популярний вибір для створення складних чат-ботів завдяки своїй здатності створювати цікаві та контекстно- відповідні відповіді.

Це допоможе навчити чат-бота давати більш природні відповіді, створюючи більш ефективний користувацький досвід. Він також має великий словниковий запас і розуміння граматики, завдяки складним алгоритмам ми можемо розширити можливості чат-бота і зробити його більш релевантним до поточного контексту.

Microsoft Research випустила Orca LLM, доопрацьовану версію Llama, яка працює так само добре або навіть краще, ніж моделі, що містять у 10 разів більше параметрів. Для досягнення такої високої продуктивності Orca використовує синтетичний навчальний набір даних і нову техніку під назвою швидке стирання.

Модель Orca навчається за схемою "вчитель-учень", де більший і потужніший LLM виступає в ролі вчителя і учня, з метою покращення результатів роботи учня, щоб його можна було порівняти з іншими більшими моделями. Методика навчання Microsoft навчає меншу модель множинним методам міркувань, а також тому, як вибрати найефективніший метод для конкретного завдання. Для цього вчитель отримує складні підказки, які викликають певну поведінку міркувань. Однак у схемі під назвою Prompt Erasure учень отримує лише вимоги до завдання та бажану відповідь, а не підказку вчителя.

2.5 Вибір технологій для побудови чат ботів з великими мовними моделями

У наступному розділі я буду використовувати технології, про які я розповів у попередньому розділі, щоб об'єднати все разом і дати краще розуміння технологій і середовищ, в яких я створюю свого чат-бота.

Питання полягає в тому, які завдання виконуватиме мій чат-бот, як я можу з ним взаємодіяти, де він буде розгорнутий і звідки я візьму ресурси, тому в цьому розділі ми відповімо на всі ці питання.

Для побудови треба обрати LLM, якаа працює з певними наборами даних. Мною було вікористано дві найбільш популярні моделі - LLama та Orca. Ці моделі є безкоштовними.

Для програмування була обрана мова програмування Python та бібліотеки Flake8 для формуваання темплейтів та Pelance для більш швидкого написання коду.

Для створення UX рішень було використано open source платформу Chainlit.

Для контейнерування була використана платформа Docker, про яку буде написано в наступномуу розділі.

3 РОЗРОБКА ЧАТ-БОТУ З ВИКОРИСТАННЯМ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ LLAMA ТА ORCA.

3.1 Розгортання середовища розробки та встановлення потрібних додатків

Я буду використовувати технології, про які розповідав у попередній главі, щоб об'єднати все разом і дати краще розуміння технологій і середовищ, в яких я створюю свого чат-бота.

У цьому розділі розглянемо всі ці питання, зокрема, які завдання виконуватиме мій чат-бот, як з ним взаємодіяти, де його розгортатимуть, і звідки візьмуться ресурси.

Для налаштування свого середовища я буду використовувати два різних середовища: локальне та хмарне. Локальним середовищем буде Visual Studio Code. Вибір цих середовищ робить мою роботу з кодом набагато зручнішою, і я можу дуже легко візуалізувати свій код.

Оскільки мій комп'ютер і система англомовні, я буду писати майбутні скріншоти і кроки переважно англійською мовою.

Отже, почнемо з того, як встановити та створити папку для нашого чат- бота локально:

- Встановіть редактор коду VScode з офіційного сайту "https://code.visualstudio.com.".
- Створіть папку, де буде ваш код і оточення, і назвіть її "MyChatBot".
- Щоб відкрити папку MyChatBot у VScode, спочатку відкриваємо VScode, а потім вибераємо Файл > Відкрити папку > Робочий стіл > MyChatBot (у моєму випадку папка знаходиться на робочому столі).
- Переходимо до папки Vscode Extensions, щоб завантажити наступні файли:
    - Python - це наша мова програмування.
    - Ми використовуємо Docker для створення та контролю наших контейнерів.

- Ми використовуємо WSL для запуску наших командних рядків Python і додатків на Windows.
- З Devcontainer ми можемо використовувати наші Docker-контейнери як повнофункціональне середовище розробки.
- Ви можете знайти всі пакунки для віддаленої розробки за допомогою пошуку "@recommended:remotes".

Тепер, коли завантажили залежності та наші вимоги, треба встановити devcontainers та системні залежності.

Ми можемо почати з цих кроків:

- Створюємо файл "requirements.txt" у папці "MyChatBot".
- Усередині файлу "requirements.txt" встановлюємо багато розширень, тому я надам цей файл у розділі коду.
- Потім створюємо папку всередині "MyChatBot" під назвою "devcontainer".
- У новій папці створюємо файл 'devcontainer.json'.

Наразі ми встановили всі пакунки та розширення. Тепер нам потрібно перейти до наступних кроків, щоб запустити і зібрати наш проект. Оскільки ми встановили Dev Container, ми можемо зібрати наш проект локально.

- Переходимо до командної палітри, натиснувши "F1.".
- Шукаємо опцію "Dev Container: Додати файли конфігурації контейнера Dev".
- Далі вибираємо опцію "Додати конфігурацію до даних користувача". Це додасть всі конфігурації нашої локальної машини.
- Ми обрали наш шаблон "Python 3" та його версію за замовчуванням.
- Ми зможемо завантажити наступні функції:
  - Python Devcontainers.
  - Vscode CLI devcontainers-contrib.
  - Black (через Pipx)
  - Flake8 (через Pipx).

– Після вибору функцій натискаємо "ОК".
– Вона буде запитати, чи залишити налаштування за замовчуванням для обраної нами конфігурації, "опція конфігурації".
– Далі обираємо перший варіант для нашого плагіна flake8.
– Для нашого Python ми обрали останню версію.

Після виконання цих кроків з'явиться ".devcontainer", і vscode запропонує повторно відкрити контейнер. Однак, оскільки ми вже додали теку вручну, нам не потрібно цього робити. Замість цього ми скопіюємо наданий код, який представляє нашу файлову конфігурацію, і перейдемо до наступних кроків:

– Вставте конфігурацію коду в наш вручну створений файл "devcontainer.json".
– Додайте розширення та командні рядки до нашого коду.
– Файл вимагає встановлення наступних розширень:
    - "dbaeumer.vscode-eslint",
    - "GitHub.codespaces",
    - "GitHub.github-vscode-theme",
    - "GitHub.vscode-pull-request-github",
    - "ms-python.black-formatter",
    - "ms-python.debugpy",
    - "ms-python.flake8",
    - "ms-python.python",
    - "ms-python.vscode-pylance"
- Після цього у конфігурації файлу ми розкоментуємо рядок коду після команди, щоб при перезбудові контейнера він запустив файл і встановив усі наші пакунки для проекту.
- Натисніть "F1", щоб відкрити палітру команд.
- Знайдіть опцію "Dev Containers: Відновлення та повторне відкриття в контейнерах".
- Ми обираємо власну папку проекту, а не ту, що надається Vscode, і створюємо наш контейнер.

Тепер ми бачимо, що наша папка працює на локальному контейнері з обраним нами шаблоном Python3, "Dev Container Python 3".

3.2 Використання технології Docker для розгортання додатків

Щоб пришвидшити запуск програмного забезпечення, ми хочемо відокремити нашу систему від апаратного забезпечення. Використання переваг Docker для доставки, тестування та випуску коду значно скоротить час від написання коду до його запуску у виробництво.

Docker - це інструмент з відкритим вихідним кодом, який дозволяє запускати програму в контейнері, який є контейнерною системою. Цей поділ дає нам хороший захист для запуску багатьох контейнерів на одному хості одночасно.

Docker також надає інструменти та платформу для управління життєвим циклом ваших контейнерів:

- Він розробляє наш додаток і підтримує компоненти, що використовують контейнери.
- Контейнер стає одиницею для розповсюдження та тестування нашого додатку.
- Коли ми готові, Docker дозволяє нам розгорнути наш додаток у виробничому середовищі, як контейнер або організований сервіс.

Контейнери - це виконувані екземпляри образу, де ми можемо створювати, запускати, зупиняти, переміщувати або видаляти контейнер за допомогою Docker API або CLI. Я можу підключити контейнер до однієї або декількох мереж, додати до нього сховище або навіть створити новий образ на основі його поточного стану.

За замовчуванням, контейнер відносно добре ізольований від інших контейнерів та хост-машини. Ми можемо контролювати, наскільки ізольовано мережу, сховище або інші базові підсистеми контейнера від інших контейнерів або від хост-машини.

Контейнер визначається його зображенням, а також будь-якими параметрами конфігурації, які ми надаємо йому під час створення або запуску. Видалення контейнера стирає всі зміни стану, які не зберігаються у постійному сховищі.

Для цього ми повинні зрозуміти, що таке об'єкти зображень у Docker. Зображення в Docker - це шаблон тільки для читання з інструкціями для створення контейнера Docker. Часто зображення базується на іншому зображенні з деякими додатковими налаштуваннями.

Використання Docker в нашій системі нагадує наступний сценарій:

– Вважайте себе розробником; ви будете писати код локально і ділитися ним з колегами, використовуючи контейнери Docker.
– Ви можете перенести свій додаток у тестове середовище і запустити автоматизовані та ручні тести.
– Коли ви виявляєте помилки, ви можете виправити їх у середовищі розробки і перенести в тестове середовище для тестування та перевірки.
– Після того, як я завершив тестування, передати виправлення клієнту було так само просто, як перенести оновлений образ у виробниче середовище.

Архітектура Docker (Рисунок 3.1).

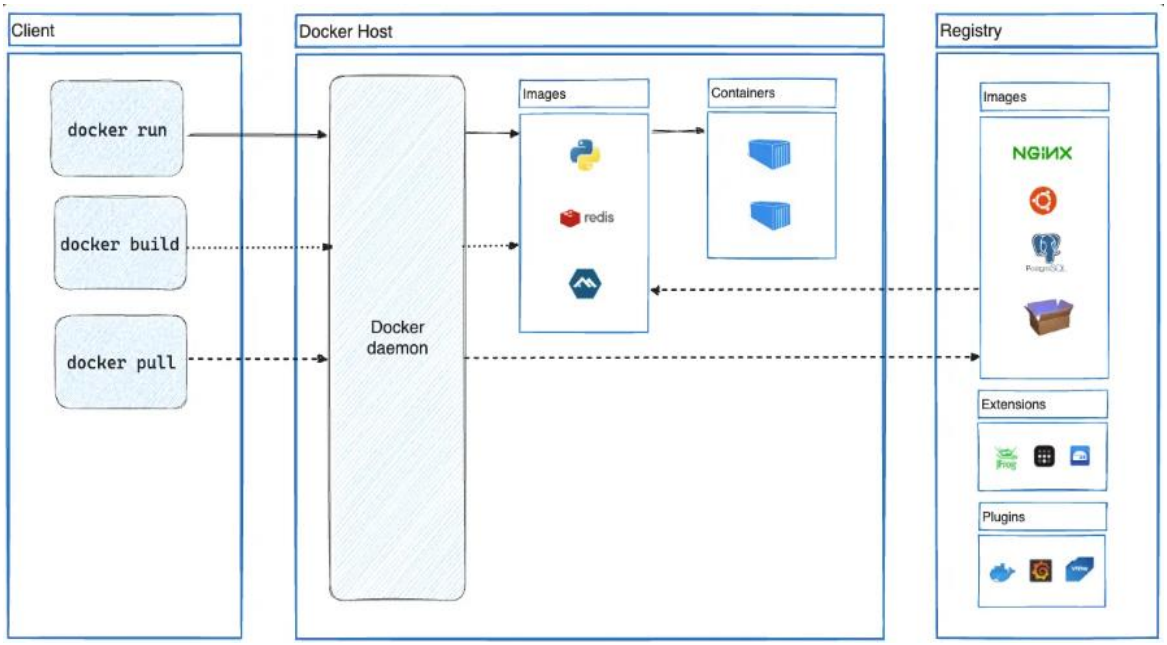


Рисунок 3.1 - Архітектура Docker

Вибір Docker дозволив мені працювати в стандартизованому середовищі, використовуючи локальні контейнери, які надають мої програми та сервіси, які мені потрібні.

3.3 Побудова великих мовних моделей

У цьому розділі ми розглянемо дві моделі LLM, а саме моделі Llama та Orca. У першій частині ми будемо використовувати модель Orca, а в другій частині - модель Llama.

Як ми вже згадували раніше, ми будемо використовувати наш Hugging Face з відкритим вихідним кодом для встановлення наших моделей LLM, і я знайшов там ресурс про те, як встановити нашу модель, яка нам підходить.

- Перейдіть на "Hugging Face", наше відкрите джерело, звідки ми беремо нашу картку моделі, а саме "https://huggingface.co/pankajmathur/orca_mini_3b".
- Ми додали ще три бібліотеки до нашого файлу "requirements.txt": transformers, sentencepiece та accelerate.
- Відновіть і знову відкрийте контейнер, як ми це робили раніше.
- Усередині нашої папки ми створюємо файл на Python з назвою "orca.py".
- Тепер, як зазначено в наданому джерелі, ми завантажимо бібліотеку crtransformers Python за допомогою команди "pip install crtransformers".
- Усередину файлу імпортуйте пакунок torch.
- Імпортуйте LlamaForCausalLM, LlamaTokenizer з нашої бібліотеки трансформаторів.
- Потім напишіть наш шлях до моделі та мову моделі.
- Ми написали функцію генерації тексту для нашої моделі.
- Ми створюємо базовий логічний набір даних і зразок завдань, яка повинна виконувати модель.

Тепер, коли ми побудували нашу модель Llama2, ми використовуємо модель з нашої бібліотеки з відкритим вихідним кодом Hugging Face. Оскільки ми вже створили Orca, дуже легко створити Llama2, виконавши наведені нижче кроки:

- Ми відкрили документацію llama2 з Hugging Face: "https://huggingface.co/TheBloke/Llama-2-7B-Chat-GGUF".
- Створюємо новий файл під назвою "llama2.py".
- Встановлюємо пакет ctransformers, "pip install ctransformers".
- Додаємо шлях моделі.
- швидке формування для специфічного для llama2.
- Ми додаємо простий набір запитань, щоб допомогти нам вирішити, який тип завдань ми хочемо, щоб наш чат-бот виконував.

3.4 Використання Chainlit для створення інтерфейсу чат-бота

Спочатку ми побудуємо модель Llama2, а потім модель Orca. Всі ці кроки описані на їхньому офіційному сайті з документацією.

- У нашому проекті ми встановлюємо пакунок, просто написавши команду "pip install chainlit".
- Створіть нову папку з назвою "Chat_System.".
- Ми створили новий файл під назвою "Chainlit_Llama.py".
- Ми імпортуємо Chainlit за допомогою "import chainlit as cl" .
- Ми додали функцію "on_message".
- Ми додали функцію "при_запуску_чату".
- Додати функцію потокового передавання до "on_message".

Для нашої моделі Orca ми розробляємо користувацький інтерфейс Chainlit.

- У нашій папці "Chat_System" створіть файл з назвою Chainlit_Orca.py.
- Оскільки оригінальний розмір моделі дуже великий для Chainlit і мій комп'ютер не може з ним впоратися, ми знайдемо іншу стиснуту модель, яку можна знайти на "https://huggingface.co/juanjgit/orca_mini_3B-GGUF.".
- Ми імпортуємо Chainlit.
- Імпортуємо AutomodelForCasualLlm.

- Далі ми додаємо логіку підказок і формат моделі Orca.
- Ми додаємо ланцюжкові обробники "on_message".
- Ми додаємо ланцюжкові обробники "on_chat_start".
- Запустіть файл у терміналі: "chainlit run./Chat_System/Chainlit_Orca.py - w".

3.5 Використання Langchian для поєднання моделі

Тепер ми з'єднуємо наші моделі ланцюжком за допомогою LangChain:

- Створіть файл у папці "Chat_System" з назвою "Langchain.py".
- Встановіть пакунки, виконавши "pip install langchain langchain-community".
- Імпортувати Chainlit, BaseCallbackhandler, LLMchain, ConverstaionBufferMemory, CTransformers ,
- Ми створили обробник потоку класу.
- Завантажуйте наш ІЛМ.
- визначити шаблони підказок LM.
- створити ланцюжкові обробники: on_message , on_start_chat.
- Запустіть файл за допомогою команди Chainlit.

Виконавши ці кроки, ми побачимо, що Vscode відкриє нову вкладку нашого інтерфейсу Chainlit з langChain у нашому Chrome або іншій пошуковій системі, яку ви використовуєте.

4 ТЕСТУВАННЯ РОЗРОБЛЕНОЇ МОДЕЛІ

4.1 Огляд тестування програми

Я буду слідувати інструкціям, наведеним у попередньому розділі, і представлю результат у вигляді скріншоту в Додатку В для наочного представлення. Крім того, я додам код у Додаток А, щоб продемонструвати, чи дійсно інструкції дають бажаний результат.

Наразі я дійшов до того, що скопіював конфігураційний файл, наданий Vscode, і помістив його в свою папку json. Крім того, я додав розширення, як вказано в інструкції. Щоб полегшити розуміння, я позначу його як тест 1 і використаю той самий підхід для позначення інших етапів мого процесу тестування. У тесті 2 я додав три пакунки до файлу "requirements.txt" і реконструював контейнер.

Тест 2 передбачає використання файлу "orca.py" для оцінки функціональності моделі та її реакції на початкову підказку: "Яке місто є столицею України?". Правильна відповідь - "Київ".

Тест 3: Я запустив файл "llama.py" нашої моделі Llama2, щоб перевірити, як вона реагує на мої запити, і дослідити її можливості пам'яті. Зокрема, я перевірив, чи може вона зберігати контекст запиту і відповідну відповідь, як і в попередніх випадках.

- Я запитую: "Яке місто є столицею Індії?".
- Відповідь: "Столиця Індії - Нью-Делі".
- Я даю додаткову підказку, запитуючи: "Який варіант відповідає Німеччині?".
- Правильною відповіддю буде "Берлін".

Для тесту 4, я використовував команду Chainlit для запуску чат-бота на локальному хості. Я дотримувався інструкцій, наведених у посібнику, і звертався до документації по Chainlit. В результаті я успішно розробив користувацький інтерфейс (UX) для моделей Llama2 та Orca. Крім того, я перевірив, чи обидві моделі мають можливості пам'яті. Я поставив багато

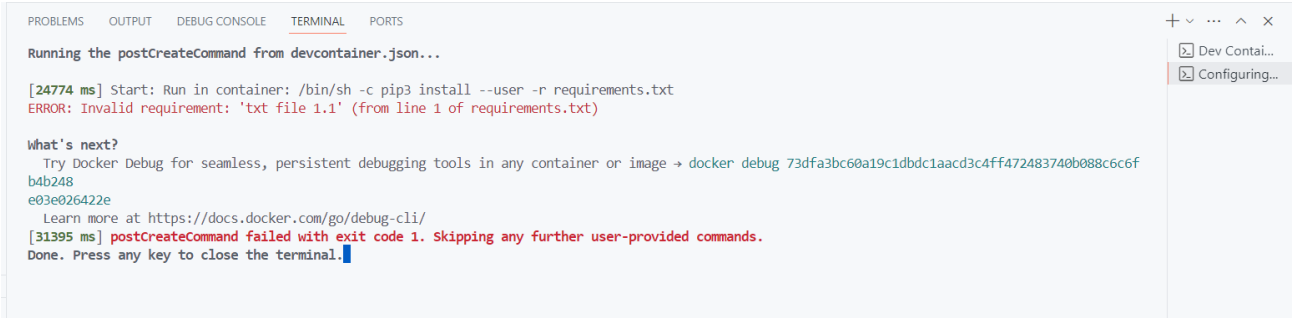
запитів до обох моделей, які охоплювали такі теми, як математика та столиці міст.

Під час фінального тесту я провів експеримент, щоб з'ясувати, чи зможу я ефективно пов'язати наші моделі за допомогою Chainlit, зокрема, з LangChain. Мета полягала в тому, щоб оцінити, чи зможу я легко переходити між моделями за допомогою простої підказки.

4.2 Покращення та виправлення багів

Коли ми будуємо проект з самого початку, очікується, що ми можемо зіткнутися з помилками та недоліками. Саме тому ми присвятили цю частину виключно вирішенню цієї проблеми.

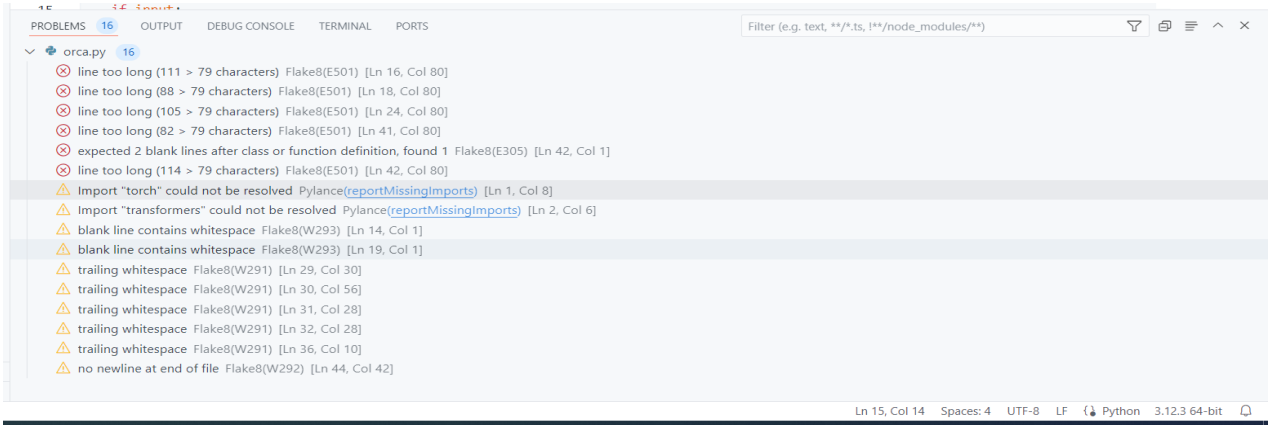
Під час виконання тесту 2 я побачив повідомлення про помилку (Рисунок 4.1).



4.1 - Повідомлення про помилку.

Це повідомлення про помилку означає, що команда postCreateCommand у моєму конфігураційному файлі не змогла виконатися через наявність рядка "txt file 1.1". Щоб вирішити цю проблему, я відредагував свій файл "requirements.txt" і видалив вищезгаданий рядок, після чого продовжив збірку контейнера.

Після виконання файлу у тесті 3 я зіткнувся з помилкою на своєму терміналі, (Рисунок 4.2).



4.2 - Помилка на терміналі

Щоб вирішити ці проблеми, мені потрібно було виконати наведені нижче процедури.

Створити файл “.flake8” і напишіть цей фрагмент коду:

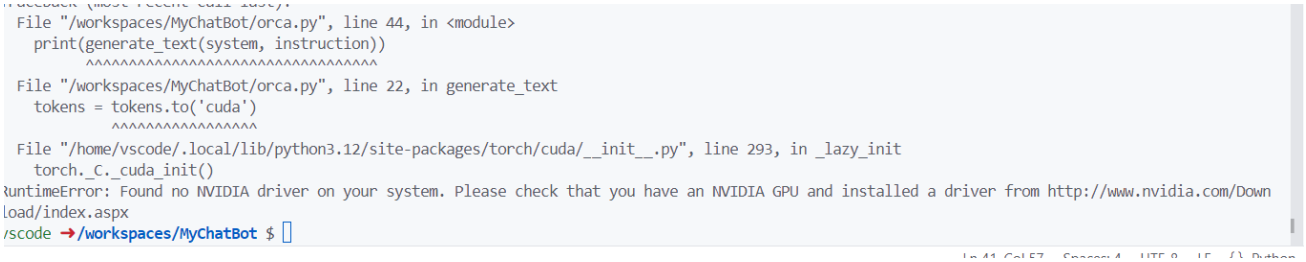
[flake8]

ignore = E501

ignore = E305

Тепер перед нами стоїть завдання вирішити проблеми з імпортом. Для цього запускаємо новий термінал і приступаємо до імпорту необхідних пакунків, виконавши команду "pip install torch transformers". Після запуску команди з'являється повідомлення про те, що пакунки вже встановлено, що є позитивною ознакою.

Після запуску файлу "orca.py" та отримання пакетів моделі, я побачив повідомлення про помилку, що вказує на відсутність драйвера NVIDIA. Це очікувано, оскільки модель вимагає апаратного забезпечення GPU (Рисунок 4.3).

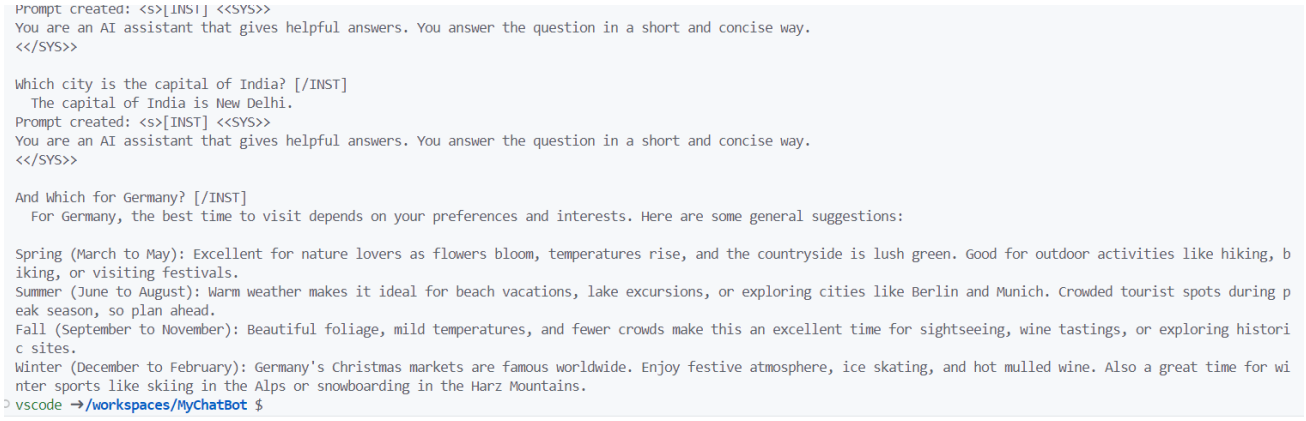


4.3 - Повідомлення про відсутність драйверу.

Для того, щоб вирішити цю проблему, необхідно перевірити наявність CUDA і виконати його на обладнанні з GPU. Крім того, важливо переконатися, що файл може працювати і на комп'ютерах, які не мають GPU. Для цього я розробив стислий фрагмент коду:

```
if torch.cuda.is_available():
    tokens = tokens.to('cuda')
    model.to('cuda')
```

Провівши тест на файлі "llama.py", ми отримали точну відповідь на наше первинне запитання про столицю Індії. Однак, коли ми ввели другий запит "А яка для Німеччини?", він дав чітку відповідь, як показано на (Рисунок 4.4).



4.4 - Відповідь на запит.

Для вдосконалення моделі необхідно створити систему зберігання або пам'яті, яка б зберігала контекст початкової підказки і застосовувала його до наступних підказок. Цього можна досягти, реалізувавши функцію історії розмови. Спочатку функція зберігає розмову і включає питання з першої підказки. Коли з'являється друге запитання, функція оновлює історію розмови відповідним чином. Наведений фрагмент коду - це функція, яка отримує частину історії чату для першої підказки.

```
# Initialize conversation history
conversation_history = ""

# First question
question = "Which city is the capital of India?"
prompt = get_prompt(question, conversation_history)
response = ""
for word in llm(prompt, stream=True):
    print(word, end="", flush=True)
    response += word
print()

# Update conversation history with the first question and response
conversation_history += f"{question}\n{response}\n"
```

Після цього ми повторюємо ту саму процедуру з другим запитом, після чого чат- бот повинен точно відповісти на наш запит.

ВИСНОВКИ

В першому ми дослідили можливості LLM? починаючи від передбачення слів і закінчуючи відеочатами зі штучним інтелектом. LLM продемонстрував свою здатність розробляти чат-боти для різних галузей, включаючи підтримку клієнтів та електронну комерцію, а також створити відому модель GPT.

Ми також дослідили різні типи агентів LLM, їхні основні компоненти та інструменти, які вони використовують для визначення своїх цілей у різних сценаріях.

Було досліджено застосування ChatGPT в різних сферах, включаючи цифровий маркетинг та контент-маркетинг. Ми також знаємо, що навчання моделі ChatGPT проходить у два етапи, а саме: етап попереднього навчання та етап точного налаштування.

У другому розділі було розглянуто технології, які використовуються для створення чат-ботів, що використовують LLM. Це комп'ютерні мови, такі як Python, Ruby та Java, а також інструменти та фреймворки для створення чат- ботів на основі LLM, такі як Flake8. Також були розглянуті унікальні комп'ютерні мови та інструменти з відкритим вихідним кодом, такі як Hugging Face і LangChain Expression Language (LCEL). Було обрані технології розробки чат-бота - Docker, а також LLM-моделі Orca і Llama.

У третьому та четвертому розділах ми використали дві моделі, Orca та Llama, щоб створити власного чат-бота. Було розроблено інтерактивний інтерфейс за допомогою Chainlit і вдосконалена модель за допомогою LangChain, створивши покрокове керівництво для створення цього простого чат-бота, який може відповідати на запитання, видавати результат і код програми.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

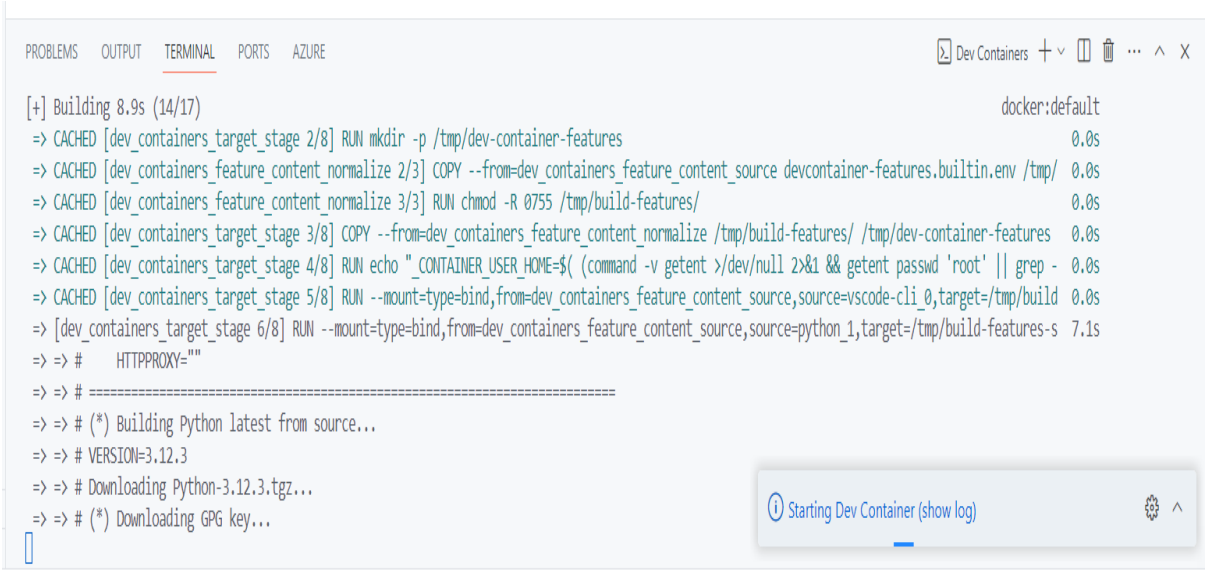
1. Valentina Alto, Building LLM Powered Application, Packt Publishing Ltd, 2024, 342 сторінок.
2. Caroline Fell Kurban, Muhammed Şahin, The Impact of ChatGPT on Higher Education: Exploring the AI Revolution, Emerald Group Publishing, 2024, 232 сторінок.
3. Prem Timsina, Building Transformer Models with PyTorch 2.0: NLP, computer vision, and speech processing with PyTorch and Hugging Face, BPB Publications, 2024, 310 сторінок.
4. Cyberpunk University, Python: The No-Nonsense Guide: Learn Python Programming Within 12 Hours, CreateSpace Independent Publishing Platform, 2017, 126 сторінок.
5. Ben Auffarth, Generative AI with LangChain: Build large language model (LLM) apps with Python, ChatGPT, and other LLMs, Packt Publishing Ltd, 2023, 360 сторінок.
6. Документація платформу Docker [Посилання на ресурс: https://www.docker.com/], (Дата звернення 11.12.2023).
7. Документація фреймворк LangChain [Посилання на ресурс: https://python.langchain.com/v0.2/docs/introduction/],(Дата звернення 08.0.2023).
8. Документація бібліотеку Chainlit [Посилання на ресурс: https://docs.chainlit.io/get-started/overview], (Дата звернення 21.07.2023).
9. Документація Hugging Face Transformers [Посилання на ресурс: https://huggingface.co/docs/transformers/index], (Дата звернення 03.11.2022).
10. Документація LLM model Orca [Посилання на ресурс: https://huggingface.co/Open-Orca/OpenOrca-Preview1-13B], (Дата звернення 01.06.2022).

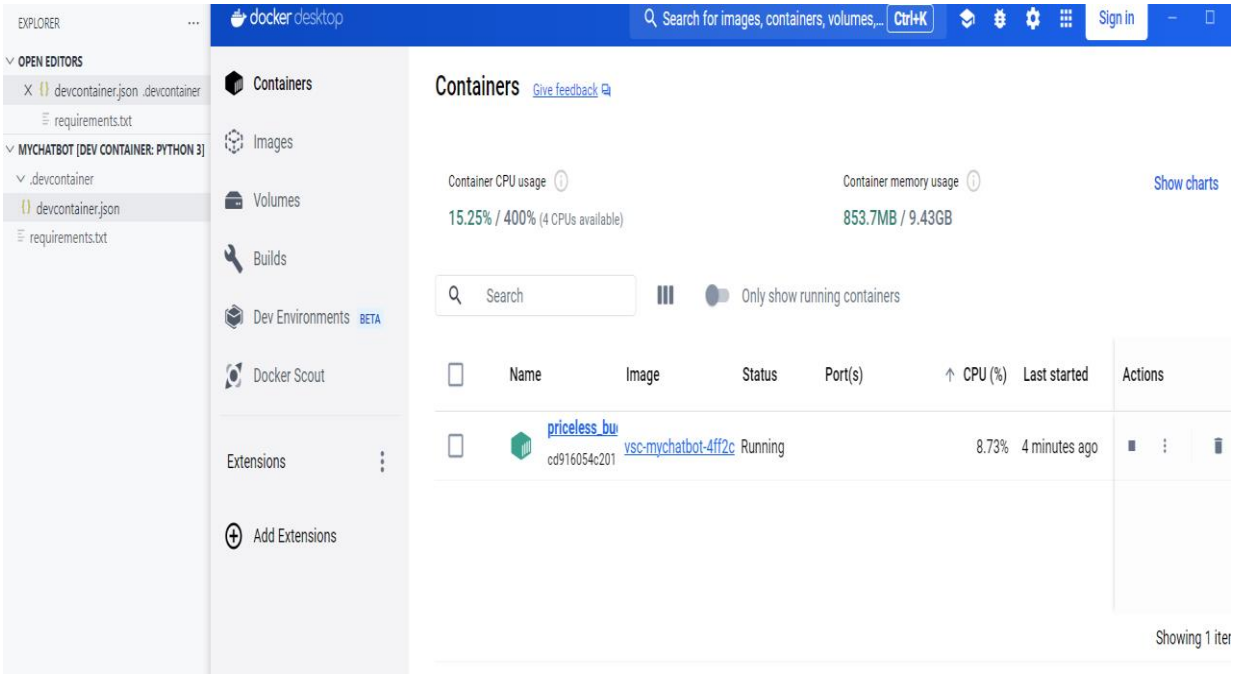
11. Документація LLM model Llama2 [Посилання на ресурс: https://huggingface.co/docs/transformers/main/en/model_doc/llama2], (Дата звернення 10.09.2022).
12. Документація бібліотека Flake8 [Посилання на ресурс: https://flake8.pycqa.org/en/latest/], (Дата звернення 08.11.2021).

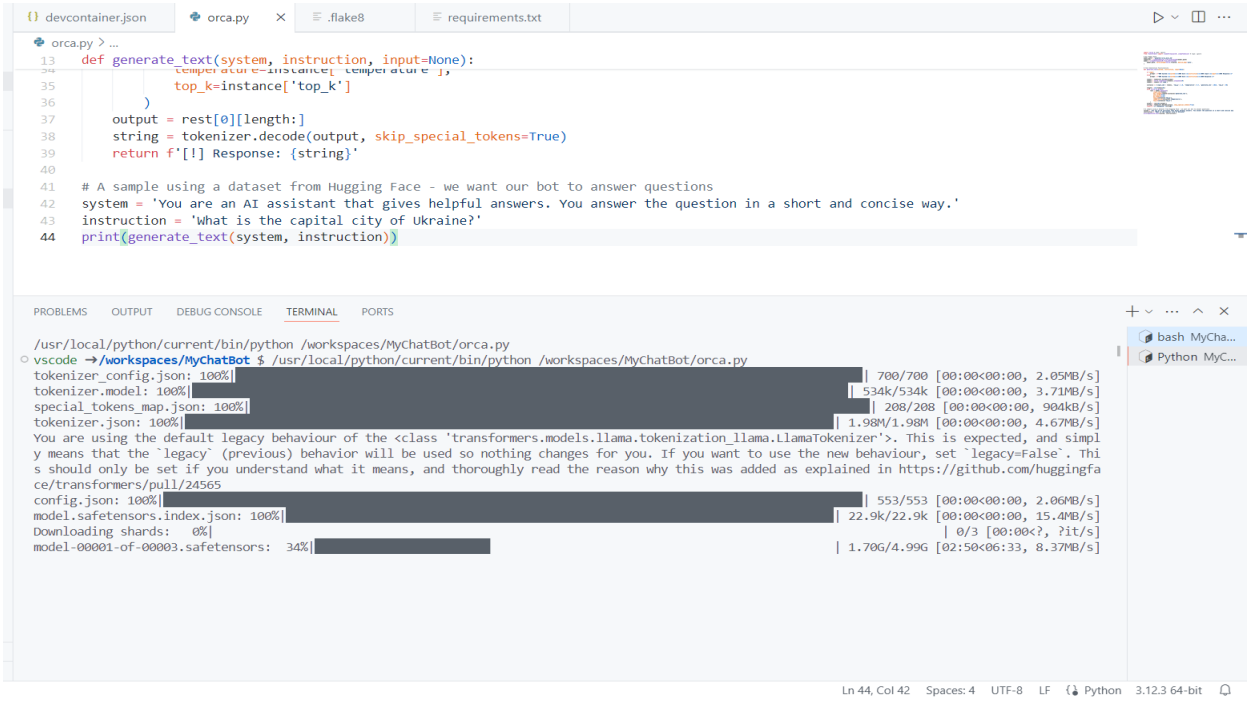
ДОДАТОК

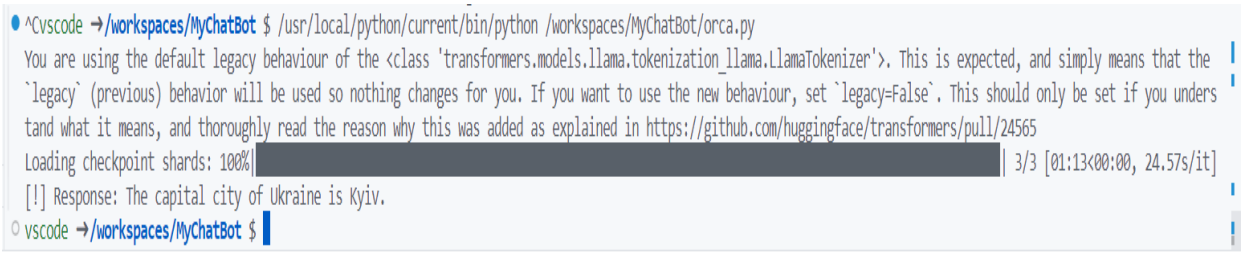
Додаток А

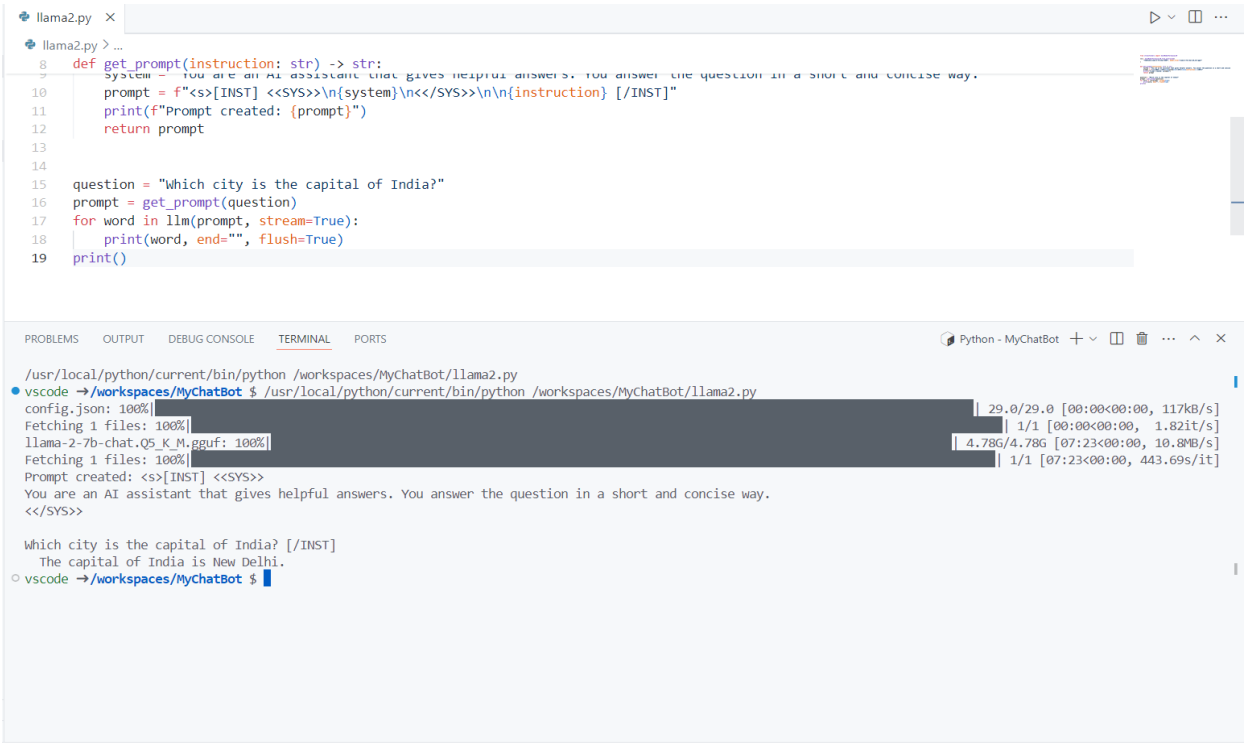
« Скріншоти і результати тестів роботи»



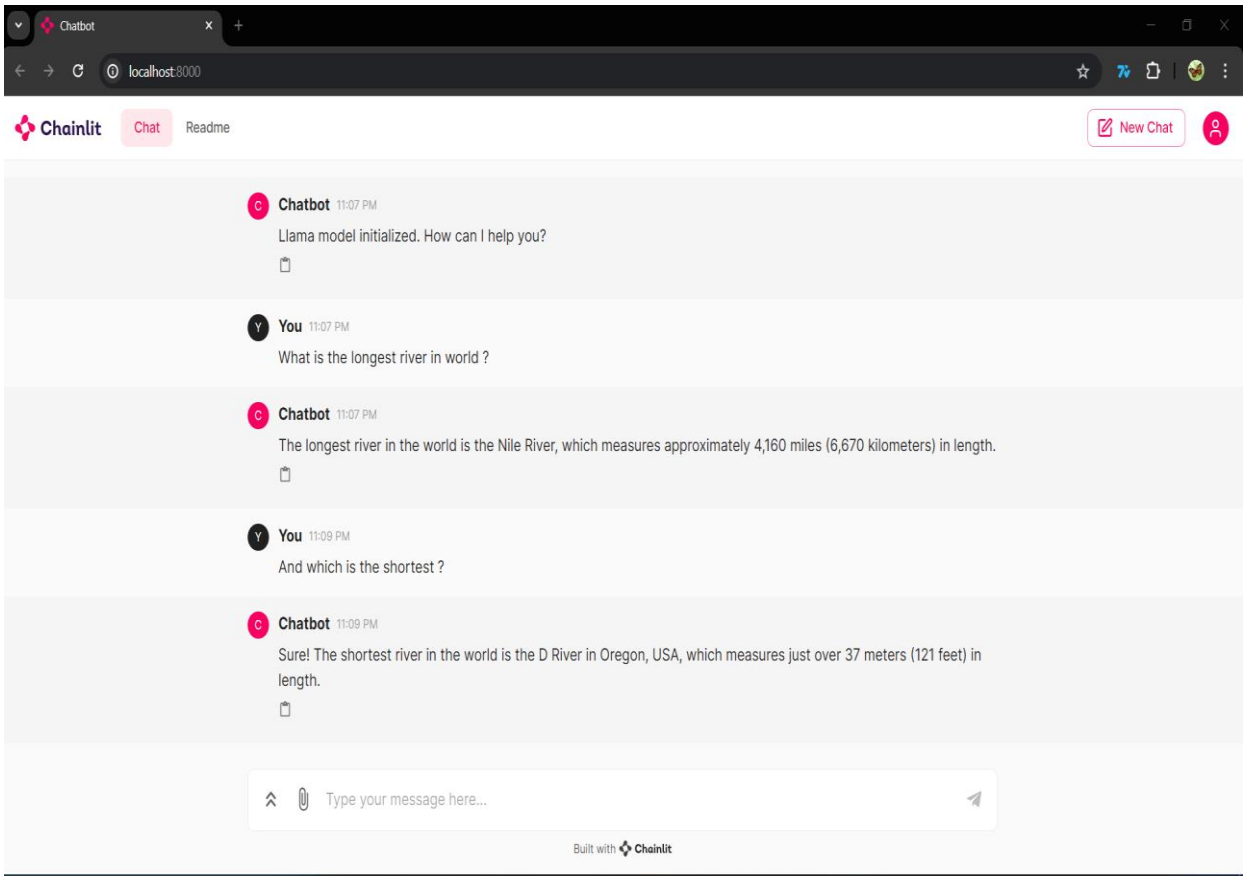


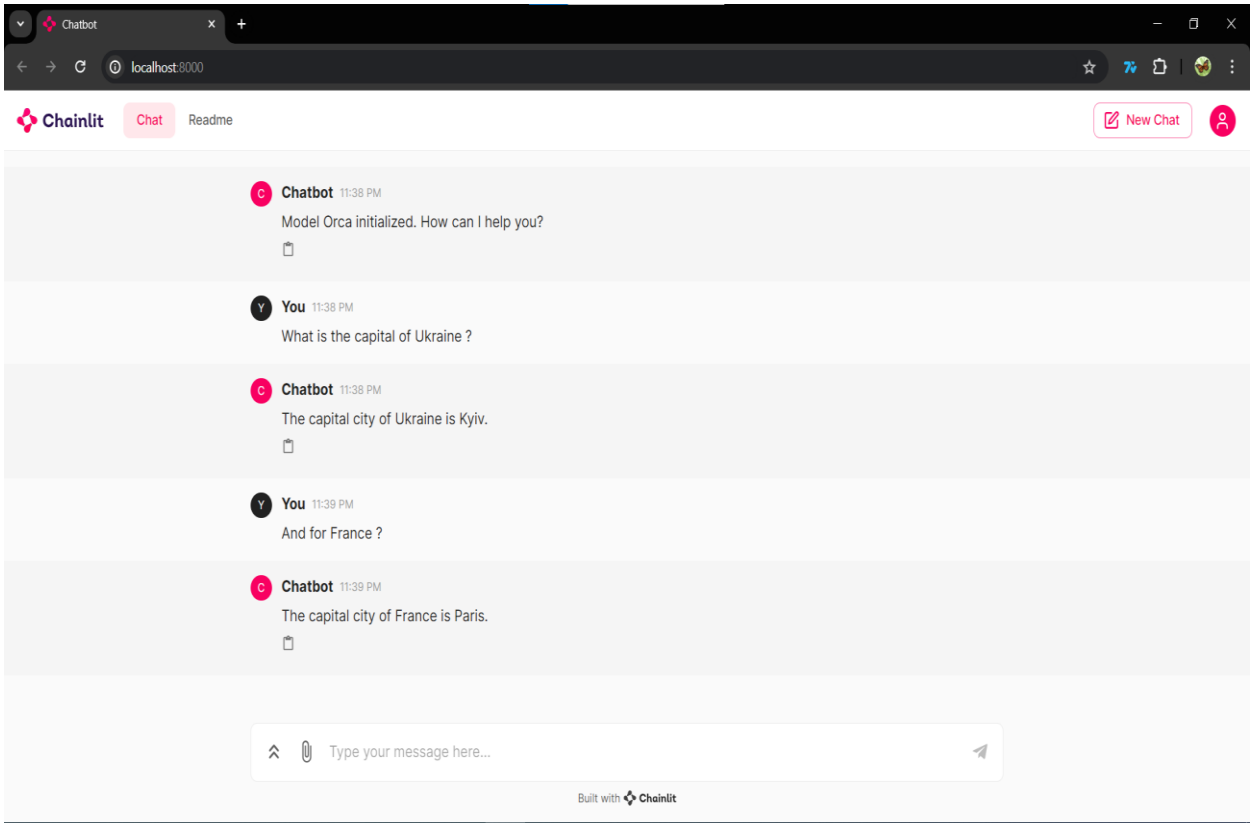

devcontainer.json
orca.py
.flake8
requirements.txt
orca.py > ...
def generate_text(system, instruction, input=None):
top_k=instance['top_k']
)
output = rest[0][length:]
string = tokenizer.decode(output, skip_special_tokens=True)
return f'[!] Response: {string}'
# A sample using a dataset from Hugging Face - we want our bot to answer questions
system = 'You are an AI assistant that gives helpful answers. You answer the question in a short and concise way.'
instruction = 'What is the capital city of Ukraine?'
print(generate_text(system, instruction))
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
/usr/local/python/current/bin/python /workspaces/MyChatBot/orca.py
vscode ➜/workspaces/MyChatBot $ /usr/local/python/current/bin/python /workspaces/MyChatBot/orca.py
tokenizer_config.json: 100%| 700/700 [00:00<00:00, 2.05MB/s]
tokenizer.model: 100%| 534k/534k [00:00<00:00, 3.71MB/s]
special_tokens_map.json: 100%| 208/208 [00:00<00:00, 904kB/s]
tokenizer.json: 100%| 1.98M/1.98M [00:00<00:00, 4.67MB/s]
You are using the default legacy behaviour of the <class 'transformers.models.llama.tokenization_llama.LlamaTokenizer'>. This is expected, and simply means that the `legacy` (previous) behavior will be used so nothing changes for you. If you want to use the new behaviour, set `legacy=False`. This should only be set if you understand what it means, and thoroughly read the reason why this was added as explained in https://github.com/huggingface/transformers/pull/24565
config.json: 100%| 553/553 [00:00<00:00, 2.06MB/s]
model.safetensors.index.json: 100%| 22.9k/22.9k [00:00<00:00, 15.4MB/s]
Downloading shards: 0%| 0/3 [00:00<?, ?it/s]
model-00001-of-00003.safetensors: 34%| 1.70G/4.99G [02:50<06:33, 8.37MB/s]
bash MyCha...
Python MyC...
Ln 44, Col 42
Spaces: 4
UTF-8
LF
Python
3.12.3 64-bit

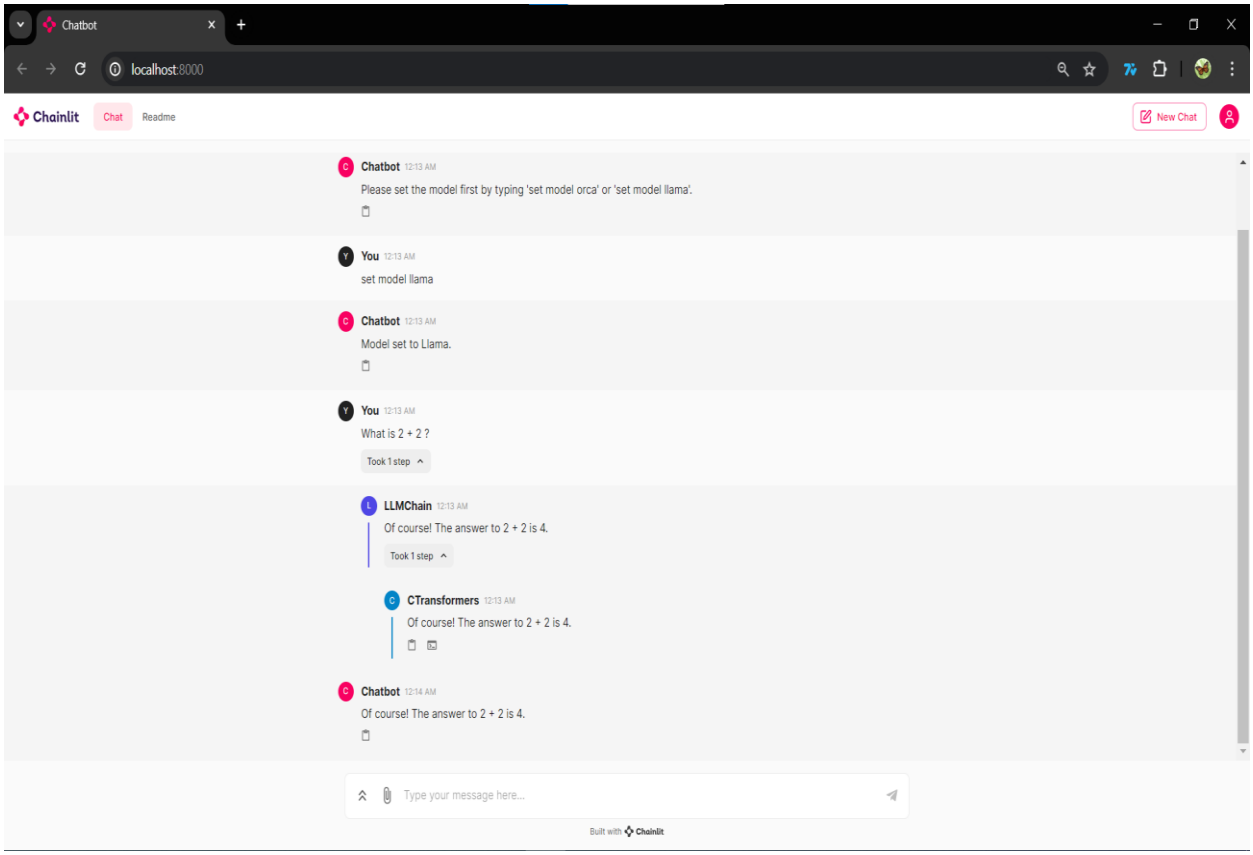

^Cvscode ➜/workspaces/MyChatBot $ /usr/local/python/current/bin/python /workspaces/MyChatBot/orca.py
You are using the default legacy behaviour of the <class 'transformers.models.llama.tokenization_llama.LlamaTokenizer'>. This is expected, and simply means that the `legacy` (previous) behavior will be used so nothing changes for you. If you want to use the new behaviour, set `legacy=False`. This should only be set if you understand what it means, and thoroughly read the reason why this was added as explained in https://github.com/huggingface/transformers/pull/24565
Loading checkpoint shards: 100%| 3/3 [01:13<00:00, 24.57s/it]
[!] Response: The capital city of Ukraine is Kyiv.
vscode ➜/workspaces/MyChatBot $

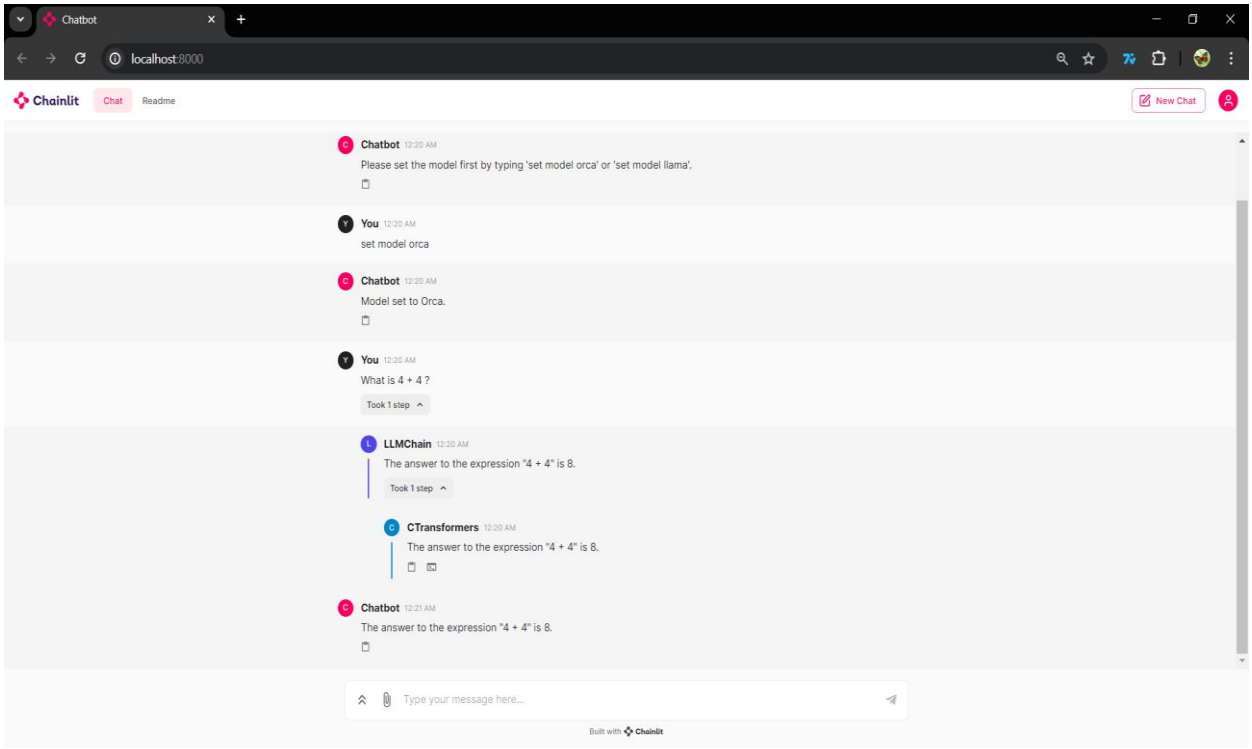

llama2.py
llama2.py > ...
def get_prompt(instruction: str) -> str:
prompt = f"<s>[INST] <<SYS>>\n{system}\n<</SYS>>\n\n{instruction} [/INST]"
print(f"Prompt created: {prompt}")
return prompt
question = "Which city is the capital of India?"
prompt = get_prompt(question)
for word in llm(prompt, stream=True):
print(word, end="", flush=True)
print()
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python - MyChatBot
/usr/local/python/current/bin/python /workspaces/MyChatBot/llama2.py
vscode ➜/workspaces/MyChatBot $ /usr/local/python/current/bin/python /workspaces/MyChatBot/llama2.py
config.json: 100%| 29.0/29.0 [00:00<00:00, 117kB/s]
Fetching 1 files: 100%| 1/1 [00:00<00:00, 1.82it/s]
llama-2-7b-chat.Q5_K_M.gguf: 100%| 4.78G/4.78G [07:23<00:00, 10.8MB/s]
Fetching 1 files: 100%| 1/1 [07:23<00:00, 443.69s/it]
Prompt created: <s>[INST] <<SYS>>
You are an AI assistant that gives helpful answers. You answer the question in a short and concise way.
<</SYS>>
Which city is the capital of India? [/INST]
The capital of India is New Delhi.
vscode ➜/workspaces/MyChatBot $


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Python - MyChatBot
Winter (December to February): Germany's Christmas markets are famous worldwide. Enjoy festive atmosphere, ice skating, and hot mulled wine. Also a great time for wi
nter sports like skiing in the Alps or snowboarding in the Harz Mountains.
vscode ➜/workspaces/MyChatBot $ /usr/local/python/current/bin/python /workspaces/MyChatBot/llama2.py
Fetching 1 files: 100%| 1/1 [00:00<00:00, 5489.93it/s]
Fetching 1 files: 100%| 1/1 [00:00<00:00, 7752.87it/s]
Prompt created: <s>[INST] <<SYS>>
You are an AI assistant that gives helpful answers. You answer the question in a short and concise way.
<</SYS>>
Which city is the capital of India? [/INST]
The capital of India is New Delhi.
Prompt created: <s>[INST] <<SYS>>
You are an AI assistant that gives helpful answers. You answer the question in a short and concise way.
<</SYS>>
Which city is the capital of India?
The capital of India is New Delhi.
And which for Germany? [/INST]
Great, thank you for asking! The capital of Germany is Berlin.
vscode ➜/workspaces/MyChatBot $
Ln 11, Col 39
Spaces: 4
UTF-8
LF
Python
3.12.3 64-bit


Chatbot
localhost:8000
Chainlit
Chat
Readme
New Chat
Chatbot 11:07 PM
Llama model initialized. How can I help you?
You 11:07 PM
What is the longest river in world ?
Chatbot 11:07 PM
The longest river in the world is the Nile River, which measures approximately 4,160 miles (6,670 kilometers) in length.
You 11:09 PM
And which is the shortest ?
Chatbot 11:09 PM
Sure! The shortest river in the world is the D River in Oregon, USA, which measures just over 37 meters (121 feet) in length.
Type your message here...
Built with Chainlit


Chatbot
localhost:8000
Chainlit
Chat
Readme
New Chat
Chatbot 11:38 PM
Model Orca initialized. How can I help you?
You 11:38 PM
What is the capital of Ukraine ?
Chatbot 11:38 PM
The capital city of Ukraine is Kyiv.
You 11:39 PM
And for France ?
Chatbot 11:39 PM
The capital city of France is Paris.
Type your message here...
Built with Chainlit


Chatbot
localhost:8000
Chainlit
Chat
Readme
New Chat
Chatbot 12:13 AM
Please set the model first by typing 'set model orca' or 'set model llama'.
You 12:13 AM
set model llama
Chatbot 12:13 AM
Model set to Llama.
You 12:13 AM
What is 2 + 2 ?
Took 1 step
LLMChain 12:13 AM
Of course! The answer to 2 + 2 is 4.
Took 1 step
CTransformers 12:13 AM
Of course! The answer to 2 + 2 is 4.
Chatbot 12:14 AM
Of course! The answer to 2 + 2 is 4.
Type your message here...
Built with Chainlit


Chatbot
localhost:8000
Chainlit
Chat
Readme
New Chat
Chatbot 12:20 AM
Please set the model first by typing 'set model orca' or 'set model llama'.
You 12:20 AM
set model orca
Chatbot 12:20 AM
Model set to Orca.
You 12:20 AM
What is 4 + 4 ?
Took 1 step
LLMChain 12:20 AM
The answer to the expression "4 + 4" is 8.
Took 1 step
CTransformers 12:20 AM
The answer to the expression "4 + 4" is 8.
Chatbot 12:21 AM
The answer to the expression "4 + 4" is 8.
Type your message here...
Built with Chainlit

Додаток Б

« Код файлів»

Код файлу "devcontainer.json":

```
// For format details, see https://aka.ms/devcontainer.json. For config options, see the
// README at: https://github.com/devcontainers/templates/tree/main/src/python
{
    "name": "Python 3",
    // Or use a Dockerfile or Docker Compose file. More info: https://containers.dev/guide/dockerfile
    "image": "mcr.microsoft.com/devcontainers/python:1-3.12-bullseye",
    "features": {
        "ghcr.io/devcontainers/features/python:1": {
            "installTools": true,
            "enableShared": true,
            "version": "latest"
        },
        "ghcr.io/devcontainers-contrib/features/black:2": {
            "version": "latest"
        },
        "ghcr.io/devcontainers-contrib/features/flake8:2": {
            "version": "latest",
            "plugins": "flake8-black"
        },
        "ghcr.io/devcontainers-contrib/features/vscode-cli:1": {
            "version": "latest"
        }
    },
    "customizations": {
        // Configure properties specific to VS Code.
        "vscode": {
            // Set *default* container specific settings.json values on container create.
            "extensions": [
                "dbaeumer.vscode-eslint",
                "GitHub.codespaces",
                "GitHub.github-vscode-theme",
                "GitHub.vscode-pull-request-github",
                "ms-python.black-formatter",
```

```
                "ms-python.debugpy",
                "ms-python.flake8",
                "ms-python.python",
                "ms-python.vscode-pylance"
            ]
        }
    },
    "postCreateCommand": "pip3 install --user -r requirements.txt"
    // Features to add to the dev container. More info:
https://containers.dev/features.
    // "features": {},

    // Use 'forwardPorts' to make a list of ports inside the container available
locally.
    // "forwardPorts": [],

    // Use 'postCreateCommand' to run commands after the container is created.

    // Configure tool-specific properties.
    // "customizations": {},

    // Uncomment to connect as root instead. More info: https://aka.ms/dev-
containers-non-root.
    // "remoteUser": "root"
}
```

Код файлу "requirements.txt":

```
aiofiles==23.2.1
aiohttp==3.9.3
aiosignal==1.3.1
annotated-types==0.6.0
anyio==3.7.1
asyncer==0.0.2
attrs==23.2.0
backoff==2.2.1
bidict==0.22.1
certifi==2024.2.2
chainlit==1.0.200
charset-normalizer==3.3.2
click==8.1.7
ctransformers==0.2.27
```

```
dataclasses-json==0.5.14
Deprecated==1.2.14
fastapi==0.108.0
fastapi-socketio==0.0.10
filelock==3.13.1
filetype==1.2.0
frozenlist==1.4.1
fsspec==2024.2.0
googleapis-common-protos==1.62.0
greenlet==3.0.3
grpcio==1.60.1
h11==0.14.0
httpcore==0.17.3
httpx==0.24.1
huggingface-hub==0.20.3
idna==3.6
importlib-metadata==6.11.0
isort==5.13.2
jsonpatch==1.33
jsonpointer==2.4
langchain==0.1.16
langchain-community==0.0.34
langchain-core==0.1.45
langchain-text-splitters==0.0.1
langsmith==0.1.49
Lazify==0.4.0
literalai==0.0.103
llamacpp==0.1.14
marshmallow==3.20.2
multidict==6.0.5
mypy-extensions==1.0.0
nest-asyncio==1.6.0
numpy==1.26.4
opentelemetry-api==1.22.0
opentelemetry-exporter-otlp==1.22.0
opentelemetry-exporter-otlp-proto-common==1.22.0
opentelemetry-exporter-otlp-proto-grpc==1.22.0
opentelemetry-exporter-otlp-proto-http==1.22.0
opentelemetry-instrumentation==0.43b0
opentelemetry-proto==1.22.0
opentelemetry-sdk==1.22.0
```

```
opentelemetry-semantic-conventions==0.43b0
orjson==3.10.1
packaging==23.2
protobuf==4.25.2
py-cpuinfo==9.0.0
pydantic==2.6.1
pydantic_core==2.16.2
PyJWT==2.8.0
python-dotenv==1.0.1
python-engineio==4.9.0
python-graphql-client==0.4.3
python-multipart==0.0.6
python-socketio==5.11.1
PyYAML==6.0.1
requests==2.31.0
simple-websocket==1.0.0
sniffio==1.3.0
SQLAlchemy==2.0.29
starlette==0.32.0.post1
syncer==2.0.3
tenacity==8.2.3
tomli==2.0.1
tqdm==4.66.1
typing-inspect==0.9.0
typing_extensions==4.9.0
uptrace==1.22.0
urllib3==2.2.0
uvicorn==0.25.0
watchfiles==0.20.0
websockets==12.0
wrapt==1.16.0
wsproto==1.2.0
yarl==1.9.4
zipp==3.17.0
transformers
sentencepiece
accelerate
```

Код моделі "orca.py":

```
import torch # type: ignore
from transformers import LlamaForCausalLM, LlamaTokenizer # type: ignore

# Our Model_Path
model_path = 'psmathur/orca_mini_3b'
tokenizer = LlamaTokenizer.from_pretrained(model_path)
model = LlamaForCausalLM.from_pretrained(
    model_path, torch_dtype=torch.float16, device_map='auto',
)


# Text Generation Functionality
def generate_text(system, instruction, input=None):

    if input:
        prompt = f"### System:\n{system}\n\n### User:\n{instruction}\n\n### 
Input:\n{input}\n\n### Response:\n"
    else:
        prompt = f"### System:\n{system}\n\n### User:\n{instruction}\n\n### 
Response:\n"

    tokens = tokenizer.encode(prompt)
    tokens = torch.LongTensor(tokens).unsqueeze(0)
    tokens = tokens.to('cuda')

    instance = {'input_ids': tokens, 'top_p': 1.0, 'temperature': 0.7, 
'generate_len': 1024, 'top_k': 50}

    length = len(tokens[0])
    with torch.no_grad():
        rest = model.generate(
            input_ids=tokens,
            max_length=length+instance['generate_len'],
            use_cache=True,
            do_sample=True,
            top_p=instance['top_p'],
            temperature=instance['temperature'],
            top_k=instance['top_k']
        )
    output = rest[0][length:]
```

```
    string = tokenizer.decode(output, skip_special_tokens=True)
    return f'[!] Response: {string}'

# A sample using a dataset from Hugging Face - we want our bot to answer questions
system = 'You are an AI assistant that gives helpful answers. You answer the question in a short and concise way.'
instruction = 'What is the capital city of Ukraine?'
print(generate_text(system, instruction))
```

Код моделі "llama.py":

```
from ctransformers import AutoModelForCausalLM

llm = AutoModelForCausalLM.from_pretrained(
    "TheBloke/Llama-2-7b-Chat-GGUF", model_file="llama-2-7b-chat.Q5_K_M.gguf"
)


def get_prompt(instruction: str) -> str:
    system = "You are an AI assistant that gives helpful answers. You answer the question in a short and concise way."
    prompt = f"<s>[INST] <<SYS>>\n{system}\n<</SYS>>\n\n{instruction} [/INST]"
    print(f"Prompt created: {prompt}")
    return prompt


question = "Which city is the capital of India?"
prompt = get_prompt(question)
for word in llm(prompt, stream=True):
    print(word, end="", flush=True)
print()
```

Код розширеної версії моделі "llama.py":

```
from ctransformers import AutoModelForCausalLM

llm = AutoModelForCausalLM.from_pretrained(
    "TheBloke/Llama-2-7b-Chat-GGUF", model_file="llama-2-7b-chat.Q5_K_M.gguf"
)

def get_prompt(instruction: str, conversation_history: str) -> str:
```

```
    system = "You are an AI assistant that gives helpful answers. You answer the question in a short and concise way."
    prompt = f"<s>[INST] <<SYS>>\n{system}\n<</SYS>>\n\n{conversation_history}\n{instruction} [/INST]"
    print(f"Prompt created: {prompt}")
    return prompt

# Initialize conversation history
conversation_history = ""

# First question
question = "Which city is the capital of India?"
prompt = get_prompt(question, conversation_history)
response = ""
for word in llm(prompt, stream=True):
    print(word, end="", flush=True)
    response += word
print()

# Update conversation history with the first question and response
conversation_history += f"{question}\n{response}\n"

# Second question
question = "And which for Germany?"
prompt = get_prompt(question, conversation_history)
response = ""
for word in llm(prompt, stream=True):
    print(word, end="", flush=True)
    response += word
print()

# Update conversation history with the second question and response
conversation_history += f"{question}\n{response}\n"
```

Код файлу моделі "Chainlit_Llama.py":

```
import chainlit as cl
from ctransformers import AutoModelForCausalLM

# Initialize global conversation history
conversation_history = []
```

```python
def get_prompt(instruction: str, history: list[str] | None = None) -> str:
    system = "You are an AI assistant that gives helpful answers. You answer the question in a short and concise way."
    prompt = f"<s>[INST] <<SYS>>\n{system}\n<</SYS>>\n\n"
    if history:
        prompt += f"{' '.join(history)}\n"
    prompt += f"{instruction} [/INST]"
    print(f"Prompt created: {prompt}")
    return prompt

@cl.on_message
async def on_message(message: cl.Message):
    global conversation_history
    msg = cl.Message(content="")
    await msg.send()

    # Create the prompt with the conversation history
    prompt = get_prompt(message.content, conversation_history)

    # Generate the response and update the conversation history
    response = ""
    for word in llm(prompt, stream=True):
        response += word
        await msg.stream_token(word)
    await msg.update()

    # Append the new question and response to the conversation history
    conversation_history.append(f"User: {message.content}")
    conversation_history.append(f"AI: {response}")

@cl.on_chat_start
async def on_chat_start():
    global llm, conversation_history

    # Reset the conversation history at the start of a new chat
    conversation_history = []

    llm = AutoModelForCausalLM.from_pretrained(
        "TheBloke/Llama-2-7b-Chat-GGUF", model_file="llama-2-7b-chat.Q5_K_M.gguf"
    )
```

```
    await cl.Message("Llama model initialized. How can I help you?").send()
```

Код файлу "Chainlit_Orca.py":

```
import chainlit as cl
from ctransformers import AutoModelForCausalLM

# Initialize global conversation history
conversation_history = []

def get_prompt(instruction: str, history: list[str] | None = None) -> str:
    system = "You are an AI assistant that gives helpful answers. You answer the
question in a short and concise way."
    prompt = f"### System:\n{system}\n\n"
    if history:
        prompt += f"{''.join(history)}\n"
    prompt += f"### User:\n{instruction}\n\n### Response:\n"
    print(f"Prompt created: {prompt}")
    return prompt

@cl.on_message
async def on_message(message: cl.Message):
    global conversation_history
    msg = cl.Message(content="")
    await msg.send()

    # Create the prompt with the conversation history
    prompt = get_prompt(message.content, conversation_history)

    # Generate the response and update the conversation history
    response = ""
    for word in llm(prompt, stream=True):
        response += word
        await msg.stream_token(word)
    await msg.update()

    # Append the new question and response to the conversation history
    conversation_history.append(f"### User:\n{message.content}\n\n###
Response:\n{response}\n")
```

```
@cl.on_chat_start
async def on_chat_start():
    global llm, conversation_history

    # Reset the conversation history at the start of a new chat
    conversation_history = []

    llm = AutoModelForCausalLM.from_pretrained(
        "zoltanctoth/orca_mini_3B-GGUF", model_file="orca-mini-3b.q4_0.gguf"
    )
    await cl.Message("Model Orca initialized. How can I help you?").send()
```

Код файлу Langchain "Langchain.py":

```
import chainlit as cl
from langchain.callbacks.base import BaseCallbackHandler
from langchain.chains import LLMChain
from langchain.memory import ConversationBufferMemory
from langchain_community.llms import CTransformers
from langchain_core.prompts import PromptTemplate

class StreamHandler(BaseCallbackHandler):
    def __init__(self):
        self.msg = cl.Message(content="")

    async def on_llm_new_token(self, token: str, **kwargs):
        await self.msg.stream_token(token)

    async def on_llm_end(self, response: str, **kwargs):
        await self.msg.send()
        self.msg = cl.Message(content="")

# Load quantized Orca and Llama models
orca_llm = CTransformers(
    model="juanjgit/orca_mini_3B-GGUF",
    model_file="orca-mini-3b.q4_0.gguf",
    model_type="llama2",
    max_new_tokens=20,
)

llama_llm = CTransformers(
```

```
    model="TheBloke/Llama-2-7B-Chat-GGUF",
    model_file="llama-2-7b-chat.Q2_K.gguf",
    model_type="llama2",
    max_new_tokens=20,
)

orca_template = """### System:\nYou are an AI assistant that gives helpful answers.
You answer the question in a short and concise way. Take this context into account
when answering the question: {context}\n
\n\n### User:\n{instruction}\n\n### Response:\n"""

llama_template = """
[INST] <<SYS>>
You are a helpful, respectful and honest assistant.
Always provide a concise answer and use the following Context:
{context}
<</SYS>>
User:
{instruction}[/INST]"""

orca_prompt = PromptTemplate(template=orca_template, input_variables=["context",
"instruction"])
llama_prompt = PromptTemplate(template=llama_template, input_variables=["context",
"instruction"])

@cl.on_chat_start
def on_chat_start():
    orca_memory = ConversationBufferMemory(memory_key="context")
    llama_memory = ConversationBufferMemory(memory_key="context")

    orca_chain = LLMChain(prompt=orca_prompt, llm=orca_llm, verbose=False,
memory=orca_memory)
    llama_chain = LLMChain(prompt=llama_prompt, llm=llama_llm, verbose=False,
memory=llama_memory)

    cl.user_session.set("orca_chain", orca_chain)
    cl.user_session.set("llama_chain", llama_chain)
    cl.user_session.set("selected_chain", None)

@cl.on_message
async def on_message(message: cl.Message):
```

```
    selected_chain = cl.user_session.get("selected_chain")

    if message.content.lower().startswith("set model orca"):
        selected_chain = cl.user_session.get("orca_chain")
        cl.user_session.set("selected_chain", selected_chain)
        await cl.Message(content="Model set to Orca.").send()
        return
    elif message.content.lower().startswith("set model llama"):
        selected_chain = cl.user_session.get("llama_chain")
        cl.user_session.set("selected_chain", selected_chain)
        await cl.Message(content="Model set to Llama.").send()
        return

    if selected_chain is None:
        await cl.Message(content="Please set the model first by typing 'set model
orca' or 'set model llama'.").send()
        return

    await selected_chain.ainvoke(
        {"instruction": message.content},
        config={"callbacks": [cl.AsyncLangchainCallbackHandler(), StreamHandler()]},
    )
```

ПРЕЗЕНТАЦІЯ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЙІНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

ДИПЛОМНА РОБОТА
на ступінь вищої освіти бакалавра
із спеціальності 122 Комп'ютерні науки

Побудова Чат-ботів з використання штучного інтелекту

Виконав: **студент 4 курсу, групи КНД-42**
Ель-Кафарна Джозеф Мухаммедович
Керівник: к.т.н. доцент Щербина І.С.

Київ - 2024

ЗАГАЛЬНА ХАРАКТЕРИСТИКА ДИПЛОМНОЇ РОБОТИ

Тема	Побудова Чат-ботів з використання ШІ.
Мета дослідження	Розуміння та підвищення роботи при використанням чат-ботів.
Наукове завдання	Розробка та побудування чат-ботів з використання ШІ за допомогою великі мовні моделі.
Об’єкт дослідження	Процедура побудови та розвитку чат-ботів з використанням штучного інтелекту.
Предмет дослідження	Використання великі мовні моделі при побудови чат-ботів.

Огляд ШІ та Чат-Ботів

- ШІ — це галузь комп’ютерних наук, яка займається створенням програмних систем, здатних розумно поводитися.
- Чат-ботів є комунікативними інструментами, які ефективно виконують звичайні завдання. Люди використовують їх тому, що вони дозволяють їм виконувати завдання швидше, що дозволяє їм зосередитися на інших важливих діяльностях, які вимагають людських здібностей, які не можуть бути повторені машинами.

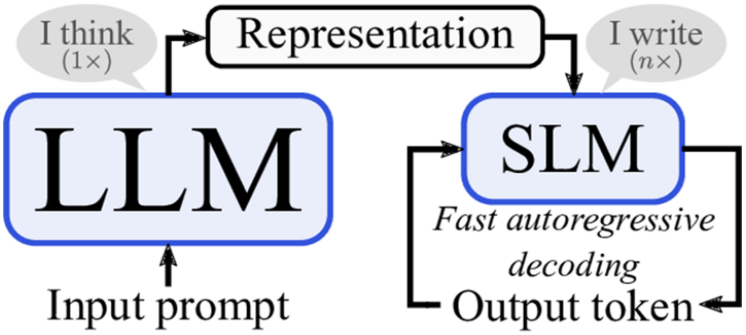
Типи Чат-ботів

- Обмежений - працює за чітко визначеного контексту і використовує визначену мову для реагування на конкретні запити клієнтів.
- Текстовий – бот розпізнає текст запиту, який клієнт вводить в поле, проводить аналіз повідомлень, після чого підбирає потрібний варіант із раніше заготовлених.
- Комунікаційний – лише для запитів споживачів. Він відповідає заздалегідь заготовленими словами, пропонує передзвонити або надає спеціальні пропозиції.
- Функціональний - альтернатива додаткам для смартфонів, що дозволяє шукати, бронювати, купувати, консультуватися з експертом тощо.

Поняття LLM

Велика мовна модель (LLM) — це сучасний підхід до розробки та застосування мовних моделей, які використовують ШІ. Зазвичай великі мовні моделі базуються на величезних наборах текстових даних і побудовані на основі рекурентних або трансформерних архітектур.

GPT (Generative Pre-trained Transformer), розроблений OpenAI, є однією з найвідоміших і впливових мовних моделей. Трансформерна архітектура GPT дозволяє йому генерувати текст і виконувати завдання, пов'язані з мовою, з вражаючою ефективністю та різноманітністю.



Технологій побудови Чат-ботів

Щоб створити чат-бота, необхідно поставити собі кілька запитань: - Які завдання буде виконувати чат-бот?

- В якому середовищі буде розгорнутий ваш чат-бот, локальному чи хмарному?
- Яку мову програмування ви будете використовувати для створення чат-бота?
- Які моделі LLM підійдуть для створення вашого чат-бота?
- Які бібліотеки та відкриті джерела ви будете використовувати для розвитку та покращення вашого чат-бота?

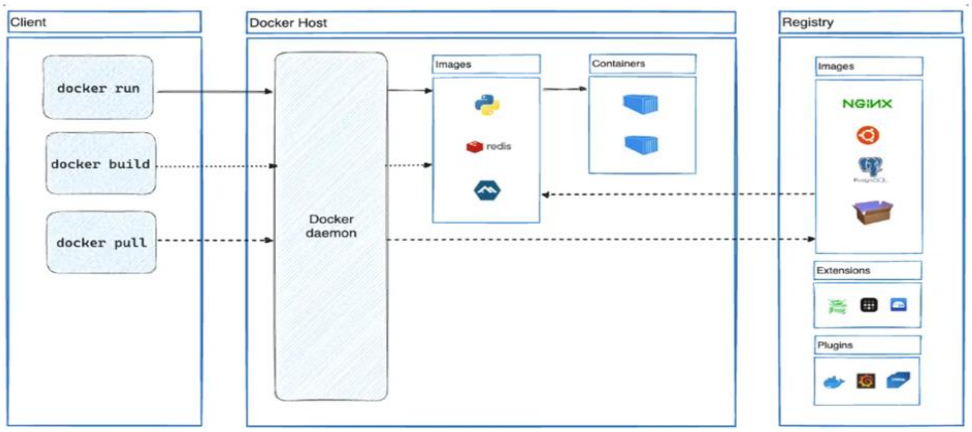
Мова програмування Python.

- Python — це потужна мова програмування, яка проста у вивченні. Він має ефективні структури даних високого рівня та простий, але ефективний підхід до об’єктно-орієнтованого програмування. Елегантний синтаксис і динамічна типізація Python разом з його інтерпретованим характером роблять його ідеальною мовою для створення сценаріїв і швидкої розробки додатків у багатьох сферах на більшості платформ.

Docker

Docker – це програмне забезпечення з відкритим кодом, найпопулярніша платформа для управління контейнерами.

Він потрібен для більш ефективного використання системи і ресурсів, швидкого розгортання готових програмних продуктів, а також для їх масштабування і перенесення в інші середовища з гарантованим збереженням стабільної роботи.



Вибір LLM моделі

Orca

Microsoft Research створила модель Orca LLM, яка є складною великою мовною моделлю, призначеною для імітації процесів мислення менших, але потужніших моделей.

Orca має особливий спосіб удосконалюватися в тому, що вона робить: вона вчиться на тому, як мислять менші моделі, а потім поєднує це з величезною кількістю інформації, яку мають розуміти більші моделі, щоб створювати кращі тексти.

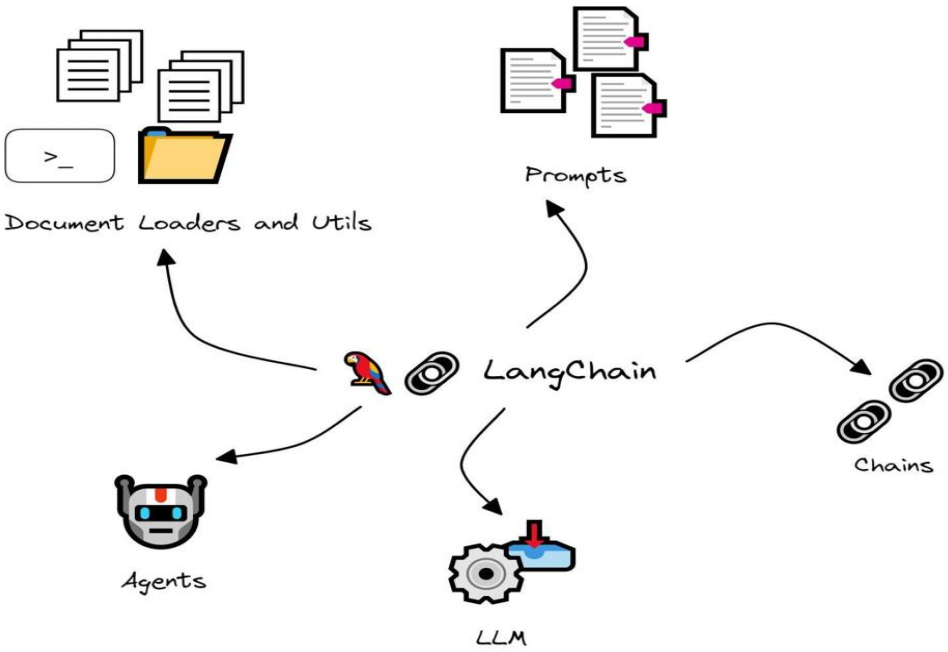
Llama

Llama - це мовна модель від компанія Meta, яка призначена для розуміння та створення текстів, подібних до людського письма. Це популярний вибір для створення складних чат-ботів завдяки своїй здатності створювати цікаві та контекстно- відповідні відповіді.

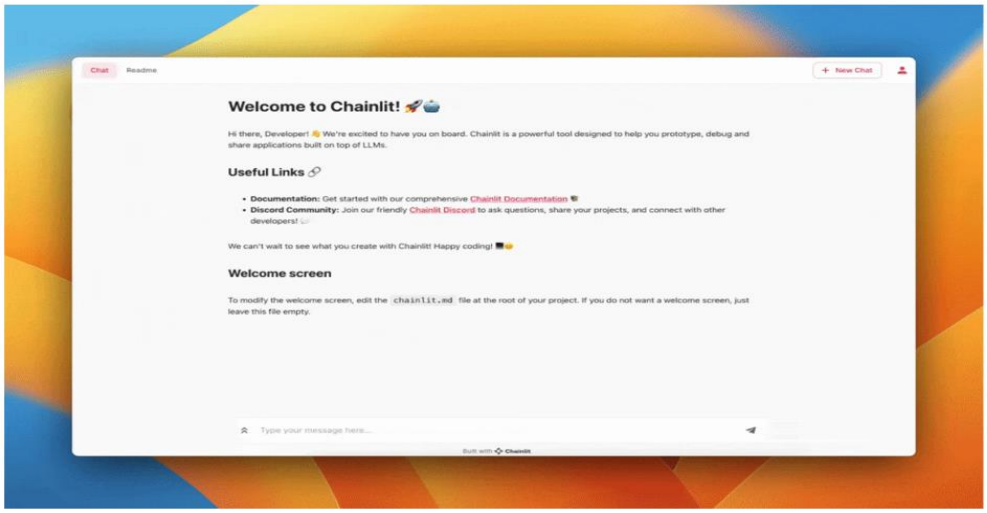
Chainlit

Chainlit - це пакет Python з відкритим вихідним кодом для створення готового до виробництва розмовного ШІ.

Він використовується для надання інтерфейсу користувача для чат-ботів, щоб покращити їхню функціональність.



Демонстрація отриманого результату побудованого Чат-бот.



Висновок

- Ми обговорили ШІ та чат-боти загалом, а також те, як LLM відіграє важливу роль у розробці різних типів чат-ботів, включаючи розмовних та обмежених чат-ботів.
- Ми детально дізналися про те, як LLM працює над розробкою чат-ботів, а також про різні типи агентів LLM.
- Ми також обговорили технології, що використовуються для створення чат-ботів, такі як Docker, Python, Chainlit LangChain, а також моделі Orca та Llama LLM.