

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра Інформаційних технологій

Кваліфікаційна робота магістра

на тему: Дослідження та розробка алгоритмів для штучного
інтелекту та нейронних мереж

Виконав студент групи МІС-22
спеціальності 122 Комп'ютерні науки
Осташков Микита Ігорович

Керівник к.т.н., доцент
Фразе-Фразенко Олексій Олексійович

Рецензент регіональний
координатор програми EGAP
в Одеській області,
Копиченко Іван Юрійович

АНОТАЦІЯ

на магістерську кваліфікаційну роботу «Дослідження та розробка алгоритмів для штучного інтелекту та нейронних мереж»,
студента Осташкова Микити Ігоровича

Актуальність теми обумовлена швидкими та постійними змінами в інформаційних технологіях, які створюють потребу в нових підходах до взаємодії організацій, зокрема вищих навчальних закладів, з користувачами їхніх послуг. Діалоговий агент, який може читати та аналізувати надану йому документацію, а також зберігати історію взаємодії з користувачем, виявляється важливим інструментом у цьому контексті. Його вартість значно менша, ніж утримання додаткового персоналу, він готовий до використання у будь-який час доби, легко взаємодіє з користувачами, відповідає вимогам сучасного темпу життя та сприяє поліпшенню комунікації із зацікавленими користувачами.

Мета роботи: розробка діалогового агента з елементами штучного інтелекту. Цей діалоговий агент призначений для читання та аналізу документації, а також зберігання історії взаємодії з користувачами. Його основною метою є полегшення доступу до інформації для студентів, викладачів та абітурієнтів факультету. Для досягнення цієї мети були вирішені такі завдання: аналіз предметної області застосування діалогових агентів, порівняльний аналіз існуючих рішень, вибір відповідних технологій для розробки, проектування системи, реалізація та тестування роботи діалогового агента.

Об'єкт дослідження моєї магістерської роботи - діалоговий агент, який виступає в ролі діалогового агента для забезпечення інформаційної підтримки користувачів. Методи дослідження включають аналіз та синтез для визначення переваг та недоліків існуючих аналогів, розгляд методів побудови діалогових систем, використаних у розробці діалогового агента, і експериментальне дослідження для перевірки достовірності отриманих результатів. Це дозволяє

систематично досліджувати та вдосконалювати процес створення діалогового агенту з використанням штучного інтелекту.

В даній магістерській роботі висвітлено значущість інтерактивних діалогових агентів в контексті інформаційних технологій. Проведено аналіз доступних засобів для розробки агентів з використанням технологій обробки природної мови. Результатом цього дослідження є створення інтерактивного цифрового агенту.

Магістерська кваліфікаційна робота містить 70 сторінок, 33 рисунків, 2 таблиць, список джерел з 11 найменувань.

Ключові слова: НЕЙРОННА МЕРЕЖА, ШТУЧНИЙ ІНТЕЛЕКТ, АГЕНТ

ABSTRACT

for a master's thesis

"Research and development of algorithms for artificial intelligence
and neural networks ",
student Ostashkov Mykyta

The relevance of the topic is due to rapid and constant changes in information technologies, which create a need for new approaches to the interaction of organizations, in particular higher education institutions, with users of their services. A dialog agent, which can read and analyze the documentation provided to it, as well as store the history of the interaction with the user, turns out to be an important tool in this context. It costs much less than maintaining additional staff, is ready for use at any time of the day, easily interacts with users, meets the requirements of the modern pace of life and helps to improve communication with interested users.

The purpose of the work: development of a dialogue agent with elements of artificial intelligence. This dialog agent is designed to read and analyze documentation, as well as store the history of interaction with users. Its main goal is to facilitate access to information for students, teachers and faculty applicants. To achieve this goal, the following tasks were solved: analysis of the subject area of application of dialog agents, comparative analysis of existing solutions, selection of appropriate technologies for development, system design, implementation and testing of dialog agent work.

The object of research of my master's work is a dialog agent, which acts as a dialog agent to provide informational support to users. Research methods include analysis and synthesis to determine the advantages and disadvantages of existing analogues, consideration of methods of building dialog systems used in the development of a dialog agent, and experimental research to verify the reliability of

the obtained results. This allows you to systematically research and improve the process of creating a dialog agent using artificial intelligence.

This master's thesis highlights the importance of interactive dialog agents in the context of information technologies. An analysis of the available tools for the development of agents using natural language processing technologies has been carried out. The result of this research is the creation of an interactive digital agent.

The master's thesis contains 70 pages, 33 figures, 2 tables, a list of sources of 11 items.

Keywords: NEURON NETWORK, ARTIFICIAL INTELLIGENCE, AGENT

ЗМІСТ

ПЕРЕЛІК скорочень.....	9
ВСТУП.....	10
1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТ.....	11
1.1. Аналіз предметної області.....	11
1.2 Аналіз існуючих програмних систем.....	12
2. ОБГРУНТУВАННЯ ЗАСОБІВ РОЗРОБКИ.....	17
2.1 Деталізація задач проекту.....	17
2.2 Обґрунтування вибору типу агента.....	18
2.3 Вибір API.....	21
3. ДОСЛІДЖЕННЯ ТЕХНОЛОГІЇ ОБРОБКИ ПРИРОДНОЇ МОВИ.....	27
3.1. Поняття обробки природної мови.....	27
3.2 Глибоке навчання у NLP.....	29
3.3 Застосування технології обробки природної мови.....	30
3.4 Інструменти для розробки помічника з використанням техно- логії NLP.....	36 36
3.5 Вибір мови програмування для розробки агента штучного інтелекту.....	40
4. ПРОЕКТУВАННЯ ДІАЛОГОВОГО АГЕНТА.....	48
4.1 Проектування варіантів використання системи.....	48
4.2 Створення діаграми діяльності.....	49
4.3 Проектування логіки сценаріїв використання.....	50
5. ПРОГРАМНА РЕАЛІЗАЦІЯ.....	53
5.1. Створення агента на основі OpenAI.....	53
5.2. Встановлення та Налаштування Оточення.....	55
5.3. Основні функції та код штучного співрозмовника.....	66
5.4. Тестування проекту та діалогового агента.....	69
ВИСНОВКИ.....	76
ПЕРЕЛІК ПОСИЛАНЬ.....	77

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних

ОС – операційна система

ПЗ – програмне забезпечення

AI, ШІ – Artificial intelligence – штучний інтелект

API – Application Programming Interface – програмний інтерфейс програми

APM – Application Performance Monitoring – моніторинг роботи застосунків

ASIC – Application-Specific Integrated Circuit – інтегральна схема для специфічного застосування

IDE – Integrated Development Environment – інтегроване середовище розробки

ML – Machine Learning – машинне навчання

NLP – Natural Language Processing – обробка природної мови

TPU – Tensor Processing Unit – тензорний процесор

UML – Unified Modeling Language – уніфікована мова моделювання

URL – Uniform Resource Locator – єдиний вказівник на ресурс

VoIP – Voice over IP – голосовий зв'язок через протокол IP

WBS – Work Breakdown Structure – ієрархічна структура робіт

ВСТУП

Штучний інтелект та нейронні мережі відіграють ключову роль у сучасних технологічних розробках, привносячи в наше життя інновації та можливості, які ще кілька десятиліть тому видавалися неймовірними. Зокрема, нейронні мережі стають все більш затребуваними у суспільстві, забезпечуючи вирішення складних завдань та надаючи нові перспективи у різних галузях.

В наш час часто виникає потреба в оперативному та ефективному отриманні інформації з конкретних тем, особливо у вузькоспрямованих галузях. Швидкі та точні відповіді на специфічні питання відіграють важливу роль у професійній діяльності, наукових дослідженнях та повсякденному житті. Саме в цьому контексті актуальність створення та використання штучного інтелекту, здатного швидко та ефективно відповідати на запити у вузькоспрямованих галузях, стає очевидною.

Метою даної роботи є дослідження та розробка алгоритмів для створення діалогового агента на основі нейронних мереж, здатного надавати інформацію та відповідати на питання в рамках вузькоспеціалізованих тематик. Створення такого діалогового агента є важливим кроком у напрямку підвищення доступності та ефективності роботи з інформацією, специфічною для певних областей знань.

Цей діалоговий агент може бути корисним у різних галузях, де потрібен оперативний обмін інформацією: від консультування у сфері медицини та юриспруденції до надання експертної підтримки з технічних питань. Розроблений алгоритм повинен демонструвати здатність ефективно обробляти запити користувачів та надавати інформацію, що відповідає вузькоспеціалізованим темам.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз предметної області.

У сучасному світі штучний інтелект стає ключовим елементом технологічного прогресу, змінюючи безліч аспектів нашого повсякденного життя. Це область інформатики, присвячену розробці систем, здатних до навчання та самостійного прийняття рішень. Однією з найперспективніших технологій у рамках цієї області є нейронні мережі.

Нейронні мережі це алгоритми машинного навчання, інспіровані будовою людського мозку. Вони складаються з багатьох взаємозалежних нейронів, об'єднаних у шари. Кожен нейрон обробляє вхідні дані та передає результат наступному шару. Ця структура дозволяє нейронним мережам виявляти складні закономірності даних і виконувати завдання, що вимагають високого ступеня абстракції.

У галузі штучного інтелекту та нейронних мереж активно досліджуються методи покращення процесів навчання, оптимізації архітектур мереж та розширення областей їх застосування. Просунуті технології обробки великих обсягів даних та обчислень у реальному часі дозволяють створювати нейронні мережі, здатні вирішувати складні завдання у різних галузях, таких як медицина, фінанси, освіта та технологічні процеси.

Діалогові агенти є програмами, спроектованими для автоматичної взаємодії з користувачами через текстові інтерфейси. Вони можуть виконувати ряд функцій від надання інформації до виконання конкретних завдань. Діалогові агенти широко використовуються в різних сферах, включаючи обслуговування клієнтів, технічну підтримку, навчання та розваги.

Переваги діалогових агентів включають оперативність, доступність 24/7, здатність обробляти велику кількість запитів одночасно та стабільність у наданні інформації. Вони забезпечують зручний спосіб взаємодії з користувачами, а їх використання дозволяє скоротити навантаження на реальних операторів та підвищити ефективність обслуговування.

Сучасні діалогові агенти, засновані на нейронних мережах, набувають здатність до більш природної та контекстної взаємодії з користувачами. Вони здатні до навчання на основі попередніх взаємодій та адаптації до умов обслуговування, що змінюються. Можливість створення діалогових агентів, які мають елементи штучного інтелекту, відкриває нові перспективи розробки більш просунутих і ефективних систем спілкування.

1.2 Аналіз існуючих програмних систем

Існує безліч діалогових агентів, розроблених різними компаніями та організаціями. Вони знайшли застосування у різних галузях і стали невід'ємною частиною взаємодії з користувачем. Розглянемо кілька успішних прикладів діалогових агентів та їх застосування у світі. 1. Google Assistant Google Assistant – це багатофункціональний віртуальний помічник, розроблений Google. Він є інтелектуальним голосовим і текстовим інтерфейсом, призначеним для забезпечення користувачів широким спектром сервісів і функцій.

Google Assistant включає потужний голосовий інтерфейс, який дозволяє користувачам взаємодіяти з пристроями, використовуючи свій голос. Система розпізнавання мови дозволяє агенту точно розуміти команди та питання, забезпечуючи природну взаємодію.

Google Assistant активно інтегрований у різні розумні пристрої, такі як смартфони, розумні стовпчики, телевізори, годинники та інші гаджети. Це дозволяє користувачам керувати своїми пристроями та отримувати інформацію без необхідності фізичної взаємодії.

Агент надає інформацію користувача, таку як поточна погода, новини, котирування акцій, результати спортивних подій та багато іншого. Користувачі можуть отримувати актуальні дані, надаючи відповідні запити.

Google Assistant є важливою частиною екосистеми розумного будинку. Користувачі можуть керувати розумними пристроями, такими як термостати, освітлення, замки та камери за допомогою голосових команд. Віртуальний помічник адаптується до уподобань користувача. Він може запам'ятовувати

попередні запити, надавати персоналізовані рекомендації та допомагати у вирішенні повсякденних завдань.

Google Assistant підтримує різні "навички" або функціональні можливості, які надаються сторонніми розробниками. Користувачі можуть взаємодіяти з програмами та сервісами, розширюючи функціональність агента.

У Google Assistant вбудований в операційні системи Android та iOS. Це забезпечує широке поширення агента серед користувачів мобільних пристроїв.

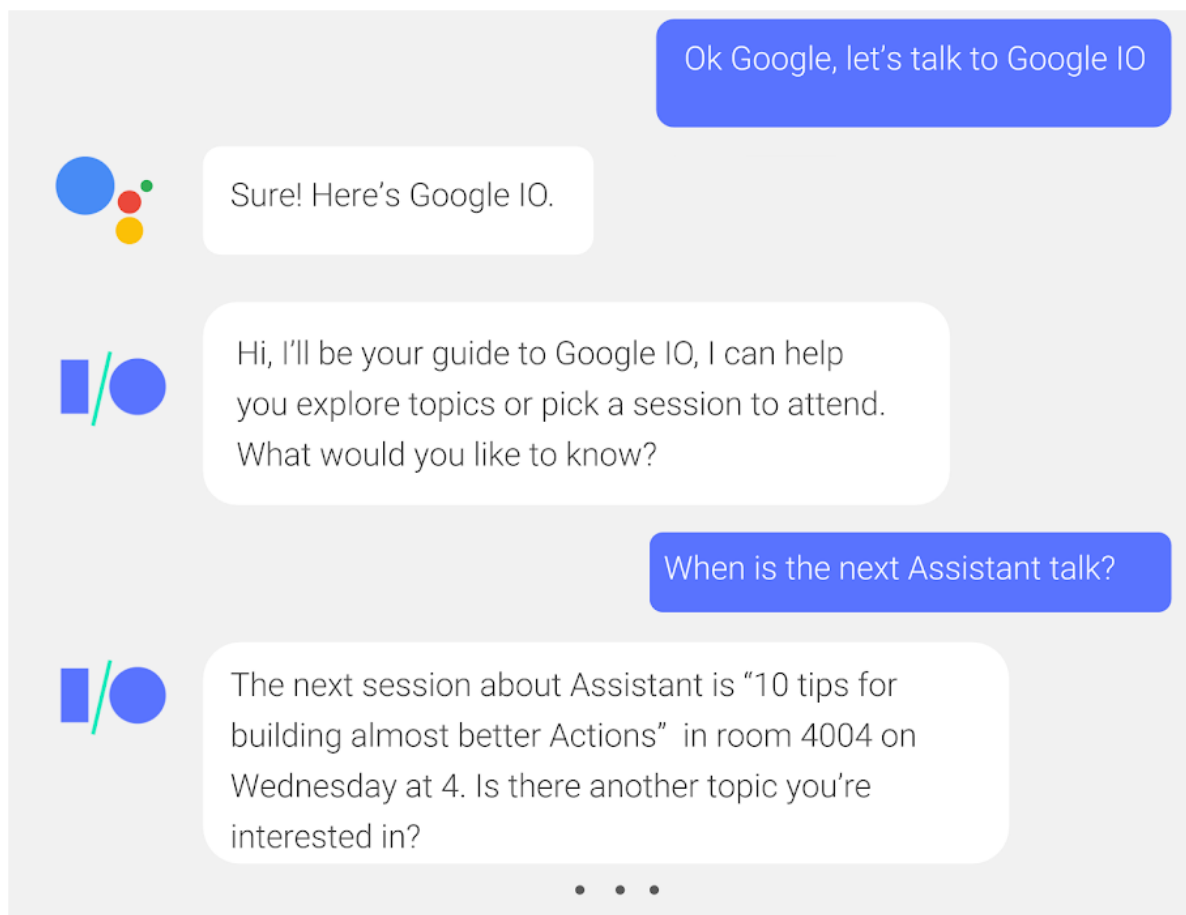


Рисунок 1.1 – Скріншот Google Assistant

2. Siri від Apple

Siri є віртуальним помічником, розробленим Apple, здатним обробляти природну мову і виконувати різноманітні завдання щодо голосових команд користувача. Запущений у 2011 році з виходом iPhone 4S, Siri стала ключовою частиною операційних систем Apple, таких як iOS, macOS, watchOS та tvOS.

Siri забезпечує голосове керування пристроями Apple. Користувачі можуть запитувати інформацію, ставити запитання, давати команди та багато іншого, використовуючи природну мову.

Siri враховує контекст та пропонує персоналізовані підказки залежно від часу доби, локації, попередніх дій користувача та інших факторів.

Siri допомагає організовувати робочі завдання, створювати нагадування, встановлювати зустрічі в календарі та надсилати повідомлення.

Siri інтегрована з різними додатками, що дозволяє виконувати дії в сторонніх додатках. Наприклад, користувач може надіслати повідомлення через WhatsApp або замовити таксі через Uber.

Siri керує музикою, відео та іншими розважальними контентом. Користувач може попросити її увімкнути певну пісню, альбом або надати інформацію про фільм.



Рисунок 1.2 – Скріншот Siri

3. ChatGPT

ChatGPT - чат-бот з генеративним штучним інтелектом, розроблений компанією OpenAI і здатний працювати в діалоговому режимі, що підтримує

запити природними мовами. Система здатна відповідати на питання, генерувати тексти різними мовами, включаючи російську, що належать до різних предметних областей. Важливою особливістю є можливість генерації на запит програм різними мовами програмування.

ChatGPT – велика мовна модель, для тренування якої використовувалися методи навчання з учителем та навчання з підкріпленням. Цей чат-бот ґрунтується на іншій мовній моделі від OpenAI – GPT-3.5 – покращеної версії моделі GPT-3. 14 березня 2023 року була випущена мовна модель GPT-4, доступна тестувальникам та платним передплатникам ChatGPT Plus. У новій версії у нї з'явилася можливість обробки не лише тексту, а й картинок.

Дані у розпорядженні у ChatGPT включають довідкові сторінки, інформацію про інтернет-феномени та інформацію про мови програмування. На відміну від більшості чат-ботів, ChatGPT відстежує стан користувача, запам'ятовуючи попередні запитання та відповіді, дані йому в тій же розмові, що, на думку деяких журналістів, дозволить використовувати ChatGPT як персоналізований терапевт. У спробі запобігти видачу образливих результатів від запитів користувачів та отримання образливих відповідей, ChatGPT запити фільтруються через API модерації.

You

Hello! Can you tell me when mathematics was established as a science?

ChatGPT

Mathematics has a long and rich history, and it is challenging to pinpoint an exact moment when it was established as a science. The development of mathematical ideas and concepts has evolved over thousands of years and across various cultures.

Ancient civilizations such as the Egyptians, Babylonians, Greeks, and Indians all made significant contributions to mathematics. The Greeks, particularly in the classical period, are often credited with laying the foundations of mathematics as a formalized and systematic discipline. Mathematicians like Euclid, Pythagoras, and Archimedes made lasting contributions to geometry, number theory, and other branches of mathematics.

However, it's important to note that different branches of mathematics developed at different times. For instance, algebra and symbolic notation were significantly developed in the Islamic Golden Age, with mathematicians like Al-Khwarizmi contributing to the field.

Рисунок 1.3 – Діалог з ChatGPT

Порівняння функцій перерахованих вище помічників наведено у табл. 1.

Критерій порівняння	Google Assistant	Siri	ChatGPT
Інтеграція із Сервісами	Так	Так	Ні
Голосове керування	Так	Так	Ні
Багатоцільова Функціональність	Ні	Ні	Так

Google Assistant і Siri орієнтовані широкий спектр завдань, включаючи голосове управління, тоді як ChatGPT сконцентрований на текстових запитах і завданнях. При створенні агента кваліфікаційної роботи було взято за основу ChatGPT.

2. ОБГРУНТУВАННЯ ЗАСОБІВ РОЗРОБКИ

2.1 Деталізація задач проекту

Метою кваліфікаційної роботи є розробка діалогового агента з елементами штучного інтелекту для обробки запитів природною мовою. Даний агент буде корисним для діяльності в будь-якій сфері, будь то робота чи навчання.

Для досягнення поставленої мети в роботі були сформульовані наступні завдання:

- провести аналіз предметної області використання цифрових агентів;
- провести порівняльний аналіз існуючих аналогів;
- обґрунтувати вибір програмних засобів розробки та технологій;
- провести проектування системи;
- виконати програмну реалізацію цифрового агента;
- підготувати інструкцію користувача;
- виконати тестування розгорнутого проекту.

Після проведення детального аналізу предметної області наступними етапами є проектування та програмна реалізація агента. Після того, як буде виконана розробка агента, наступним необхідним етапом є тестування. Недоліки, виявлені на етапі тестування мають бути проаналізовані та виправлені.

Перед початком розробки цифрового агента були визначені функціональні вимоги до нього.

До функціональних вимог належать:

- Проведення з користувачем розмови як із справжнім співрозмовником;
- Здатність прочитання та обробки інформації з псевдо бази даних, що знаходиться у діалогового агента;
- Збереження минулих розмов із користувачем;
- Обговорення всіх тонкощів специфічної теми для розмови залежно від даних;

2.2 Обґрунтування вибору типу агента

Діалогові агенти можуть бути впроваджені у вигляді самостійного програмного застосунку, веб-сторінки або інтегровані в застосунки для обміну повідомленнями. Мобільні або настільні застосунки призначені для конкретної платформи, такої як iOS, Android або Windows. Користувач завантажує та встановлює програму з відповідного магазину програм. Переваги застосунків включають: 1. Інтерактивність: Застосунок представляє новий канал взаємодії з поточними користувачами, надаючи новий функціонал та інтегруючи недоступні на сайті функції. 2. Персоналізація: Програми дозволяють користувачам налаштовувати свої уподобання відразу після завантаження та налаштовувати їх відповідно до своїх потреб. Вони також можуть використовувати взаємодію з користувачем для надання індивідуальних рекомендацій та оновлень, роблячи програму більш корисною. 3. Робота в автономному режимі: Програми можуть працювати без Інтернет-з'єднання, надаючи користувачам доступ до інформації в будь-який час і в будь-якому місці. 4. Використання можливостей пристрою: Мобільні або настільні застосунки можуть використовувати вбудовані функції пристрою, такі як камера, GPS, місцезнаходження і взаємодія з іншими IoT-пристроями або розумною технікою.

Незважаючи на ці переваги, є й недоліки, такі як потреба в створенні окремих застосунків для різних операційних систем, складність розробки і вартість розгортання, включаючи публікацію в магазинах та необхідність завантаження додаткових програм і вивчення інтерфейсу кожного нового застосунку.

Адаптивні веб-сайти - це веб-ресурси, які можуть автоматично адаптуватися до різних розмірів екранів. Переваги таких сайтів включають: 1. Доступність для всіх користувачів: На відміну від мобільних додатків, які обмежені певними платформами (iOS, Android і т.д.), адаптивний веб-сайт доступний з будь-якого пристрою, що має підключення до Інтернету. Важливо

враховувати, що якість і швидкість Інтернет-з'єднання можуть впливати на функціональність сайту. 2. Без необхідності встановлення оновлень: Користувачам не треба витратити час на встановлення нових версій або оновлень для поліпшення роботи сайту. Оскільки адаптивні веб-сайти легко оновлюються та підтримуються, користувачі можуть не помічати процесу оновлення і миттєво користуватися новими можливостями. 3. Економічна ефективність: Створення адаптивного веб-сайту коштує значно менше, ніж розробка додатка.

Серед недоліків адаптивних веб-сайтів слід відзначити швидкість роботи та те, що деякі мобільні користувачі віддають перевагу застосункам, а не використанню браузера.

Помічники, що впроваджені у застосунки для обміну повідомленнями (месенджери, соціальні мережі), наразі є дуже популярними з численних причин: 1. Відсутність необхідності вивчати новий інтерфейс користувача: У порівнянні з веб-сайтами та мобільними застосунками, які мають різні візуальні інтерфейси, використання месенджерів стає дедалі популярнішим. Взаємодія з організаціями за допомогою агента в цих застосунках вважається природною та зручною. Інтерфейс користувача при користуванні агентом в месенджерах є стандартним, тому не вимагає вивчення нового інтерфейсу. Ось чому текстові повідомлення та голосові команди стають основними та зручними засобами взаємодії, особливо в умовах, коли багато людей не мають бажання чи часу вивчати нові користувацькі інтерфейси. 2. Справжньо персоналізований досвід користувача: Здатність цифрових агентів аналізувати мову, розуміти наміри користувачів та завжди мати контекст дозволяє створювати значно більш значущі та персоналізовані враження. Діалогові агенти легко сегментують свою аудиторію та надають різним користувачам різний досвід. Кожен раз, коли користувач відкриває месенджер та починає спілкуватися в чаті, організація має можливість впізнати його та отримує доступ до всіх попередніх взаємодій. Таким чином, цифровий агент стає все розумнішим з кожною новою

взаємодією, постійно вивчаючи поведінку користувача та надаючи більш персоналізовані відповіді на основі попередніх повідомлень, скарг та запитань.

3. Ефективне залучення користувачів: Існує безліч інструментів для взаємодії організацій і компаній з користувачами — від електронного маркетингу до соціальних мереж і веб-сайтів, які дозволяють досягти широкої аудиторії. На жаль, ці інструменти не завжди дозволяють вести спілкування на індивідуальному рівні. Унікальність цифрових агентів полягає в можливості надавати індивідуально настроєні послуги та взаємодіяти з великою кількістю користувачів на індивідуальному рівні. Інформація, що збирається про користувачів, може бути використана для повторної взаємодії та побудови міцніших відносин, що спрямовані на досягнення конкретної мети, наприклад, здійснення покупки товару. Організації можуть висилати релевантні та персоналізовані повідомлення в потрібний момент, використовуючи одну з найпопулярніших програм на кожному смартфоні.

4. Покращений та швидший користувацький досвід: Розмовна мова є найбільш природнім способом спілкування між людьми, а програми обміну повідомленнями — це найновіший спосіб використання мови як основного засобу зв'язку між користувачами та організаціями. Однією з значущих переваг агентів є можливість майже миттєво отримувати користувацьку інформацію за допомогою базових засобів обробки природної мови (NLP), уникнувши необхідності в ручному пошуку на веб-сайтах.

5. Безперебійний перехід до обслуговування людьми: Самі по собі цифрові агенти не розв'язують всі проблеми, з якими можуть зіткнутися користувачі, оскільки веб-сайти та додатки також не завжди здатні це зробити. У цифрових агентах це вирішується через обмеженість технології обробки природної мови (NLP) і те, що вони іноді не можуть надати очікуваної відповіді. Тим не менше, у цифрових агентів завжди є можливість переходу до агента-людини, який готовий безперебійно вступити в розмову. У порівнянні з веб-сайтами важко визначити, коли у користувача виникають сумніви, і якщо він закриває вкладку, він може бути втрачений назавжди. З цифровим агентом

відповідь від агента-людини призведе до повідомлення на телефоні, що дозволить знову залучити користувача.

Таким чином, за підсумком аналізу існуючих варіантів розробки агента, була обрана реалізація програмного застосунку.

2.3 Вибір API

Існують декілька платформ та компанії, які надають послуги інтелектуального аналізу мови та генерації тексту. Декілька з них:

Microsoft Azure Cognitive Services:

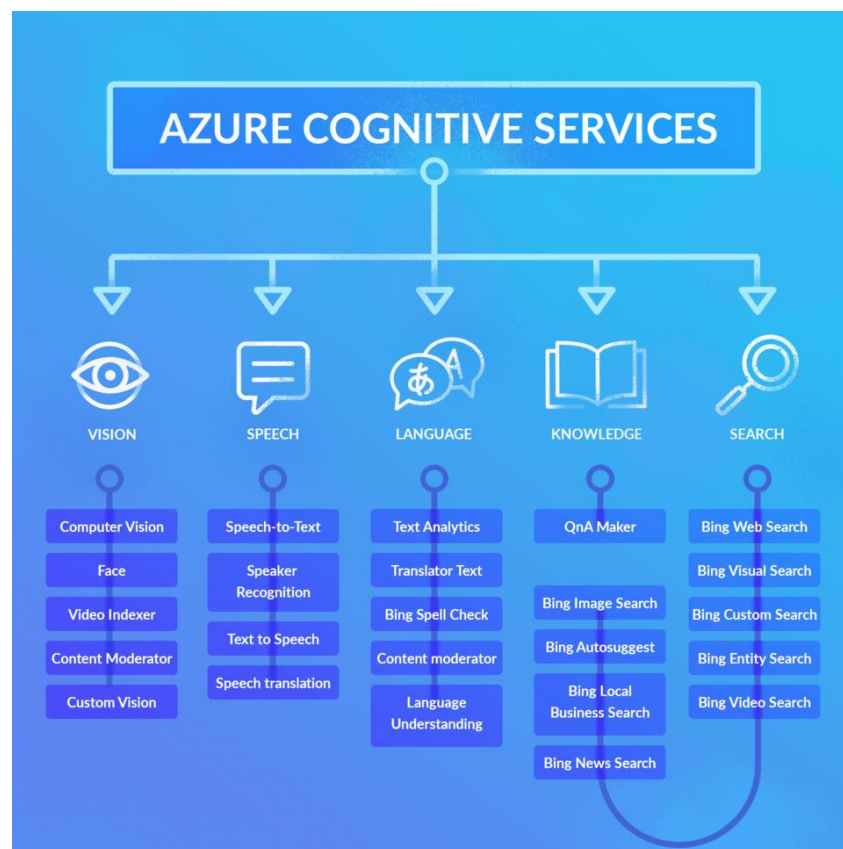


Рисунок 2.1 – Microsoft Azure Cognitive Services

Microsoft Azure Cognitive Services - це набір хмарних служб, які надають інструменти для роботи зі штучним інтелектом та обробкою природної мови. Ці служби розроблені для використання у різних сценаріях, включаючи аналіз тексту, розпізнавання мови, роботу зі зображеннями та інші завдання інтелектуального аналізу даних. Деякі ключові складові Cognitive Services: Text Analytics: Сервіс для аналізу тексту, який дозволяє визначати ключові фрази, виявляти мовні тональності, розпізнавати сутності та проводити інші завдання

обробки тексту. Language Understanding (LUIS): Цей сервіс надає можливості створення і навчання моделей обробки природної мови для розпізнавання намірів та сутностей у тексті. QnA Maker: Інструмент для створення систем питань-відповідей (QnA) на основі тексту. Він може автоматично створювати базу знань для відповідей на типові запитання. Translator Text: Сервіс перекладу тексту для автоматизованого перекладу текстової інформації між різними мовами. Computer Vision: Сервіс для обробки та аналізу зображень, включаючи розпізнавання облич, класифікацію зображень і виявлення об'єктів.

Ці служби можна використовувати окремо чи в комбінації для створення різних інтелектуальних застосунків. Azure Cognitive Services надає розробникам доступ до потужних інструментів штучного інтелекту без необхідності глибоких знань у цій області.

Google Cloud Natural Language API:

The screenshot shows the Google Cloud Natural Language API interface. At the top, there is a text input field containing the sentence: "So, whether or not, your business is in the IT sector, it is essential that Business Leaders know the AI and ML trends and can keep up with the pace and change of the business environment." To the right of the input is a "RESET" button. Below the input is a link "See supported languages".

The analysis results are displayed in a table with four tabs: "Entities", "Sentiment", "Syntax", and "Categories". The "Syntax" tab is selected. At the top of the Syntax section, there are five checked options: "Dependency" (green), "Parse label" (orange), "Part of speech" (red), "Lemma" (purple), and "Morphology" (blue).

The main part of the screenshot shows a dependency parse tree for the sentence "Ever since the advent of Globalisation, the environment in which a". The words are listed in a row, with their corresponding part-of-speech tags and dependency labels above them. Green arrows indicate the syntactic relationships between the words.

Label	Parse label	Part of speech	Lemma	Morphology							
advmod	prep	det	pobj	prep	pobj	p	det	nsubj	prep	pobj	de
Ever	since	the	advent	of	Globalisation	,	the	environment	in	which	a
ADV	ADP	DET	NOUN number=PLURAL	ADP	NOUN number=SINGULAR	PUNCT	DET	NOUN number=SINGULAR	ADP	DET	DE

Рисунок 2.2 – Google Cloud Natural Language API: Syntax Analysis

Google Cloud Natural Language API - це служба для аналізу тексту, яка дозволяє виявляти мовну тональність, розпізнавати сутності, визначати

категорії тексту та виконувати інші завдання обробки природної мови. Основні можливості API включають: Аналіз сутностей: Сутності включають в себе різні типи об'єктів, таких як особи, місця, події, організації та інше. API може визначати ці сутності в тексті.

Визначення категорій: Визначення загальної теми тексту або категорії, до якої він відноситься.

Аналіз настрою: Визначення емоційного забарвлення тексту, такого як позитивний, негативний чи нейтральний настрій. Виявлення мовної тональності: Аналіз тексту для виявлення тональності висловлення, такої як позитивна, негативна або нейтральна.

Розпізнавання синтаксичних відносин: Аналіз структури речення для визначення синтаксичних залежностей між словами.

Google Cloud Natural Language API працює за заснованою на хмарних технологіях, тобто ви можете використовувати його як хмарний сервіс, не турбуючись про інфраструктуру. Для користування API вам необхідно мати обліковий запис Google Cloud та налаштувати доступ до API.

OpenAI API:

OpenAI API - це інтерфейс програмування застосунків, який надає доступ до потужних моделей штучного інтелекту, створених OpenAI. На момент мого останнього оновлення в січні 2023 року основною моделлю був GPT-3.5-turbo. Ось деякі ключові риси OpenAI API:

Мовні можливості: OpenAI API надає широкі можливості обробки природної мови (Natural Language Processing, NLP). Ви можете використовувати його для створення тексту, відповідей на питання, генерації статей, перекладу тексту та багато іншого.

Генерація тексту: Моделі OpenAI, такі як GPT-3.5-turbo, здатні генерувати людино-подібний текст, враховуючи контекст та запитання. Вони можуть використовуватися для створення контенту, допомоги у написанні тексту або навіть для інтерактивних чат-ботів.

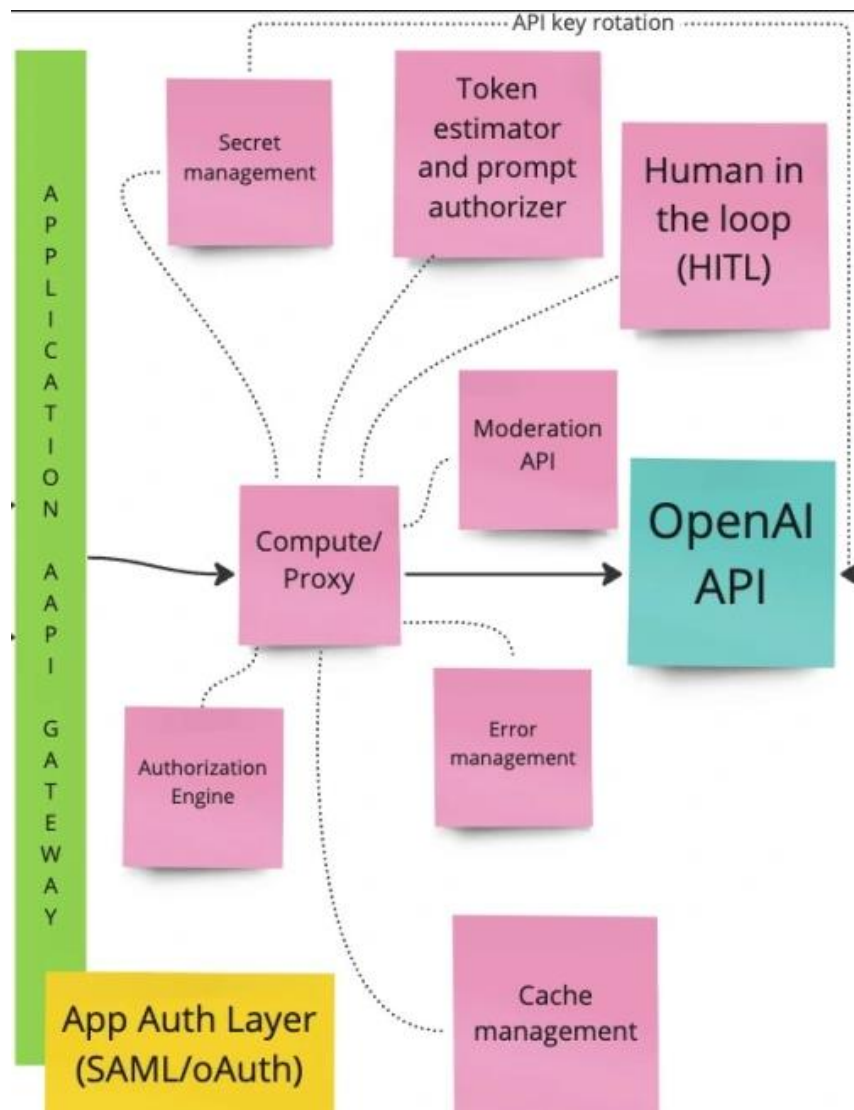


Рисунок 2.3 – Архітектура OpenAI API

Динамічні інструкції: Ви можете взаємодіяти з моделями, подаючи їм динамічні інструкції та контекст. Це дозволяє вам специфікувати, як модель повинна обробляти вхідні дані.

Ізні задачі: API може бути використано для різних задач, від вирішення завдань природної мови до генерації творчого контенту.

Запити в форматі тексту: Вам просто потрібно подати текстовий запит, і модель поверне відповідь у вигляді тексту.

Витрати ресурсів: API рахується за кожен токен у запиті, і це впливає на вартість використання. Токен - це фрагмент тексту, який може бути від одного символу до одного слова в деяких випадках.

Документація та інструменти: OpenAI надає докладну документацію та інструменти для взаємодії з API, включаючи приклади коду на різних мовах програмування.

Перед використанням OpenAI API слід ознайомитися з офіційною документацією, щоб краще зрозуміти можливості та обмеження.

Порівняльний огляд між Microsoft Azure Cognitive Services, Google Cloud Natural Language API і OpenAI API:

1. Мовні можливості:
 - Azure Cognitive Services: Потужний набір інструментів для обробки природної мови, включаючи розпізнавання мови, переклад тексту, аналіз тексту та інше.
 - Google Cloud Natural Language API: Забезпечує можливості аналізу настрою тексту, виокремлення іменованих сутностей, розпізнавання мови та інше.
 - OpenAI API: Спрямований на генерацію та розуміння тексту, включаючи відповіді на запитання, створення контенту та інші мовні завдання.
2. Типи завдань:
 - Azure Cognitive Services: Покриває широкий спектр завдань, таких як розпізнавання обличчя, підтримка мови, переклад тексту, аналіз тексту та інші.
 - Google Cloud Natural Language API: Спеціалізується на аналізі тексту для виявлення настрою, іменованих сутностей та іншого.
 - OpenAI API: Фокусується на генерації тексту та обробці мовленнєвих завдань.
3. Моделі глибокого навчання:
 - Azure Cognitive Services: Використовує глибокі нейронні мережі для різних завдань, таких як розпізнавання обличчя та мови.
 - Google Cloud Natural Language API: Засновано на передових алгоритмах глибокого навчання для аналізу тексту.
 - OpenAI API: Використовує потужні генеративні моделі, такі як GPT (Generative Pre-trained Transformer).
4. Цінова політика:
 - Azure Cognitive Services і Google Cloud Natural Language API: Розцінюється на основі використання ресурсів, таких як кількість оброблених запитів або обсяг тексту.
 - OpenAI API: Розцінюється за кожен токен (слово або символ) у вхідних і вихідних даних.
5. Інтеграція:
 - Azure Cognitive Services: Інтегрується з іншими сервісами Microsoft Azure та сторонніми продуктами.
 - Google Cloud Natural Language API: Пов'язано з іншими

сервісами Google Cloud та іншими додатками. - OpenAI API: Може використовуватися у широкому спектрі аплікацій через API взаємодії.

Проаналізувавши ці кілька API для розробки агентів штучного інтелекту, я вибрав OpenAI API у зв'язку з тим, що ця API має за собою навчену генеративну систему.

3. ДОСЛІДЖЕННЯ ТЕХНОЛОГІЇ ОБРОБКИ ПРИРОДНОЇ МОВИ

3.1. Поняття обробки природної мови

Обробка природної мови (Natural Language Processing, NLP) представляє собою галузь штучного інтелекту, що зосереджена на вивченні методів аналізу та синтезу природної мови. NLP представляє собою область інформаційних технологій, яка знаходиться на стику машинного навчання та математичної лінгвістики. Алгоритми обробки природної мови діють за аналогією з обробкою мови людини та складаються із трьох процесів: сприйняття, розуміння сенсу та реакції.

Процес сприйняття – це перетворення сенсорного сигналу в символний вигляд. Для систем NLP це означає перетворення вхідної інформації на машинозрозумілий набір символів. Сучасні нейронні мережі успішно впораються з цим завданням.

Реакція – це результат прийняття рішення. Це досить просте завдання, що вимагає формування можливих відповідей на основі сенсу сприйнятої фрази та контексту. Завдання реагування вирішується шляхом зважування альтернатив і порівняння результатів.

Розуміння сенсу – це найскладніше завдання, навіть для людей. Незнання контексту та неправильна інтерпретація фрази можуть призводити до непорозумінь чи конфліктів. У сучасній обробці мови використовуються такі методи аналізу: - Статистичний; - Формально-граматичний; - Нейромережевий.

Статистичний метод широко використовується у послугах машинного перекладу, автоматичних рецензентах та деяких чат-ботах. Цей метод полягає в "навчанні" моделі на великій кількості текстових даних, в яких встановлені статистичні закономірності. Потім ці моделі використовуються для перекладу текстів або генерації нових текстів з розумінням контексту.

Формально-граматичний підхід представляє собою математичний інструмент, який призначений для точного та однозначного визначення сенсу

фрази в природній мові настільки, наскільки це можливо для машин. Проте, не завжди це вдається реалізувати, оскільки сенс деяких фраз навіть для людей може залишатися нерозумним. Для розвинених мов, таких як українська, російська чи англійська, точний і детальний математичний опис мови є дуже складною задачею, тому формально-граматичний підхід майже не застосовується для аналізу природних розвинених мов, а частіше використовується для синтаксичного аналізу штучних мов.

У нейромережевому підході до розпізнавання сенсу вхідної фрази та генерації реакції системи штучного інтелекту використовуються нейронні мережі глибокого навчання. Вони навчаються на парах стимул-реакція, де стимулом є фраза природною мовою, а реакцією – відповідь системи на неї або будь-які дії системи. Цей підхід є найбільш перспективним, але він має всі негативні характеристики нейронних мереж. Однією з головних проблем нейронних мереж є потрібна кількість даних для їхнього навчання, що може вимагати доступу до потужного та дорогого комп'ютерного обладнання з високопродуктивними графічними процесорами. Крім того, нейромережі можуть бути використані у зловмисницьких цілях, таких як підробка голосу.

Для виконання завдань у процесі обробки природної мови застосовуються дві основні техніки: синтаксичний та семантичний аналіз. Синтаксичний аналіз пов'язаний з правильним розташуванням слів для забезпечення граматичної точності та створення зв'язного речення без помилок. Синтаксичні техніки включають лематизацію, морфологічну сегментацію, сегментацію слів, позначення (тегування) частин мови, синтаксичний розбір (парсинг), розрив речення та стеммінг.

При аналізі семантики акцент розміщується на контекстуальному значенні слів. Цей метод представляє собою найбільш складну частину природного навчання, яка на даний момент знаходиться в етапі розробки. Семантичні методи включають: - Розпізнавання сутностей за іменами – ідентифікація та категоризація слів за певними групами, такими як імена людей

або місця. - Визначення значень слів – додавання контекстного значення до слова на основі структури речення. - Генерація природної мови – використання бази даних для розшифрування логічного значення слова та перетворення зібраної інформації на людську мову.

3.2 Глибоке навчання у NLP

Значна частина технологій обробки природної мови (NLP) базується на глибокому навчанні, що є розділом машинного навчання, що став популярним лише на початку 2010-х років з наступних причин: - Накопичено значні обсяги навчальних даних. - Розвинуті обчислювальні потужності, такі як багатоядерні процесори та графічні прискорювачі. - Створено нові моделі та алгоритми з розширеними можливостями та підвищеною продуктивністю, з гнучким навчанням на проміжних уявленнях. - Розроблені методи навчання з використанням контексту, нові методи регуляризації та оптимізації.

У глибокому навчанні алгоритм намагається автоматично отримати найкращі ознаки чи подання із сирих вхідних даних. Ручне створення ознак часто є надто спеціалізованим, неповним та вимагає великого зусилля на їх створення та перевірку. На відміну від цього, глибоке навчання дозволяє легко адаптувати виявлені ознаки.

Глибоке навчання надає гнучкий, універсальний та ефективний фреймворк для подання інформації як візуальної, так і лінгвістичної. Спочатку це призвело до проривів у розпізнаванні мови та комп'ютерному зорі.

У традиційному NLP слова сприймаються як дискретні символи, які потім представляються як вектори. Проблема полягає в тому, що для дискретних символів слів немає чіткого поняття схожості між векторами. Тому альтернативою є навчання кодувати схожість у вектори.

Векторне представлення - це метод представлення слів у вигляді векторів зі значеннями. Для кожного слова створюється щільний вектор так, щоб слова, які з'являються в схожих контекстах, мали схожі вектори. Векторне представлення вважається стартовою точкою для більшості завдань NLP і

робить глибоке навчання ефективним при обробці невеликих обсягів даних. Техніки векторних представлень, такі як Word2vec і GloVe, розроблені компанією Google та університетом Stanford, користуються популярністю і широко застосовуються в задачах NLP. Word2vec використовує великий корпус тексту для створення вектора для кожного слова в словнику. Алгоритм порівнює вектори центрального слова та контекстного слова, щоб розрахувати ймовірність появи одного слова при заданому іншому, і піддається налаштуванню для максимізації цієї ймовірності.

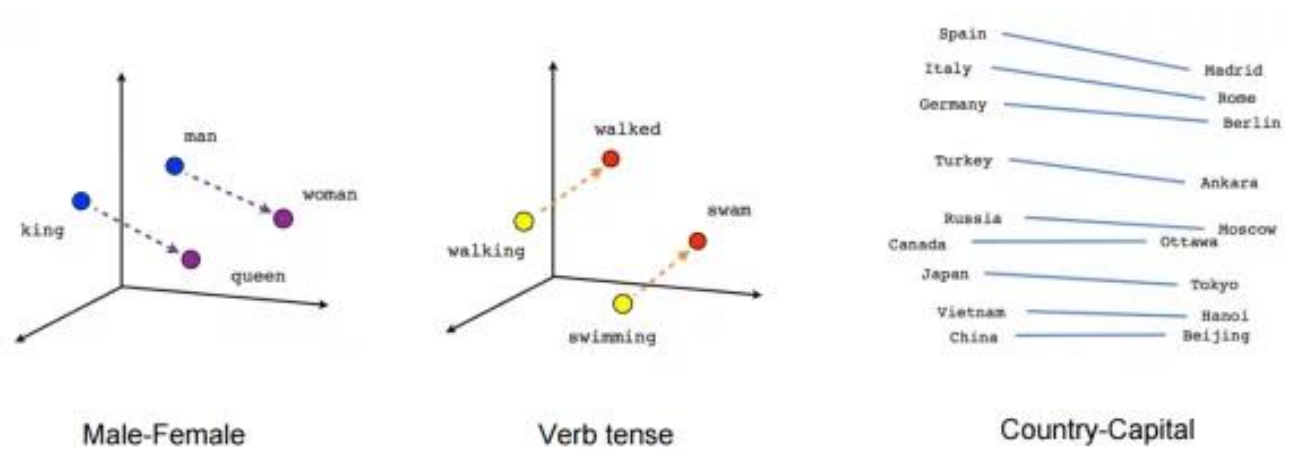


Рисунок 3.1 – Регулювання вектор слів для максимізації

Для досягнення найкращого результату з Word2vec на наборі даних проводиться видалення зайвих слів (наприклад, слів з високою частотою вживання, таких як сполучники). Це покращує точність моделі та скорочує час тренування. Також застосовується негативна вибірка для кожного введення, оновлюючи ваги для всіх правильних міток, але лише для обмеженої кількості некоректних міток.

GloVe створено для вирішення цієї проблеми, захоплюючи значення одного векторного представлення із структури всього корпусу, доступного для огляду. Для цього модель виявляє глобальні відповідності між числом слів та мінімізує середньоквадратичне відхилення. Такий підхід дозволяє визначати схожість слова за векторною відстанню.

3.3 Застосування технології обробки природної мови

На сьогоднішній день технологія обробки природної мови (NLP) використовується для вирішення різноманітних завдань у різних галузях: - пошук за допомогою письмового або усного вводу; - показ релевантної онлайн-реклами; - автоматичний переклад текстів; - фільтрація спаму; - виправлення граматичних та контекстуальних помилок у тексті; - аналіз настроїв для маркетингових завдань; - передбачення наступних слів в контексті; - розпізнавання мови; - розробка голосових та текстових помічників.

Машинний переклад - це процес перетворення тексту однієї природної мови на еквівалентний йому текст іншою мовою. Цей процес виконується програмою чи машиною без участі людини. У машинному перекладі використовується статистика використання слів в контексті. Системи машинного перекладу знаходять широке комерційне застосування, оскільки галузь перекладів мов світу має обсяг бізнесу на рівні \$40 мільярдів щорічно.

У традиційних системах машинного перекладу використовується паралельний корпус - набір текстів, кожен з яких перекладено однією чи кількома іншими мовами. Наприклад, при перекладі з вихідної мови українською (u) на цільову англійською (e), потрібно створити статистичну модель, що включає ймовірне формулювання для правила Байєса, модель перекладу $p(u|e)$, навчену на паралельному корпусі, і мовну модель $p(e)$, яка навчена лише на корпусі з англійською мовою.

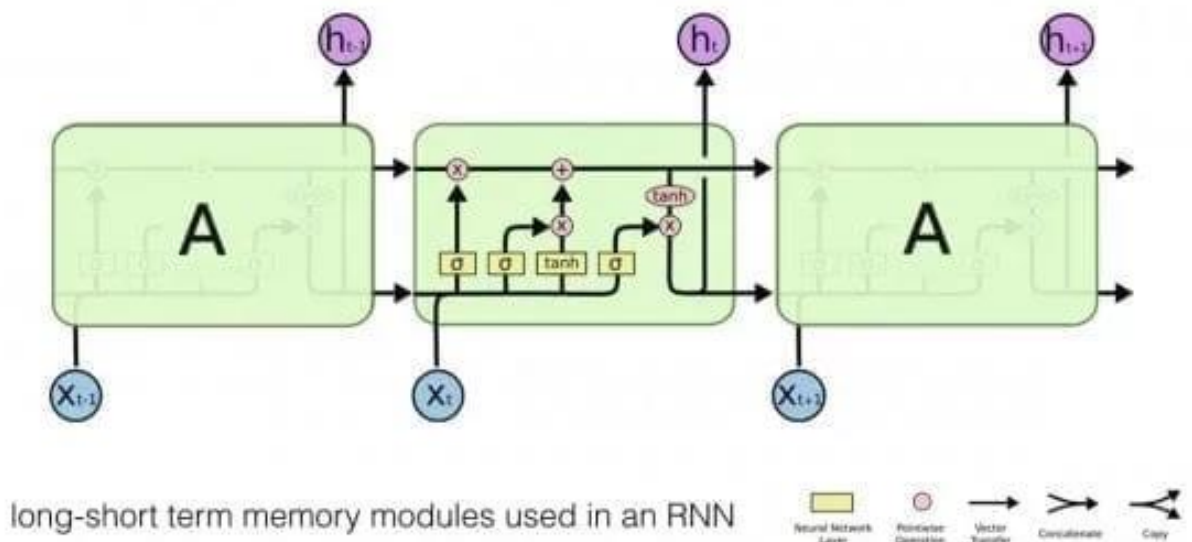
Зрозуміло, що такий підхід не враховує безліч важливих деталей, що вимагають великої кількості ручно розроблених ознак і включають в себе різні та незалежні задачі машинного навчання.

Нейромережевий машинний переклад - це підхід до створення моделей перекладу з використанням рекурентної нейронної мережі (RNN). RNN - це нейронна мережа, яка залежить від попередніх станів і має зв'язки між різними проходами. Нейрони отримують інформацію від попередніх шарів і від самих себе на попередньому кроці. Це означає, що порядок подачі даних і тренування мережі має важливе значення. Стандартна модель нейромережевого машинного

перекладу використовує наскрізну нейромережу, де RNN-кодер кодує вихідне речення, а RNN-декодер передбачає цільове слово. Кодер "читає" вихідне речення і об'єднує його в останньому прихованому шарі, а декодер використовує зворотне поширення помилки для навчання та повертає переклад. Цей метод машинного перекладу став стандартом через два роки з моменту початку досліджень.

Основна проблема RNN - це проблема зникнення градієнта, коли інформація втрачається з часом. Це може призвести до труднощів у запам'ятовуванні слів, які стоять далеко від початку послідовності, і до того, що передбачення здійснюються на основі крайніх слів. Мережі короткостроково-довгострокової пам'яті (LSTM) намагаються вирішити цю проблему, введенням ґрат і комірок пам'яті. Кожен нейрон у LSTMs представляє собою клітинку пам'яті з трьома ґратами: на вхід, на вихід та на забування. Ці ґрати виконують функцію охоронців для інформації, дозволяючи або забороняючи її потік.

Long-Short Term Memory module: LSTM



long-short term memory modules used in an RNN

<http://colah.github.io/posts/2015-08-Understanding-LSTMs/> Espenio Casalicchio © 2018

Рисунок 3.2 – Модуль мережі короткостроково-довгострокової пам'яті

Вхідний ґейт визначає, скільки інформації з попереднього шару буде зберігатися в даному вузлі, тоді як вихідний ґейт керує тим, яка частина наступного шару отримає інформацію про поточний стан комірки. Ґейт забуття

контролює, наскільки довго зберігається значення у пам'яті, подібно до того, як неймережі можуть вирішувати, коли необхідно забути деякі слова з попереднього розділу при вивченні нового.

Модель довготривалої пам'яті (LSTM) продемонструвала успішність у навчанні на складних послідовностях, таких як написання у стилі конкретного автора або створення простої музики. LSTM широко використовується в машинному перекладі і є стандартною моделлю для завдань маркування послідовностей з великою кількістю даних.

Застосування технології NLP включає системи "Питання-відповідь" (QA), які витягують інформацію безпосередньо з різних джерел для задоволення потреб користувача. QA системи надають короткі та лаконічні відповіді замість вимагання від користувача читати повний текст. Ці системи легко поєднуються з інтерактивними помічниками, а також можуть взаємодіяти з іншими джерелами інформації, такими як набори зображень.

Багато завдань NLP можуть бути розглянуті як завдання "питання-відповідь", де машина надає відповідь на запитання, аналізуючи текст чи інструкції. Для таких завдань розроблено потужну архітектуру глибокого навчання — Мережу Динамічної Пам'яті (Dynamic Memory Network, DMN). Ця архітектура використовує епізодичні спогади, що формуються на основі вхідних даних та питань, для генерації відповідей із високою точністю.

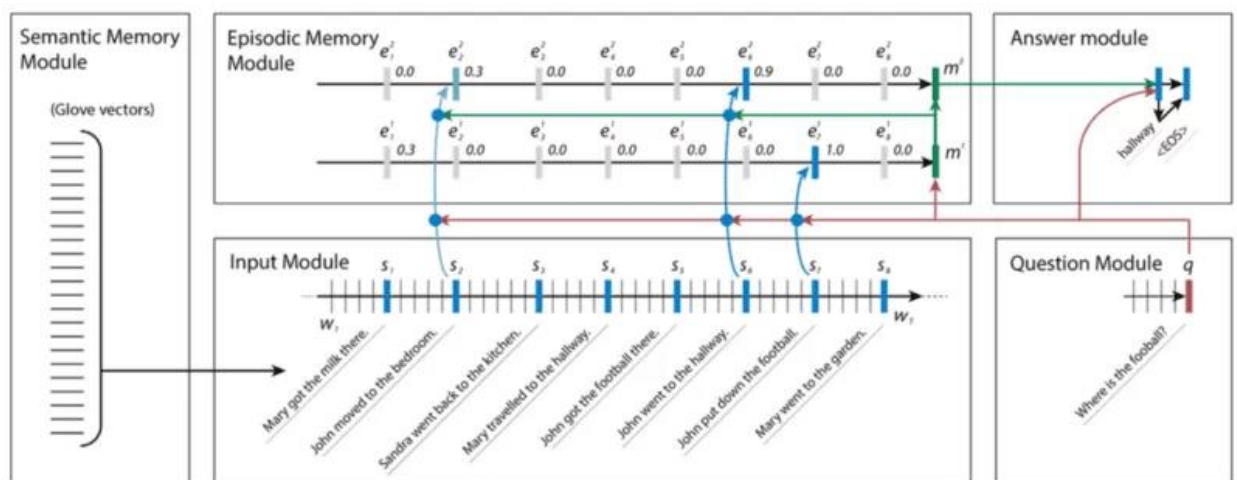


Illustration of DMN performing transitive inference.

Рисунок 3.3 – Архітектура мережі динамічної пам'яті

Модуль семантичної пам'яті, подібний до бази знань, складається з попередньо підготовлених векторів, які використовуються для утворення послідовностей векторних уявлень слів зі вхідних речень. Ці вектори функціонують як вхідні дані для моделі.

Вхідний модуль обробляє пов'язані з питанням вхідні вектори на набір векторів, відомих як факти. Для цього використовується блок керованого рекурентного блоку (Gated Recurrent Unit, GRU), що дозволяє мережі визначати релевантність розглядуваної пропозиції.

Модуль питань обробляє запитання слово за словом і генерує вектор за допомогою того самого GRU, що і вхідний модуль, із тими ж вагами.

Модуль епізодичної пам'яті зберігає витягнуті вектори фактів та питань у вигляді вкладень.

Модуль відповідей генерує відповідь, використовуючи інший GRU, навчений з класифікацією крос-ентропійної помилки правильної послідовності, яка потім конвертується назад у природну мову.

Ще одна сфера застосування технології NLP – створення точних та лаконічних резюме вхідного документа. Вилучення короткого змісту є важливим інструментом для сприяння інтерпретації текстової інформації. Запитання та резюме статей завдають все більше інтересу, і кількість завдань щодо створення розумних та точних резюме для обширних фрагментів тексту постійно зростає.

Механізм автоматичного вилучення короткого змісту з тексту функціонує наступним чином: спочатку обчислюється частота появи кожного слова в повному текстовому документі, після чого зберігається та сортується сто найчастіших слів. Далі кожне речення оцінюється за кількістю вживаних слів, при цьому вага приділяється словам, які найчастіше зустрічаються. За результатом, перші X пропозицій сортуються з урахуванням їхнього положення в оригінальному тексті.

Алгоритм автоматичного вилучення короткого змісту, завдяки своїй простоті та узагальнюючій здатності, ефективний навіть у випадках складних сценаріїв. Наприклад, він успішно працює з текстами іноземними мовами або з унікальними словниковими асоціаціями, що відсутні в стандартних текстових масивах.

Існують два основних підходи до скорочення тексту: витягувальний та абстрактний. В першому випадку відбираються фрази або речення з оригінального тексту для формування резюме. В абстрактному підході досліджується внутрішнє мовне уявлення для створення адекватного викладу, перефразовуючи оригінальний текст.

Методи витягувального скорочення працюють на основі вибору підмножини, отримуючи фрази або речення з тексту для формування резюме. Моделі для абстрактного скорочення використовують глибоке навчання, забезпечуючи значні досягнення. Деякі приклади відомих компаній у цьому напрямку включають Facebook Neural Attention, Google Sequence-to-sequence та IBM Watson.

При аналізі тексту важливо враховувати, що природна мова включає в себе не лише слова та їхнє чітке значення, але й нюанси та складнощі. Технологія обробки природної мови (NLP) також орієнтується на аналіз настроїв, спираючись на вибір слів та пунктуацію для визначення емоційного стану співбесідника чи інші нюанси в тексті.

Стандартний підхід до аналізу настроїв передбачає розгляд речень як мішків слів та оцінку списків "позитивних" і "негативних" слів для визначення настрою кожного конкретного речення. Цей метод вимагає використання вручну створених функцій для фіксації настрою, і відзначається великим часовим затратами і важкістю масштабування.

Сучасний підхід глибокого навчання до аналізу настроїв використовується для врахування морфології, синтаксису та логічної семантики, і серед них найефективнішою є рекурсивна нейронна мережа.

Основним припущенням застосування рекурсивної нейронної мережі є те, що рекурсія є природним засобом опису мови. Використання рекурсії корисне для роз'яснення неоднозначностей, ефективно в завданнях, які включають посилення на конкретні фрази, і особливо успішне в завданнях, де використовується структура граматичного дерева. Рекурсивні нейронні мережі ідеально підходять для аналізу текстів, які мають вкладену ієрархію та внутрішню рекурсивну структуру.

3.4 Інструменти для розробки помічника з використанням технології NLP

Існують спеціальні інструменти для створення діалогових агентів з технологією NLP. Серед таких платформ – Visual Studio, Jupyter Notebook, PyCharm, VS Code.

Jupyter Notebook – це інтерактивне середовище розробки та обміну документами, в яких можна вбудовувати живий код, рівняння, візуалізації та текст. Вона надає зручний спосіб створювати та ділитися документами, дозволяючи об'єднувати описовий текст, код та результати його виконання у єдиний файл. Jupyter Notebook широко використовується в галузі аналізу даних, машинного навчання, наукових досліджень та освіти. Він підтримує кілька мов програмування, включаючи Python, що робить його привабливим вибором для розробки та прототипування проектів, включаючи створення чат-ботів.

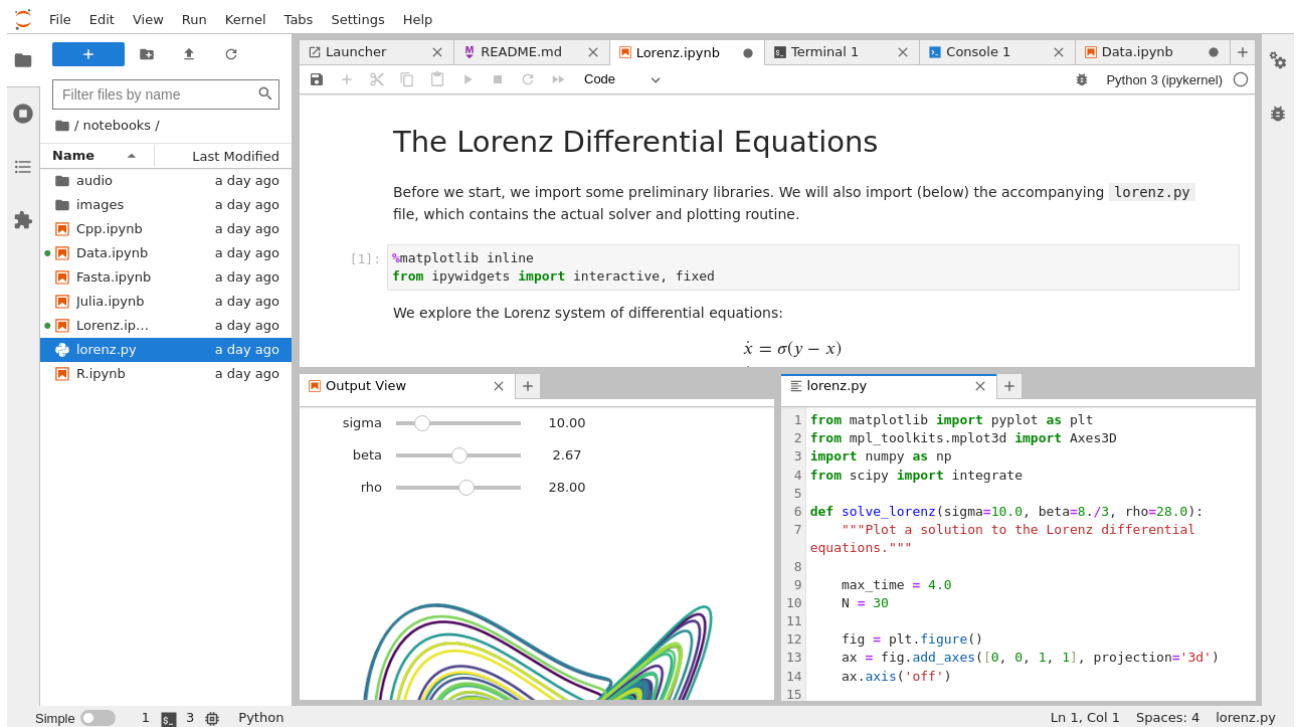


Рисунок 3.4 – Інтерфейс Jupyter Notebook

PyCharm - це інтегроване середовище розробки (IDE) від JetBrains, спеціально створене для мови програмування Python. Це середовище надає розробникам зручні інструменти та функціональність, які сприяють ефективній роботі над проектами на Python. Основні характеристики PyCharm: Інтелектуальне підсвічування коду: PyCharm надає потужний механізм підсвічування коду, який полегшує читання та написання коду. Автоматичне доповнення: IDE надає автоматичне доповнення коду, що суттєво прискорює процес написання коду та знижує кількість помилок. Налаштування: PyCharm включає відладник, який дозволяє проводити покрокове виконання коду, встановлювати точки зупинки і аналізувати значення змінних. Управління залежностями: Середовище надає інструменти для управління залежностями проекту, що робить процес розробки зручнішим. Інтеграція із системами керування версіями: PyCharm інтегрується з популярними системами керування версіями, такими як Git, що спрощує роботу в команді. Підтримка віртуальних середовищ: Розробники можуть створювати віртуальні середовища для своїх проектів, ізолюючи залежності та забезпечуючи портативність. Аналіз коду: PyCharm здійснює статичний аналіз коду, виявляючи потенційні

проблеми та застерігаючи від помилок. Підтримка різних фреймворків: Середовище розробки забезпечує інтеграцію з різними фреймворками Python, що є актуальним для створення чат-ботів.

PyCharm надає інструменти, необхідні для розробки програм Python, включаючи ті, які можуть бути використані при створенні чат-ботів.

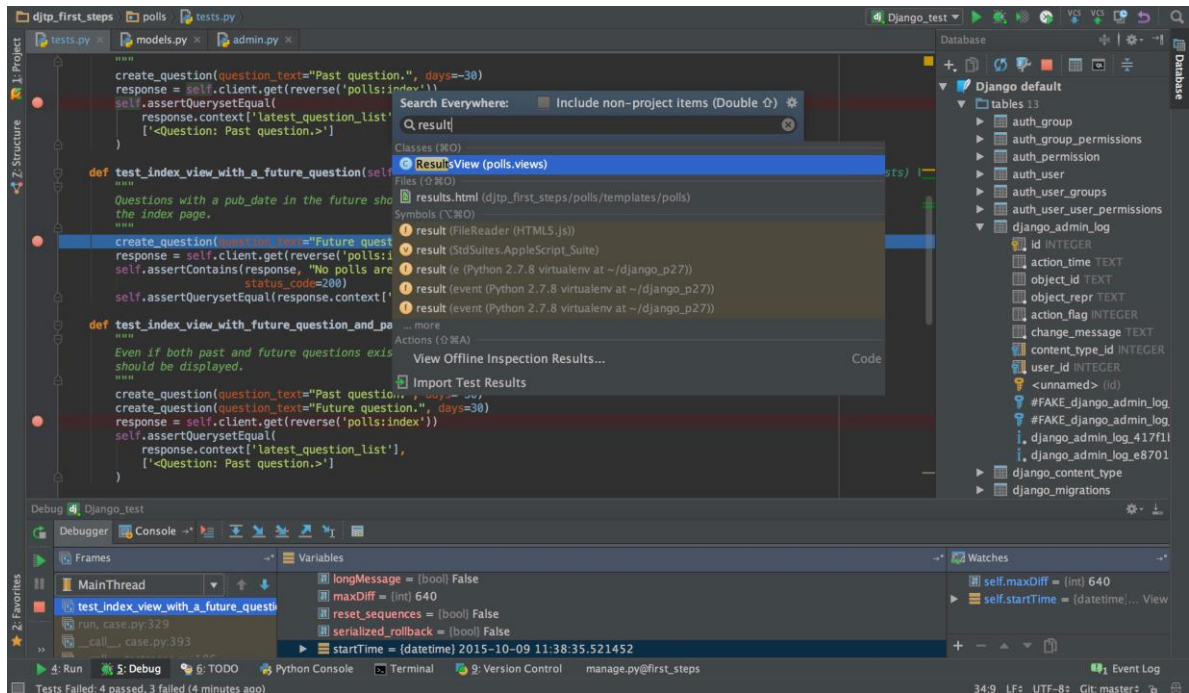


Рисунок 3.5 – Інтерфейс PyCharm

Visual Studio Code (VS Code): Легкий, потужний редактор коду від Microsoft, що надає широкий набір інструментів для розробки різних програм. VS Code орієнтований на простоту використання, швидкий запуск та розширюваність.

Однією з ключових особливостей VS Code є його легкість та висока продуктивність. Він підходить для розробки на різних мовах програмування, включаючи Python, що робить його чудовим вибором для створення чат-ботів. Редактор підтримує автоматичне завершення коду, інтегровані системи керування версіями та багатий набір розширень.

VS Code також має широку базу користувацьких розширень, створених спільнотою розробників. Це дозволяє адаптувати середовище під свої потреби та інтегрувати додаткові інструменти.

Зі зручним інтерфейсом та можливістю запуску на різних платформах (Windows, macOS, Linux), VS Code надає відмінне оточення для розробки чат-ботів, особливо якщо важливі легкість використання та широкі можливості налаштування.

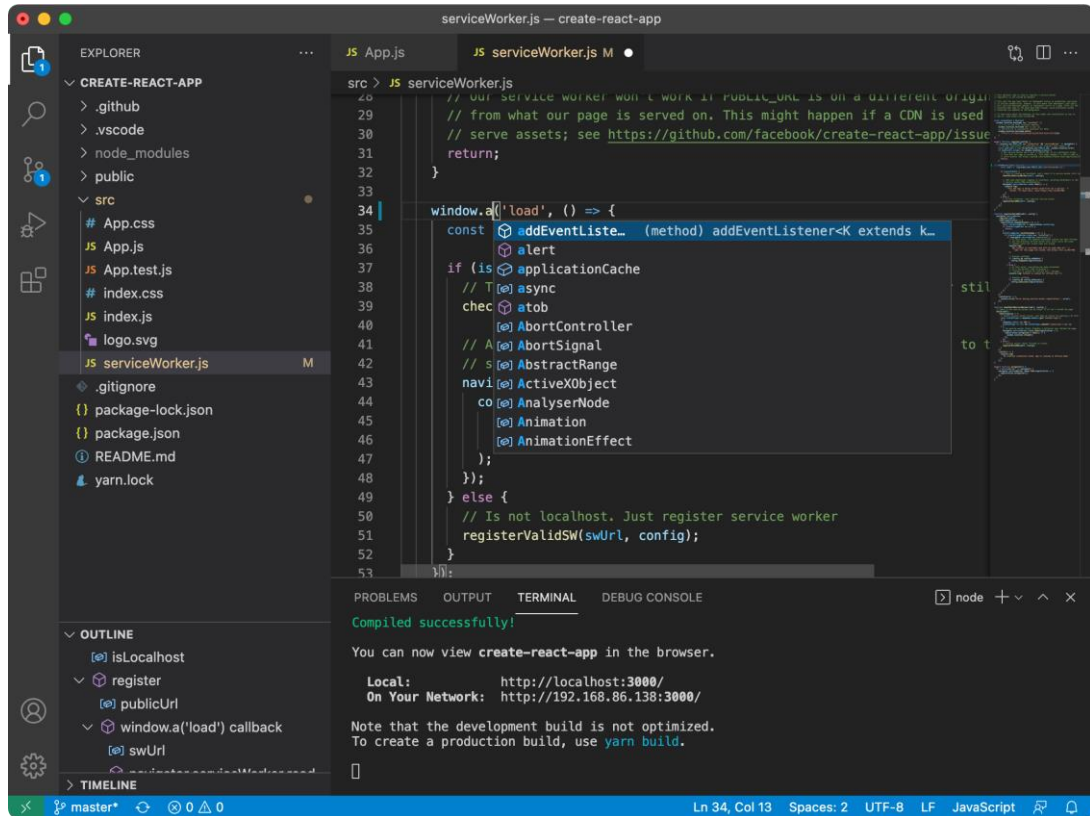


Рисунок 3.6 – Інтерфейс VS Code

Visual Studio – це інтегроване середовище розробки (IDE), що надає великі інструменти та ресурси для створення, налагодження та розгортання різноманітних програм. Розроблена компанією Microsoft Visual Studio підтримує безліч мов програмування, включаючи Python, що робить її популярним вибором для розробників. Основні характеристики Visual Studio включають: Багатозадачність: Visual Studio забезпечує зручне управління проектами, кодом та ресурсами, дозволяючи розробникам ефективно працювати над кількома аспектами проекту. Інтеграція з Git: Підтримка системи контролю версій Git вбудована безпосередньо в середу розробки, забезпечуючи зручне відстеження змін та спільну роботу в команді. Налагодження та профілювання: Вбудовані інструменти налагодження та профілювання дозволяють виявляти та усувати помилки, підвищуючи якість

коду. Великі бібліотеки та плагіни: Різноманітні бібліотеки та розширення полегшують інтеграцію сторонніх інструментів та покращують процес розробки. Підтримка Python: Visual Studio забезпечує підтримку Python та інструменти для розробки програм цією мовою, включаючи створення чат-ботів. Спільна технологія: Інструменти для спільної роботи над проектами дозволяють розробникам ефективно обмінюватися кодом та ідеями в реальному часі.

Загальна гнучкість та багатофункціональність Visual Studio роблять її привабливим вибором для розробників, включаючи тих, хто працює зі штучним інтелектом та створенням чат-ботів.

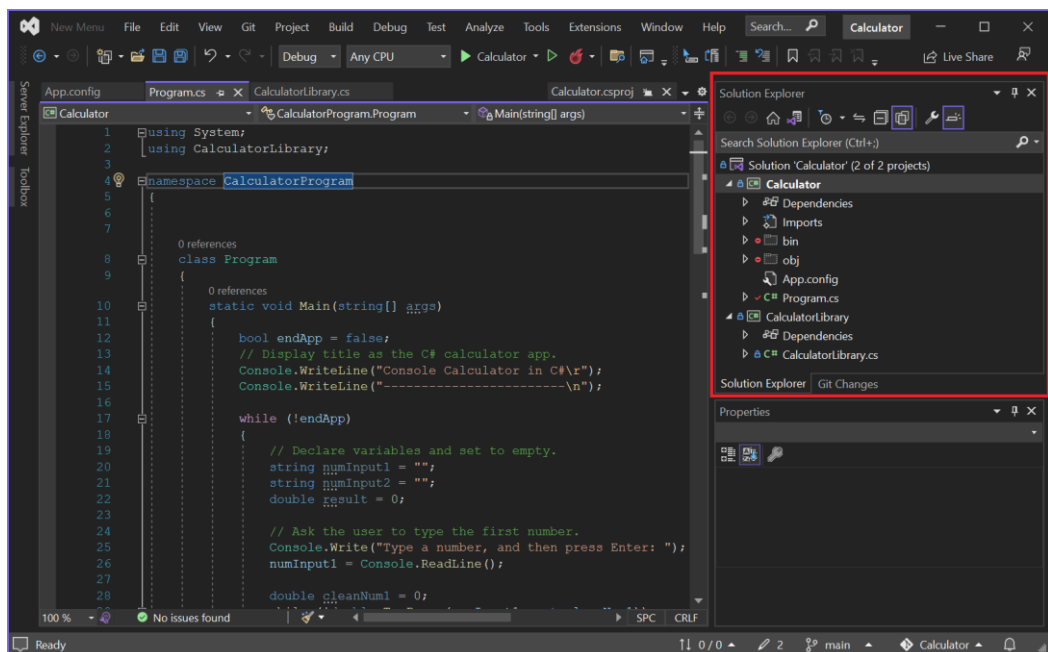


Рисунок 3.7 – Інтерфейс Visual Studio

Керуючись великими можливостями та зручністю використання Visual Studio, а також її інтеграцією з мовою програмування Python, я вирішив вибрати цю середу розробки для створення діалогового агента. Сучасні інструменти, підтримка безлічі мов, ефективні засоби налагодження та розгортання роблять Visual Studio оптимальним вибором для роботи над проектом, пов'язаним із штучним інтелектом та діалоговими агентами.

3.5 Вибір мови програмування для розробки агента штучного інтелекту

Мови надають різноманітні інструменти та ресурси для розробки та реалізації нейронних мереж, спеціалізованих на обробці природної мови. Вони активно використовуються в індустрії та науковому співтоваристві для вирішення завдань, пов'язаних з NLP. Серед найпоширеніших мов програмування, що використовуються для розробки нейронних мереж, відзначають Python, Java і JavaScript.

Java

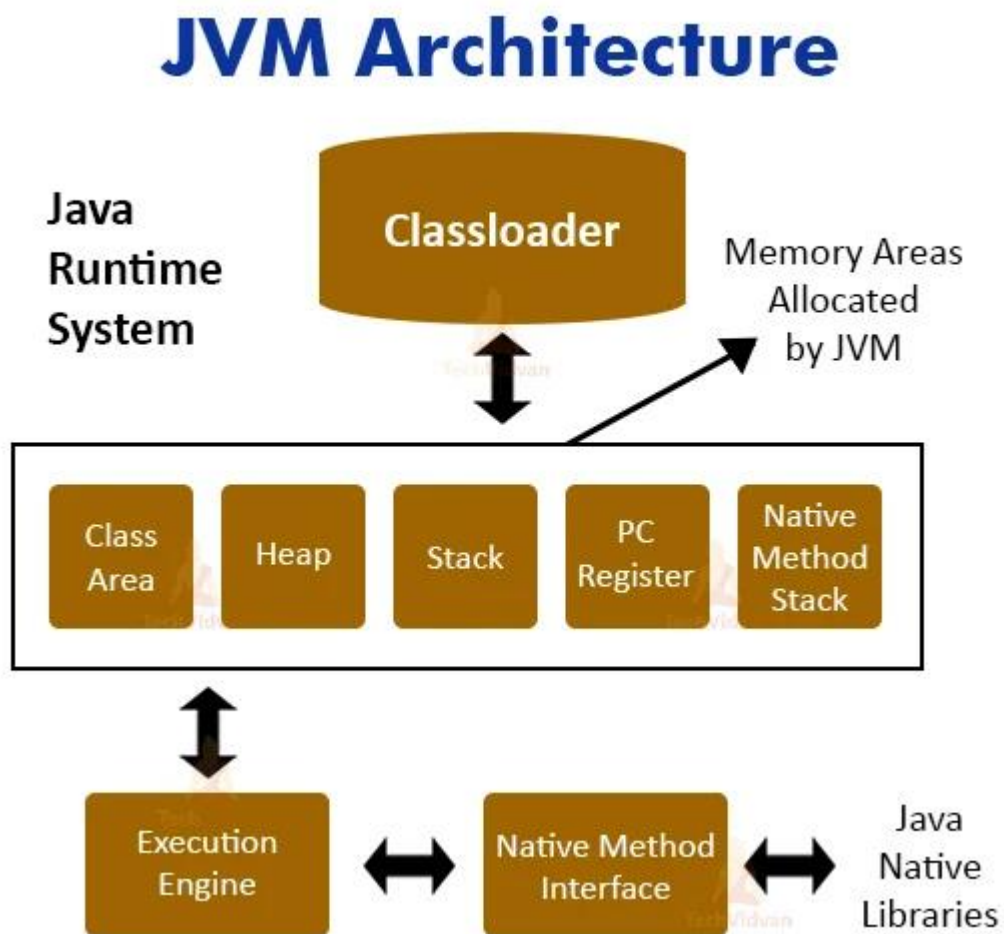


Рисунок 3.8 – Архітектура Java Virtual Machine

Java є потужною та універсальною мовою програмування, яка також знаходить своє застосування в галузі обробки природної мови (NLP). Ось кілька аспектів використання Java у вирішенні завдань NLP:

Apache OpenNLP: Однією з ключових бібліотек обробки природної мови на платформі Java є Apache OpenNLP. OpenNLP надає набір інструментів для обробки тексту природною мовою, включаючи такі завдання, як поділ на

пропозиції, токенізація, виділення іменованих сутностей, розбір пропозицій та багато іншого. Це дозволяє розробникам легко впроваджувати NLP-функціональність у свої Java-програми.

Deeplearning4j: Deeplearning4j (DL4J) - це бібліотека глибокого навчання для Java, яка дозволяє створювати та навчати нейронні мережі. DL4J підтримує роботу з NLP-завданнями та надає можливість створювати та навчати моделі для обробки природної мови. Завдяки підтримці Apache UIMA (Unstructured Information Management Architecture), DL4J добре інтегрується з рішеннями NLP на основі Java.

Stanford NLP: Stanford NLP – це ще одна потужна бібліотека для обробки природної мови на Java. Вона надає реалізації різних NLP-задач, таких як аналіз настрою, розмітка частин мови, аналіз залежностей та багато іншого. Бібліотека також включає передбачені моделі, що полегшує використання її для вирішення конкретних завдань. **Gate NLP Library:** GATE (General Architecture for Text Engineering) – це інструментарій та бібліотека для обробки та аналізу тексту. Вона написана на Java та надає широкий набір інструментів для роботи з текстовими даними. GATE також включає компоненти для NLP, такі як аналізатори, токенізатори і засоби для вилучення інформації.

Робота з бібліотеками Machine Learning: Java також використовується для інтеграції з бібліотеками машинного навчання, які можуть застосовуватись до завдань NLP. Наприклад, розробники можуть використовувати бібліотеку Weka для вирішення завдань класифікації або Apache Mahout для обробки великих обсягів даних та рекомендацій.

Java забезпечує широкі можливості для розробки NLP-рішень завдяки своєму великому екосистемному інструментарію та міцній платформі для побудови складних програм обробки природної мови.

JavaScript

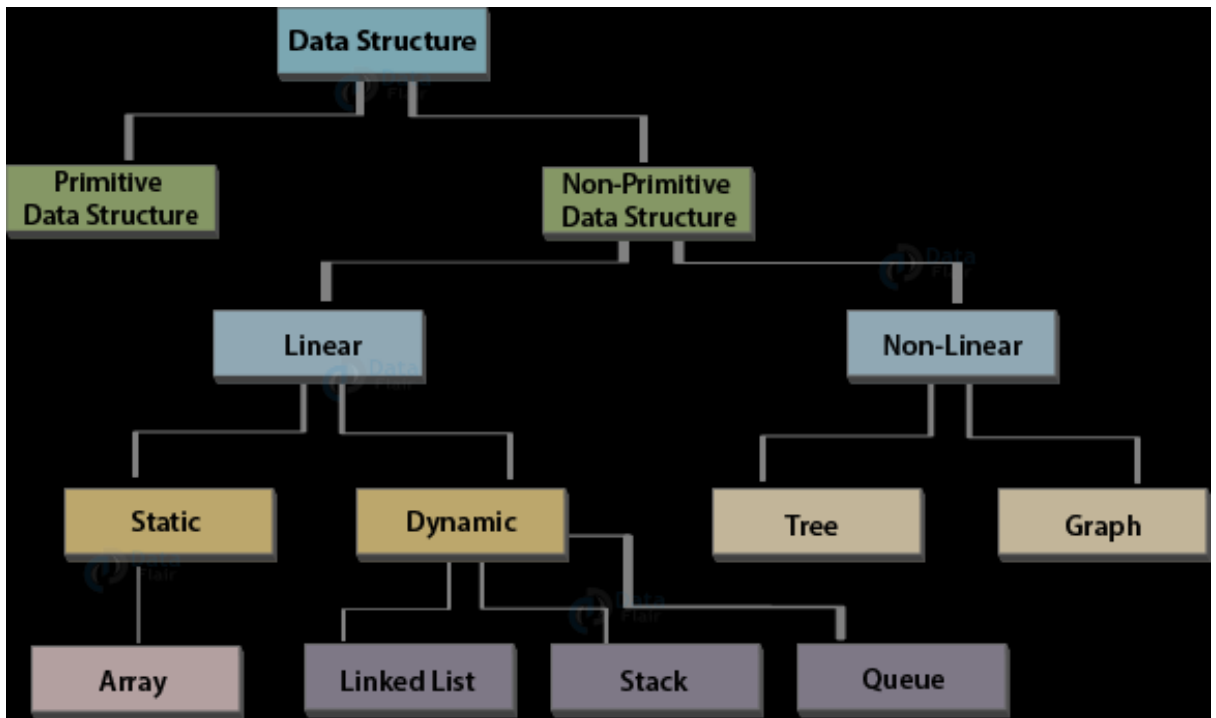


Рисунок 3.9 – Структура JavaScript

JavaScript, що широко використовується у веб-розробці, також знаходить своє місце у вирішенні завдань, пов'язаних з обробкою природної мови (NLP). Ось кілька аспектів використання JavaScript у цьому контексті:

TensorFlow.js: TensorFlow.js надає JavaScript-інтерфейс для створення та навчання нейронних мереж безпосередньо у браузері. Це дозволяє розробникам вирішувати завдання NLP, такі як класифікація тексту, обробка мови та машинний переклад, використовуючи сучасні технології глибокого навчання прямо на клієнтській стороні веб-додатків.

Natural: Бібліотека Natural є реалізацією алгоритмів обробки природної мови JavaScript. Вона включає функціональності, такі як токенізація, стеммінг, визначення частин мови та інші. Natural дозволяє розробникам проводити базовий аналіз тексту та вирішувати деякі NLP-завдання.

Compromise: Compromise - це ще одна бібліотека для обробки природної мови JavaScript. Вона забезпечує можливість аналізу та маніпулювання текстовими даними. Compromise підтримує безліч функцій, таких як вилучення сутностей, аналіз настрою, визначення мови та інші.

Node.js для серверної частини: Node.js дозволяє використовувати JavaScript на сервері, що дає можливість реалізації NLP-задач у бекенді веб-додатків. Це особливо корисно для обробки великих обсягів даних або виконання складних обчислень в області NLP. Різні бібліотеки, такі як Natural та Compromise, можуть бути легко інтегровані до програм, що працюють на Node.js.

Brain.js: Brain.js - це бібліотека машинного навчання JavaScript, призначена для браузерного та серверного середовища. Вона може використовуватися для створення та навчання простих нейронних мереж та застосовується до різних завдань, включаючи NLP. Brain.js дозволяє створювати та навчати моделі нейронних мереж для вирішення завдань, таких як класифікація тексту або прогнозування.

Stanford JavaScript NLP: Stanford JavaScript NLP – це порт деяких NLP-інструментів від університету Стенфорда на JavaScript. Він включає інструменти для аналізу частин мови, аналізу залежностей та інші, що збагачує функціональність JavaScript в обробці природної мови.

JavaScript, завдяки своїй широкій підтримці у веб-розробці та можливості використання як на клієнтській, так і на серверній стороні, стає все більш популярним інструментом для вирішення завдань, пов'язаних із NLP.

Python

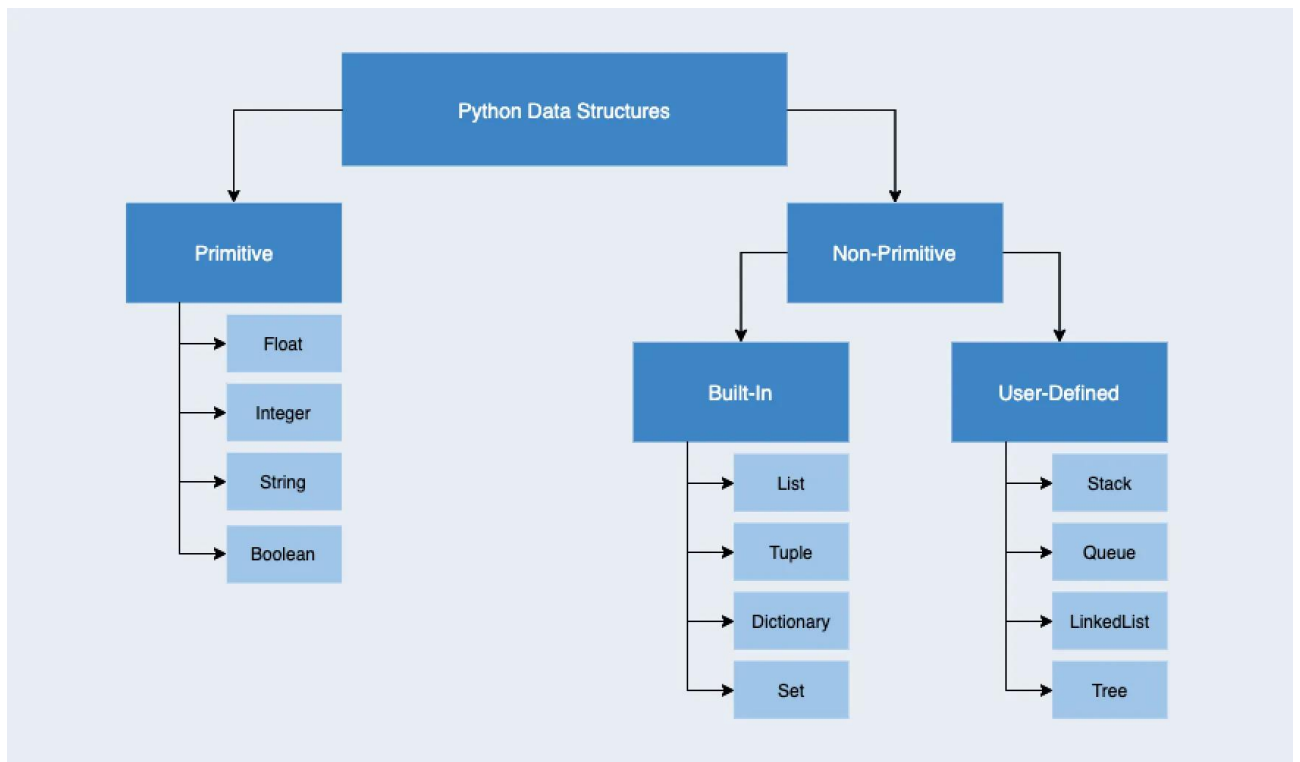


Рисунок 3.10 – Структура даних Python

Python є однією з найпопулярніших мов програмування в галузі обробки природної мови (NLP) завдяки безлічі бібліотек та фреймворків, розроблених для спрощення роботи з текстовими даними. Ось кілька ключових аспектів використання Python у NLP:

NLTK (Natural Language Toolkit): NLTK - це потужна бібліотека для роботи з природною мовою Python. Вона надає засоби для токенізації, стеммінгу, лематизації, аналізу частин мови, отримання інформації, обробки текстових даних та багато іншого. NLTK також включає в себе корпуси та ресурси для навчання та тестування моделей NLP.

SpaCy: SpaCy – це високопродуктивна бібліотека для обробки природної мови. Вона призначена для виконання завдань, таких як вилучення іменованих сутностей, аналіз залежностей та обробка текстових даних. SpaCy має оптимізоване ядро на мові C, що робить його одним з найшвидших інструментів для NLP на Python.

Gensim: Gensim – це бібліотека для роботи з тематичним моделюванням та векторним поданням тексту. Вона широко використовується для створення векторних уявлень слів (Word Embeddings) та тематичного моделювання, що

корисно у завданнях, пов'язаних із обробкою природної мови, таких як кластеризація текстів або аналіз схожості документів.

TensorFlow та PyTorch: TensorFlow і PyTorch - два популярні фреймворки глибокого навчання, які надають високорівневі API для створення та навчання нейронних мереж. Вони широко використовуються для вирішення NLP-завдань, таких як машинний переклад, аналіз настрою, та отримання інформації. Існує безліч передвиборних моделей та інструментів для обробки текстових даних.

Transformers (Hugging Face): Transformers – це бібліотека від Hugging Face, що надає безліч передвиборних моделей для різних NLP-завдань. Вона заснована на фреймворку PyTorch та TensorFlow. Transformers включає GPT (Generative Pre-trained Transformer), BERT (Bidirectional Encoder Representations from Transformers) та інші передові архітектури.

Stanford NLP: Бібліотека Stanford NLP надає реалізації багатьох NLP-завдань, таких як аналіз залежностей, аналіз настрою, іменовані сутності та інші. Ця бібліотека розроблена мовою Java, але має обгортки для використання на Python.

Python, завдяки своїй простоті, багатій екосистемі та великій спільноті розробників, залишається однією з кращих мов програмування для вирішення завдань NLP. З його допомогою розробники можуть ефективно створювати та розробляти моделі, обробляти та аналізувати текстові дані.

Java, JavaScript і Python - всі три мови програмування активно використовуються у створенні агентів нейронних мереж, але кожен з них має свої особливості та переваги у контексті розробки та застосування нейронних мереж. Давайте розглянемо їх порівняння: Java
Переваги: - Платформенна
Незалежність: Java є платформно-незалежною мовою, що означає, що код, написаний на Java, може виконуватися на різних платформах без змін. -
Великі Обчислювальні Завдання: Java підходить для обробки великих обсягів даних і виконання обчислювально складних завдань, що може бути корисним

при створенні складних агентів нейронних мереж. Недоліки: - Швидкість Виконання: Java зазвичай менш продуктивна в порівнянні з деякими іншими мовами, такими як C++ або Python. JavaScript Переваги: - Інтеграція з Веб-Додатками: JavaScript широко використовується у веб-розробці, і фреймворк TensorFlow.js дозволяє створювати та навчати моделі нейронних мереж безпосередньо у браузері. - Асинхронність: Асинхронна природа JavaScript може бути корисною для обробки великої кількості запитів у розподілених системах. Недоліки: - Обмежені можливості обчислень: В деяких випадках JavaScript може виявитися менш ефективним для виконання складних обчислювальних завдань у порівнянні з мовами, такими як Python або Java. Python Переваги: - Велика Екосистема: Python має велику бібліотеку та фреймворки для машинного навчання та глибокого навчання, такі як TensorFlow, PyTorch, Keras, scikit-learn та ін. - Простота Вивчення та Використання: Python має простий і зрозумілий синтаксис, що спрощує створення та навчання моделей нейронних мереж. - Спільнота та Документація: Python має велику та активну спільноту розробників, а також велику документацію, що полегшує процес навчання та вирішення проблем. Недоліки: - Швидкість Виконання: Python може бути не таким швидким, як деякі компіювані мови, що може бути важливим при виконанні великих обчислювальних завдань. Зведення: - Якщо потрібна платформна незалежність і ви більше дбаєте про промислове програмування, Java може бути хорошим вибором. - JavaScript відмінно підходить для веб-додатків та розподілених систем, але його обчислювальні можливості можуть бути обмежені. - Python широко використовується в галузі машинного навчання та глибокого навчання, надаючи великі інструменти та ресурси для створення та навчання агентів нейронних мереж.

Проаналізувавши всі особливості, переваги та недоліки трьох мов програмування, я вибрав Python для створення діалогового агента.

4. ПРОЕКТУВАННЯ ДІАЛОГОВОГО АГЕНТА

Після проведення аналізу предметної області, агентів-аналогів та вибору засобів розробки, необхідним етапом є проектування інформаційної системи. На цьому етапі було створено UML-діаграми варіантів використання (прецедентів), активності та послідовностей. Всі діаграми були створені в об'єктно-орієнтованому CASE-засобі проектування інформаційних веб-систем Visual Paradigm. Visual Paradigm є інтегрованим засобом проектування архітектури, аналізу, моделювання та розробки ІС.

4.1 Проектування варіантів використання системи

Основна ідея діаграми прецедентів полягає у представленні системи як групи акторів, які взаємодіють із нею через різні варіанти використання. Актор – це сутність, яка взаємодіє із системою для вирішення певних завдань. Актором може бути особа, інша система, пристрій чи програмний засіб. У цьому контексті акторами є користувач діалогового агента, адміністратор та сам агент, що обробляє запити користувача. На діаграмі також зображено сховище інформації.

Після ідентифікації акторів системи необхідно скласти перелік усіх варіантів використання, з якими будуть взаємодіяти визначені актори. Серед ключових варіантів використання системи від користувача входять отримання інформації з певної теми, інформації з попередніх діалогів, аналіз документів, отримання загальних даних із наявних ресурсів та узагальнення інформації на основі даних. З боку адміністратора включається додавання, зміна та видалення інформації.

На основі сформованих варіантів використання та ідентифікованих акторів була розроблена схема використання варіантів.

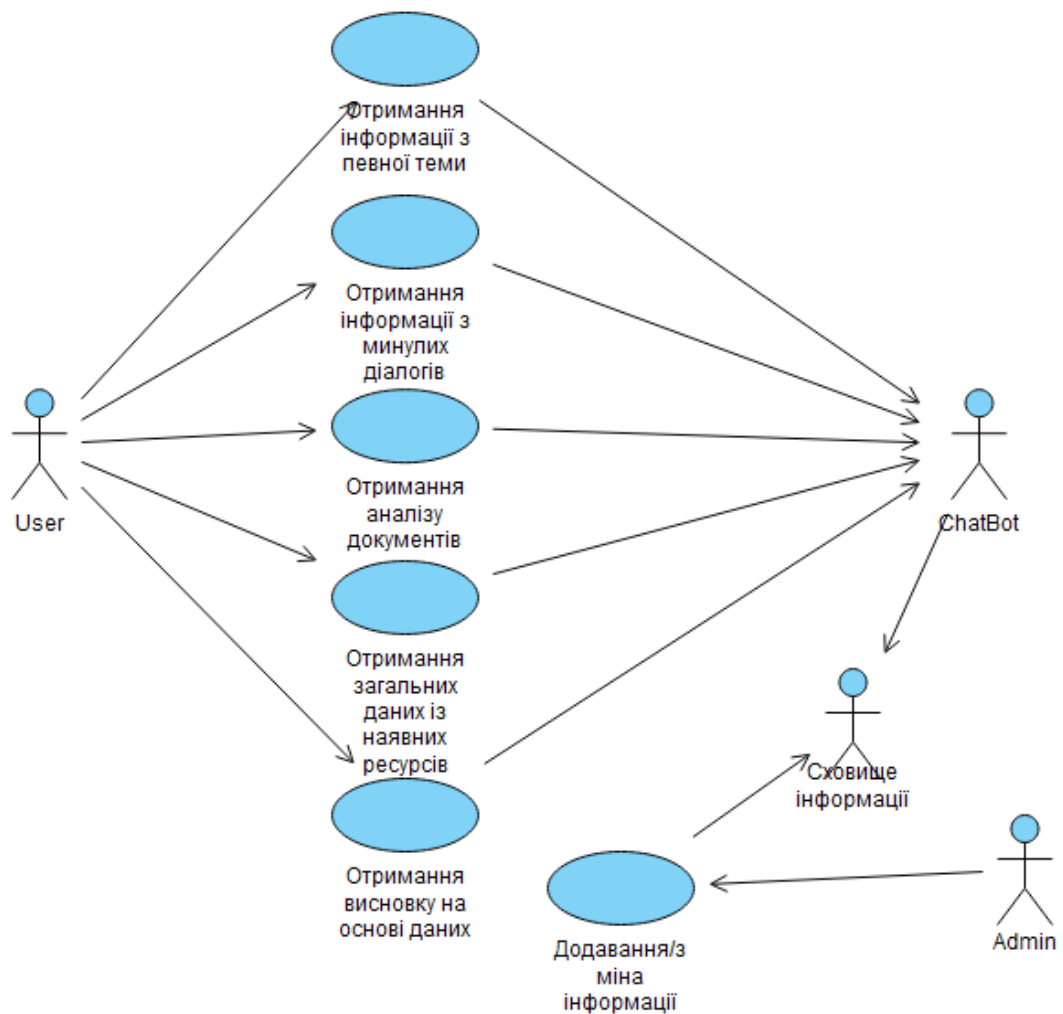


Рисунок 4.1 – Діаграма прецедентів

4.2 Створення діаграми діяльності

Для моделювання виконання операцій у мові UML використовуються діаграми діяльності, також відомі як діаграми активностей. На цих діаграмах відображається логіка чи послідовність переходу від однієї діяльності до іншої, при цьому увага зосереджується на результаті виконаної діяльності. Сам результат може вплинути на стан системи або вивести певне значення. Діаграма діяльності відтворює візуально ряд дій чи потік керування в системі, що подібно блок-схемі чи діаграмі потоку даних. Такі діаграми можуть описувати кроки в сценарії використання, де модельовані дії можуть відбуватися послідовно або одночасно. Незалежно від цього, діаграма діяльності має початок (початковий стан) та кінець (кінцевий стан).

Була створена діаграма діяльності для системи, що відображає основний процес проектування системи – введення користувачем запиту та його обробка агентом.

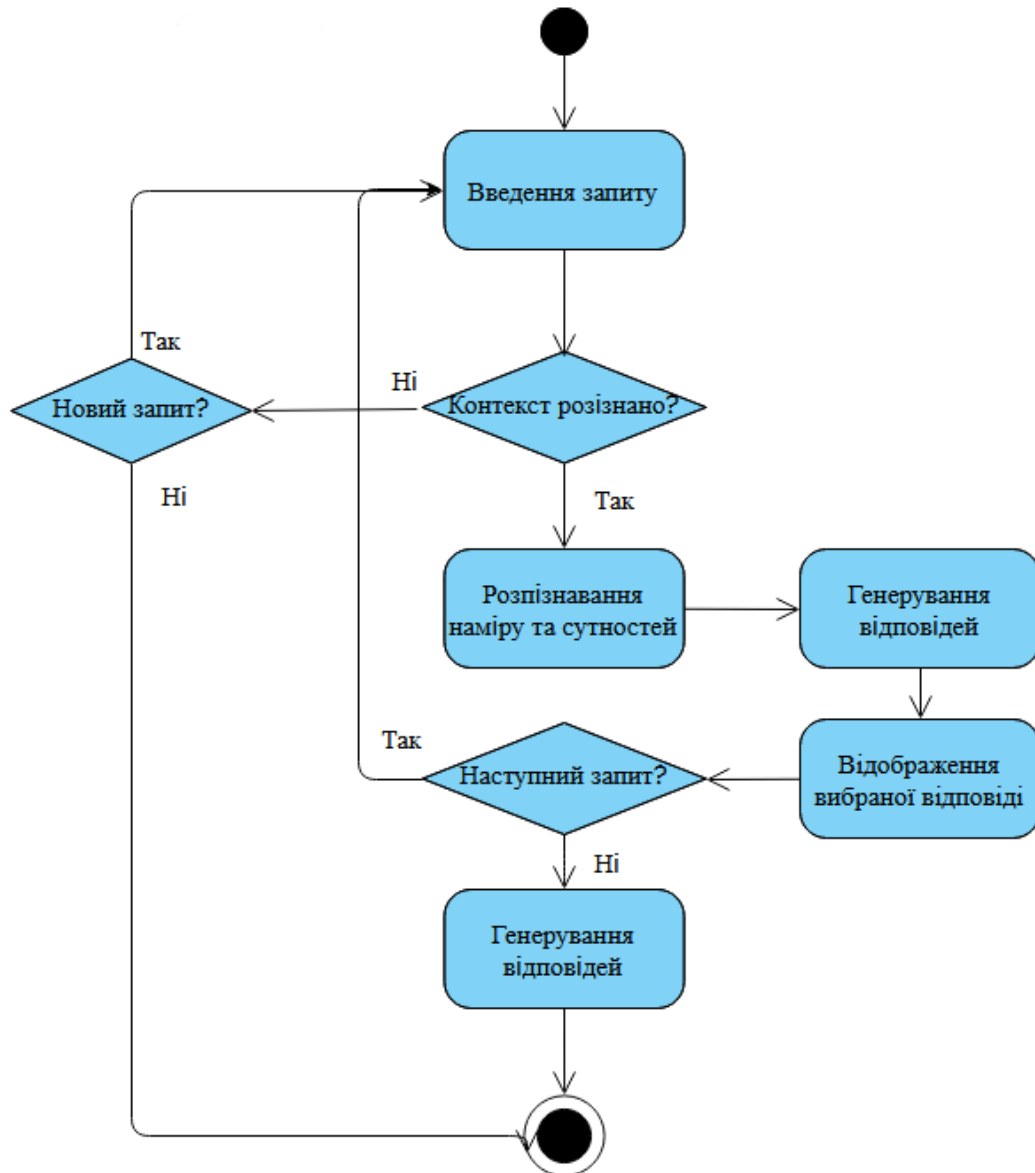


Рисунок 4.2 – Діаграма діяльності

4.3 Проектування логіки сценаріїв використання

Для реалізації логіки сценаріїв використання мого діалогового агента використовується інтеграція з OpenAI API та внутрішньою логікою обробки запитань та відповідей. Процес взаємодії з агентом можна подати у вигляді послідовності кроків: 1) Введення запитання від користувача: - Користувач вводить запитання природною мовою через інтерфейс терміналу Visual Studio. 2) Взаємодія з OpenAI API: - Запитання надсилається до OpenAI API для

отримання текстової відповіді. - OpenAI модель обробляє запитання і генерує відповідь на основі навчання. 3) Повернення відповіді користувачеві: - Відповідь від OpenAI API отримується та виводиться в термінал Visual Studio. 4) Збереження історії чату: - Кожна взаємодія додається до історії чату, що дозволяє агенту зберігати та використовувати попередні діалоги.

Цей процес створює ефективний механізм обробки запитань та надання відповідей, який використовує потужності OpenAI для створення текстового контенту.

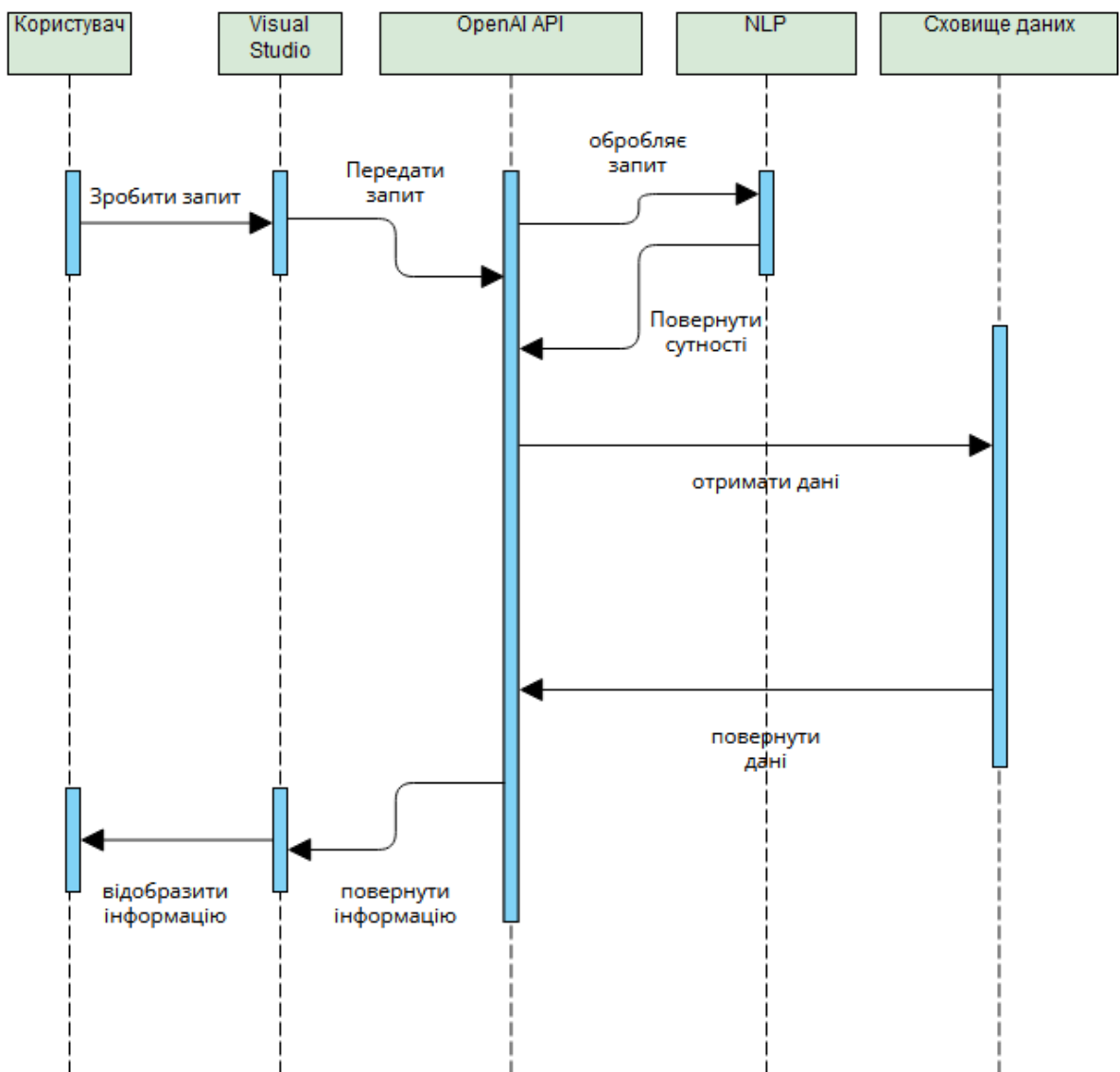


Рисунок 4.3 – Діаграма послідовностей

Таким чином, на етапі проектування системи були створені UML-

діаграми прецедентів, активності і послідовності, які допомогли зрозуміти

основні варіанти використання системи та логіку роботи агента.

5. ПРОГРАМНА РЕАЛІЗАЦІЯ

5.1. Створення агента на основі OpenAI

При створенні агентів на основі OpenAI API важливо враховувати кілька ключових особливостей.

Використання мовленнєвих моделей в рамках OpenAI API є ключовим елементом для розробки діалогових агентів. Основною моделлю, яка використовується, є GPT (Generative Pre-trained Transformer) - мовленнєва модель, створена за допомогою трансформерної архітектури. Мовленнєві моделі, такі як GPT, спочатку піддаються пре-тренуванню на великому обсязі текстових даних. Цей етап дозволяє моделі отримати загальне розуміння мови та лінгвістичні здібності. Після пре-тренування модель може бути тюнірована на конкретні завдання або сценарії, що стосуються вашого діалогового агента. Це дозволяє моделі ліпше розуміти контекст та природну мову, яка використовується користувачем. Мовленнєві моделі можуть ефективно розуміти вхідний текст та генерувати природну мову відповідно до контексту. Вони здатні враховувати різні варіанти запитань та надавати логічні та змістовні відповіді. Завдяки здатності зберігати чатову історію та враховувати контекст, мовленнєві моделі дозволяють створювати довгі та послідовні діалоги. Вони можуть легко узгоджувати відповіді з попередніми фразами та створювати змістовні розмови. Мовленнєві моделі, особливо трансформери, проявляють велику креативність у відповідях. Вони можуть генерувати тексти різноманітної структури та стилю, що дозволяє створювати цікаві та природні діалоги.

Запитання та контекст грають ключову роль у взаємодії з мовленнєвими моделями, такими як GPT, через OpenAI API. Можна ставити різноманітні та специфічні запитання, використовуючи різні структури та формулювання. Це дозволяє отримати різні типи відповідей та перевірити поведінку моделі у різних сценаріях. Запитання можна ставити як окремо, так і в контексті

попередніх фраз чатової історії. Це дозволяє управляти ходом діалогу та отримувати відповіді, враховуючи попередні обговорення. Мовленнєві моделі у OpenAI API здатні зберігати чатову історію, що дозволяє враховувати попередні фрази та надавати відповіді, спираючись на контекст. Збереження контексту дозволяє створювати діалоги як лінійні, так і нелінійні. Можна взаємодіяти з моделлю в багатьох різних сценаріях, побудованих на попередніх обговореннях. Збереження контексту дозволяє створювати діалоги як лінійні, так і нелінійні. Можна взаємодіяти з моделлю в багатьох різних сценаріях, побудованих на попередніх обговореннях. За допомогою контексту можна ефективно управляти темою розмови. Можна ставити додаткові запитання, спираючись на попередні відповіді, що робить діалог більш глибоким та змістовним. Запитання та контекст взаємодії забезпечують гнучкість та можливість створювати різноманітні та персоналізовані діалоги з допомогою мовленнєвих моделей.

Робота з відповідями API в контексті OpenAI API включає в себе взаємодію з системою для отримання та обробки відповідей на поставлені запитання. Запитання передаються через API у форматі JSON, що містить необхідні дані, такі як текст запитання та контекст чатової історії. Система використовує передані дані для взаємодії з мовленнєвою моделлю (наприклад, GPT) та генерації відповіді на запитання користувача. Отримані відповіді аналізуються та обробляються для визначення тексту відповіді та інших важливих атрибутів, таких як достовірність та стиль. У відповідях може враховуватися контекст чатової історії, що дозволяє моделі надавати згідно з попередніми обговореннями та ставити запитання на основі попередніх відповідей. API може підтримувати додаткові параметри, такі як температура генерації (що впливає на креативність відповіді) та кількість варіантів відповіді. Зазвичай, відповіді та їх контекст зберігаються для подальшого логування, аналізу та використання у подальших інтеракціях. Розробники можуть

вдосконалювати відповіді, експериментуючи з різними параметрами, такими як температура та максимальна кількість слів.

Валідація та поліпшення в контексті OpenAI API включає в себе процес визначення якості та доопрацювання відповідей моделі для забезпечення задоволення потреб користувачів. Проведення аналізу та перевірка згенерованих відповідей з використанням тестових запитань та реальних сценаріїв для визначення точності та адекватності відповідей. Залучення людей до тестування та оцінки відповідей для отримання об'єктивних відгуків та інсайтів щодо якості взаємодії. Використання отриманих результатів валідації для створення додаткових даних для тренування моделі та вдосконалення її відповідей. Поглиблений аналіз попередніх взаємодій, щоб виявити патерни, недоліки та можливості для оптимізації відповідей. Випробовування різних параметрів запитань та параметрів генерації для визначення оптимальних налаштувань для конкретного використання. Систематичне оновлення моделі з урахуванням виявлених слабких місць, нових даних та вимог користувачів для постійного покращення відповідей.

В цілому, реалізація даного проекту визначається комплексним підходом до створення та оптимізації діалогового агента з використанням передових технологій та інструментів, що дозволяє ефективно вирішувати завдання в узкоспеціалізованих галузях та забезпечує високу якість обслуговування користувачів.

5.2. Встановлення та Налаштування Оточення

У цьому розділі описується процес встановлення та налаштування необхідного оточення для розробки чатбота з використанням Visual Studio та мови програмування Python. 1) Встановлення Visual Studio

Встановлення та налаштування Visual Studio — це ключовий етап у створенні програмного забезпечення, включаючи чат-боти. Процес установки Visual Studio досить простий та інтуїтивний, і включає кілька основних етапів.

Перш за все, для початку роботи потрібно завантажити дистрибутив Visual Studio з офіційного веб-сайту компанії Microsoft. На сайті ви можете знайти різні версії Visual Studio, виберіть ту, яка вам підходить, наприклад, Visual Studio Community для особистого використання.

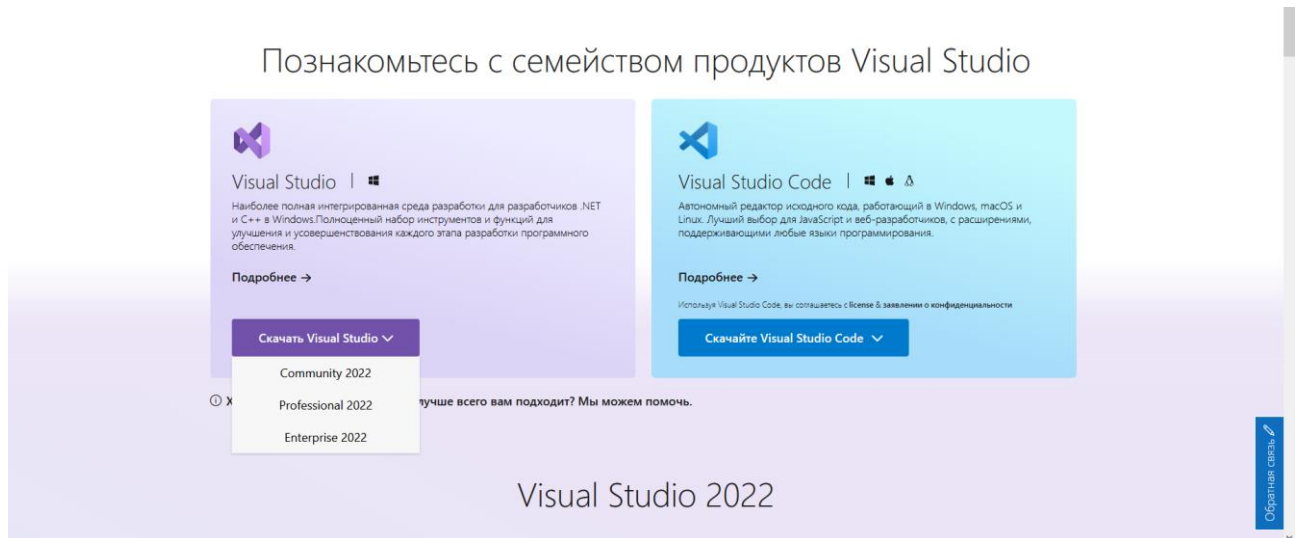


Рисунок 5.1 – Сайт Visual Studio Сайт з можливістю встановити їх IDE

Після завантаження виконайте файл інсталятора. Інсталятор пропонуватиме вам вибрати тип інсталяції. Основні компоненти включають інтегровану середовище розробки (IDE), різні інструменти розробки та необхідні бібліотеки.

Оберіть компоненти, які вам необхідні для вашого проекту. У вас може бути вибір мов програмування, інструментів для розробки веб-застосунків, мобільних додатків та інших опцій.

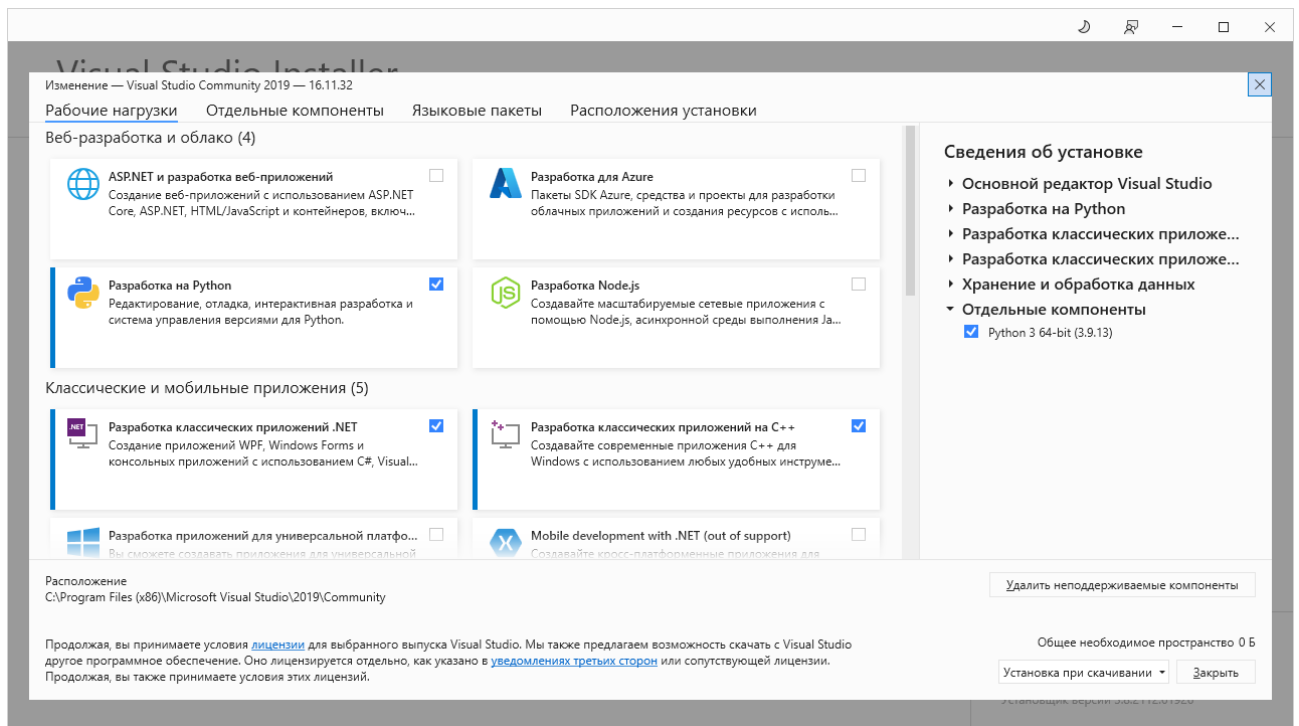


Рисунок 5.2 – Инсталлятор Visual Studio

Після цього оберіть місце для установки Visual Studio на вашому комп'ютері. Рекомендується обрати системний диск C: або інший диск з достатньою кількістю вільного місця.

Налаштування може також включати вибір опцій інтеграції з іншими інструментами розробки, налаштування вигляду IDE та інші параметри.

Після завершення установки запустіть Visual Studio. Під час першого запуску система може запропонувати вам вибрати тему оформлення та налаштування інших параметрів IDE.

Такий процес установки Visual Studio дозволить вам швидко почати розробку проекту. 2) Встановлення Python:

Встановлення Python на операційну систему Windows є простим та доступним процесом. Виконайте ці кроки для встановлення Python на свій комп'ютер:

Першим кроком є завантаження файлу інсталяції Python з офіційного веб-сайту Python. Відвідайте сайт та натисніть кнопку "Downloads". Потім виберіть останню версію Python для Windows та завантажте інсталлятор.

Після завершення завантаження відкрийте завантажений інсталятор Python (зазвичай це файл з розширенням .exe). Для встановлення рекомендується вибрати опцію "Add Python to PATH" під час встановлення, щоб додати Python до змінного середовища PATH і спростити доступ до інтерпретатора Python з командного рядка.

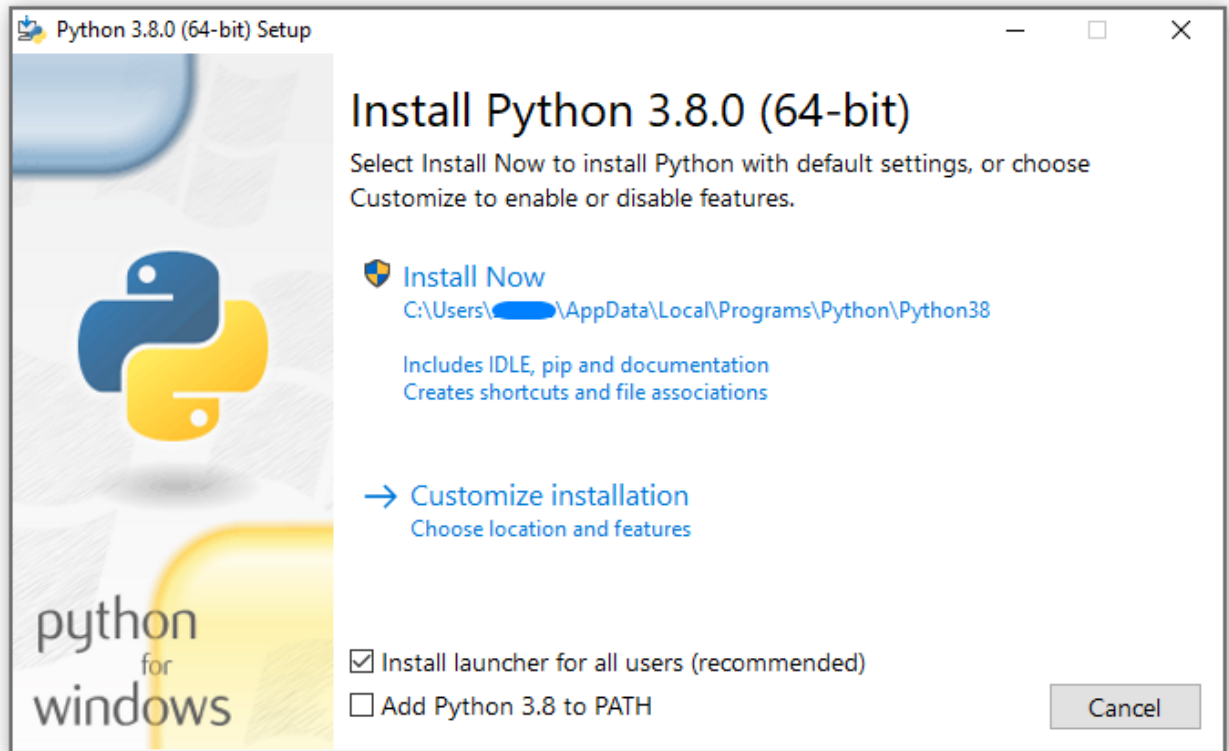


Рисунок 5.3 – Інсталятор Python

Якщо вибрати «Встановити зараз»:

Вам не потрібно бути адміністратором (якщо не потрібне оновлення системи для C Runtime Library або якщо ви не інстальєте програму запуску Python для Windows для всіх користувачів)

Python буде встановлено у ваш каталог користувача

Програму запуску Python для Windows буде встановлено відповідно до опції внизу першої сторінки

Буде встановлено стандартну бібліотеку, набір тестів, засіб запуску та pip

Якщо вибрано, каталог встановлення буде додано до вашого ШЛЯХУ

Ярлики будуть видимі лише для поточного користувача

Вибравши «Налаштувати інсталяцію», ви зможете вибрати функції для інсталяції, місце інсталяції та інші параметри або дії після інсталяції. Щоб установити символи налагодження або двійкові файли, вам потрібно буде скористатися цією опцією.

Щоб виконати інсталяцію для всіх користувачів, вам слід вибрати «Налаштувати інсталяцію». В цьому випадку:

Вам може знадобитися надати облікові дані адміністратора або схвалення.

Python буде встановлено в каталог Program Files.

Програму запуску Python для Windows буде встановлено в каталог Windows.

Під час встановлення можна вибрати додаткові функції.

Стандартна бібліотека може бути попередньо скомпільована в байт-код.

Якщо вибрано, каталог встановлення буде додано до системного ШЛЯХУ.

Ярлики доступні для всіх користувачів.

Історично у Windows довжина шляху обмежена 260 символами. Це означало, що шляхи, довші за цей, не вирішуватимуться, і спричинятимуться помилки.

В останніх версіях Windows це обмеження можна розширити приблизно до 32 000 символів. Вашому адміністратору потрібно буде активувати групову політику «Увімкнути довгі шляхи Win32» або встановити для LongPathsEnabled значення 1 у розділі реєстру HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem.

Це дозволяє функції open(), модулю os і більшості інших функцій шляху приймати та повертати шляхи, довші за 260 символів.

Після зміни наведеного вище параметра подальша конфігурація не потрібна.

Усі параметри, доступні в інтерфейсі користувача інсталятора, також можна вказати з командного рядка, що дозволяє інсталяторам зі сценарієм копіювати інсталяцію на багатьох машинах без участі користувача. Ці параметри також можна встановити без придушення інтерфейсу користувача, щоб змінити деякі параметри за замовчуванням.

У програму встановлення можна передати такі параметри (found by executing the installer with /?):

Ім'я	Опис
/passive	для відображення прогресу, не вимагаючи взаємодії з користувачем
/quiet	для встановлення/видалення без відображення інтерфейсу користувача
/simple	щоб запобігти налаштування користувача
/uninstall	видалити Python (без підтвердження)
/layout [directory]	щоб попередньо завантажити всі компоненти
/log [filename]	щоб вказати розташування файлів журналу

Усі інші параметри передаються як ім'я=значення, де значення зазвичай дорівнює 0, щоб вимкнути функцію, 1, щоб увімкнути функцію, або шлях.

Виконані дії встановили мову програмування Python на комп'ютер. Повернувшись у Visual Studio, необхідно завантажити в IDE компіляцію з Python.



Рисунок 5.4 – Категорія із встановленням Python

3) Встановлення бібліотек та залежностей:

Роль бібліотек і залежностей Visual Studio безсумнівно відіграє ключову роль у процесі розробки програмного забезпечення. Visual Studio, що надається Microsoft, є одним із найпотужніших і найпопулярніших інтегрованих середовищ розробки (IDE) серед розробників.

У Visual Studio бібліотеки та залежності відіграють вирішальну роль в управлінні зовнішніми ресурсами та компонентами, які використовуються у

проектах розробки. Ці елементи дозволяють розробникам включати у свої проекти готові компоненти, що прискорює розробку, підвищує повторне використання коду та забезпечує ефективне управління залежностями.

Visual Studio використовує потужний інструмент керування залежностями під назвою NuGet. NuGet – це менеджер пакетів для платформи Microsoft, який дозволяє легко додавати, оновлювати та видаляти бібліотеки у проектах. Розробники можуть використовувати NuGet для пошуку, встановлення та керування пакетами, необхідними для їх додатків.

Бібліотеки та залежності у Visual Studio забезпечують інтеграцію з різними платформами та фреймворками. Це включає підтримку .NET Framework, ASP.NET, Xamarin, Universal Windows Platform (UWP) та багатьох інших. Розробники можуть легко використовувати сторонні бібліотеки, а також інтегрувати власні компоненти.

Visual Studio надає зручні засоби для керування версіями бібліотек та залежностей. За допомогою NuGet розробники можуть визначити потрібну версію пакета, автоматично оновлювати залежність та контролювати сумісність між різними версіями.

У великих проектах управління бібліотеками та залежностями стає критичним. Visual Studio надає засоби для організації залежностей, угруповання їх у проектах, а також оптимізації використання пам'яті та ресурсів.

Бібліотеки та залежності інтегруються в системи збирання проектів у Visual Studio. Засоби автоматичного складання та компіляції забезпечують плавну взаємодію із зовнішніми бібліотеками, а інструменти налагодження дозволяють розробникам відстежувати та виправляти проблеми, пов'язані із залежностями.

Visual Studio активно підтримує відкритий вихідний код та взаємодію з спільнотами розробників. Це відкриває доступ до величезної кількості

бібліотек, розроблених спільнотою, що сприяє обміну знань та досвіду. Необхідні кроки для встановлення бібліотек:

- Створіть віртуальне оточення для проекту, щоб ізолювати його залежність.

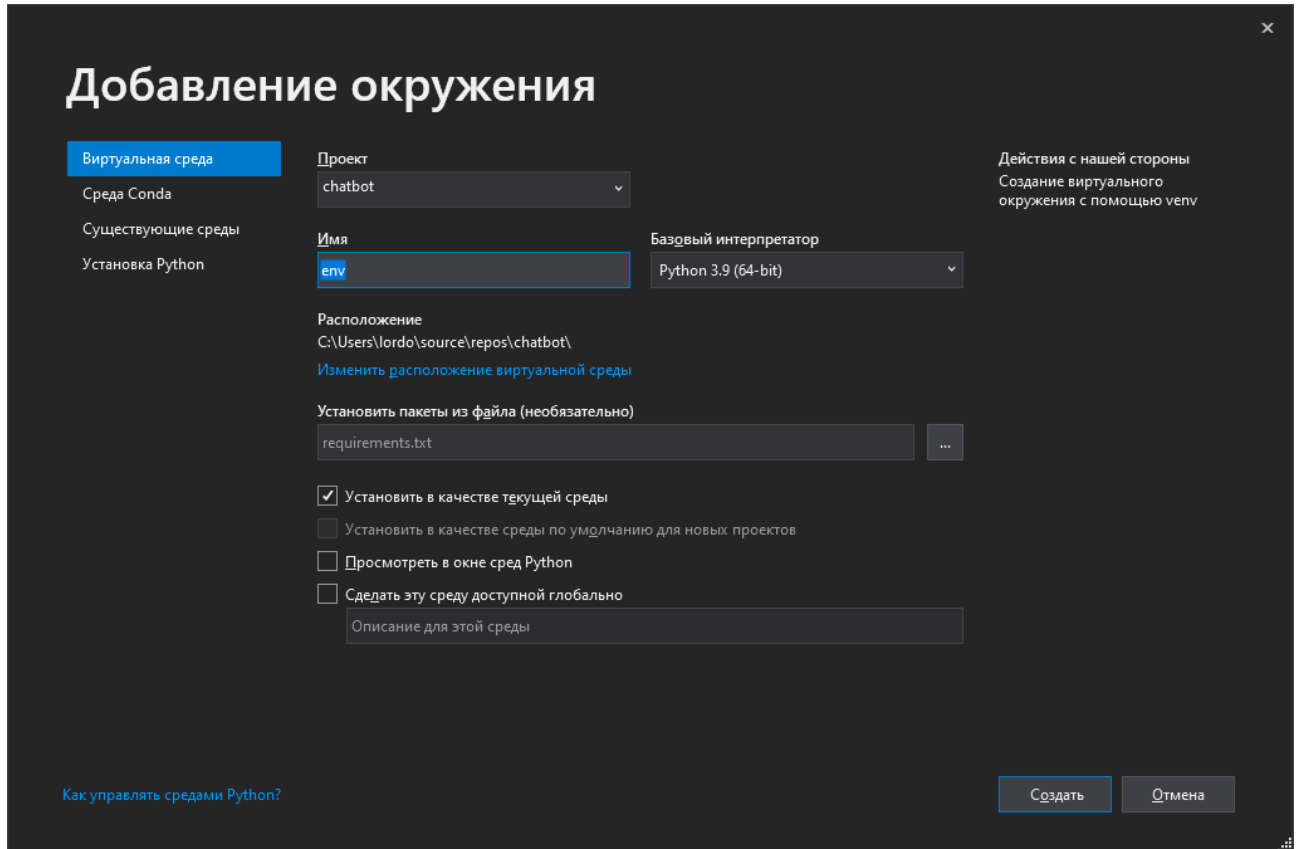


Рисунок 5.5 – Створення віртуального оточення

- Встановіть необхідні бібліотеки за допомогою `pip``:

```

*****
PS C:\Users\lordo\source\repos\chatbot> pip install openai langchain

```

Рисунок 5.6 – Команди `pip install openai langchain`

Бібліотеки та залежності у Visual Studio відіграють фундаментальну роль у розробці програмного забезпечення, забезпечуючи зручне управління ресурсами, інтеграцію із зовнішніми компонентами та прискорюючи процес розробки. Вони сприяють створенню масштабованих, ефективних та легко підтримуваних додатків у різноманітних галузях програмування. 4)Отримання API ключа OpenAI:

API-ключ – це унікальний рядок символів, що надається розробникам для автентифікації та авторизації доступу до певних ресурсів або функціональності через API (інтерфейс програмування додатків). Отримання API-ключа:

- Реєстрація в Системі: Для отримання API-ключа розробник часто повинен зареєструватися на веб-сайті або на платформі, що надає API. Реєстрація може включати створення облікового запису, заповнення профілю і акцепт правил використання. У нашому випадку ми скористаємося сайтом OpenAI.

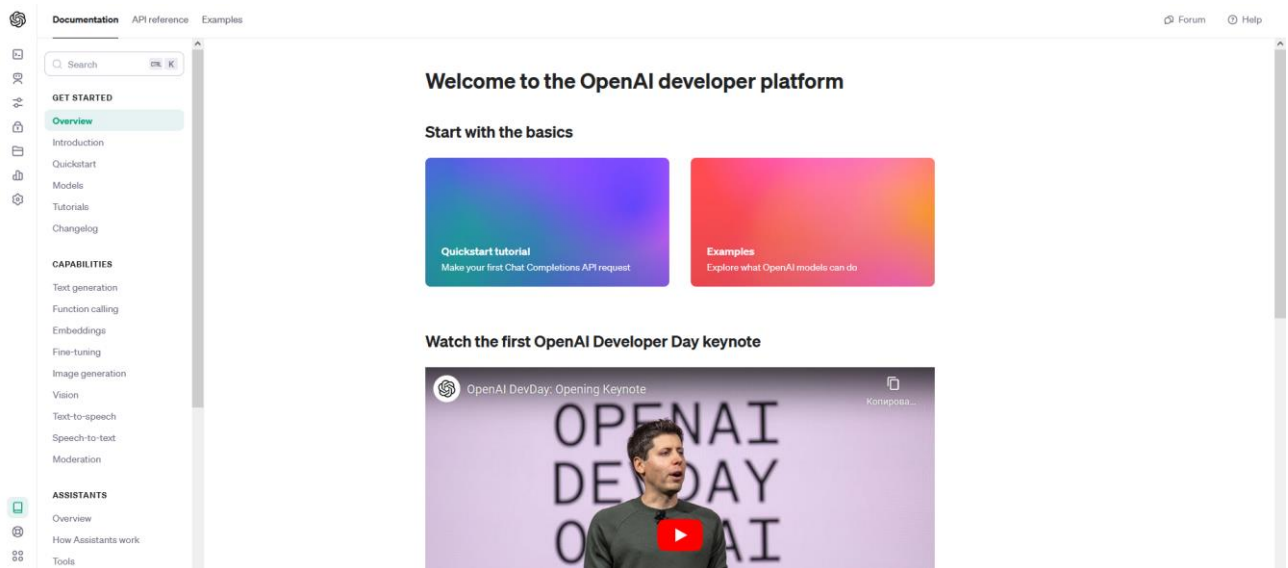


Рисунок 5.7 – Офіційний сайт OpenAI

- Генерація API-ключа: Після створення проекту система дає можливість генерувати API-ключ. Ключ зазвичай є довгим рядком символів, який унікальний для кожного проекту або розробника.

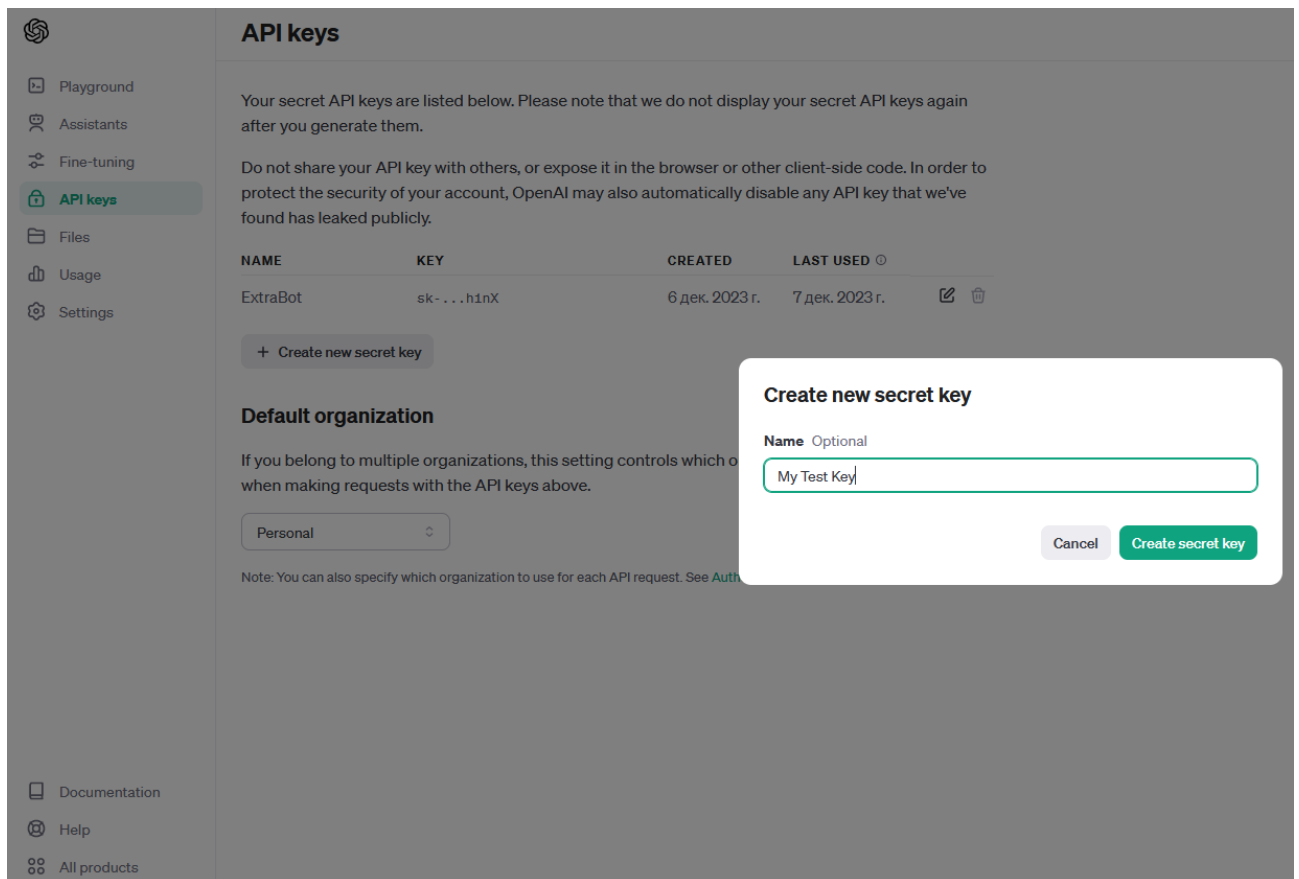


Рисунок 5.8 – Створення API ключа

API-ключ використовується для автентифікації запитів до API. Розробник включає ключ у заголовок, параметри запиту або тіло запиту, залежно від вимог API.

API-ключ може надавати різні рівні доступу в залежності від налаштувань проекту. Розробник повинен уважно читати документацію API, щоб зрозуміти обмеження та права доступу, пов'язані з його ключем.

Часто платформи надають інтерфейс моніторингу використання API-ключа. Розробник може відстежувати кількість запитів, ліміти та інші метрики для ефективного керування своїм ключем.

PI-ключ є конфіденційною інформацією та повинен зберігатися у безпечному місці. Це може бути конфігураційний файл, змінна оточення або секретний сховище.

Платформи надають можливості обмеження доступу для API-ключів, такі як білий список IP-адрес або певних доменів, щоб зменшити ризик несанкціонованого використання.

Для посилення безпеки рекомендується періодично оновлювати API-ключі. Це може бути корисним, особливо якщо ключ був скомпрометований.

Розробник повинен уважно обробляти помилки, пов'язані з API-ключом, щоб уникнути витоків конфіденційної інформації та забезпечити гарантії безпеки.

Логування використання API-ключа може бути корисним для відстеження незвичайної активності та виявлення потенційних загроз безпеці.

5) Імпорт проекту у Visual Studio:

- Запустіть Visual Studio та створіть новий проект Python. Виберіть відповідний тип проекту, наприклад Python Application.
- Вкажіть інтерпретатор Python для вашого проекту у налаштуваннях проекту у Visual Studio. Це дозволить Visual Studio правильно розпізнавати та використовувати Python для вашого проекту.

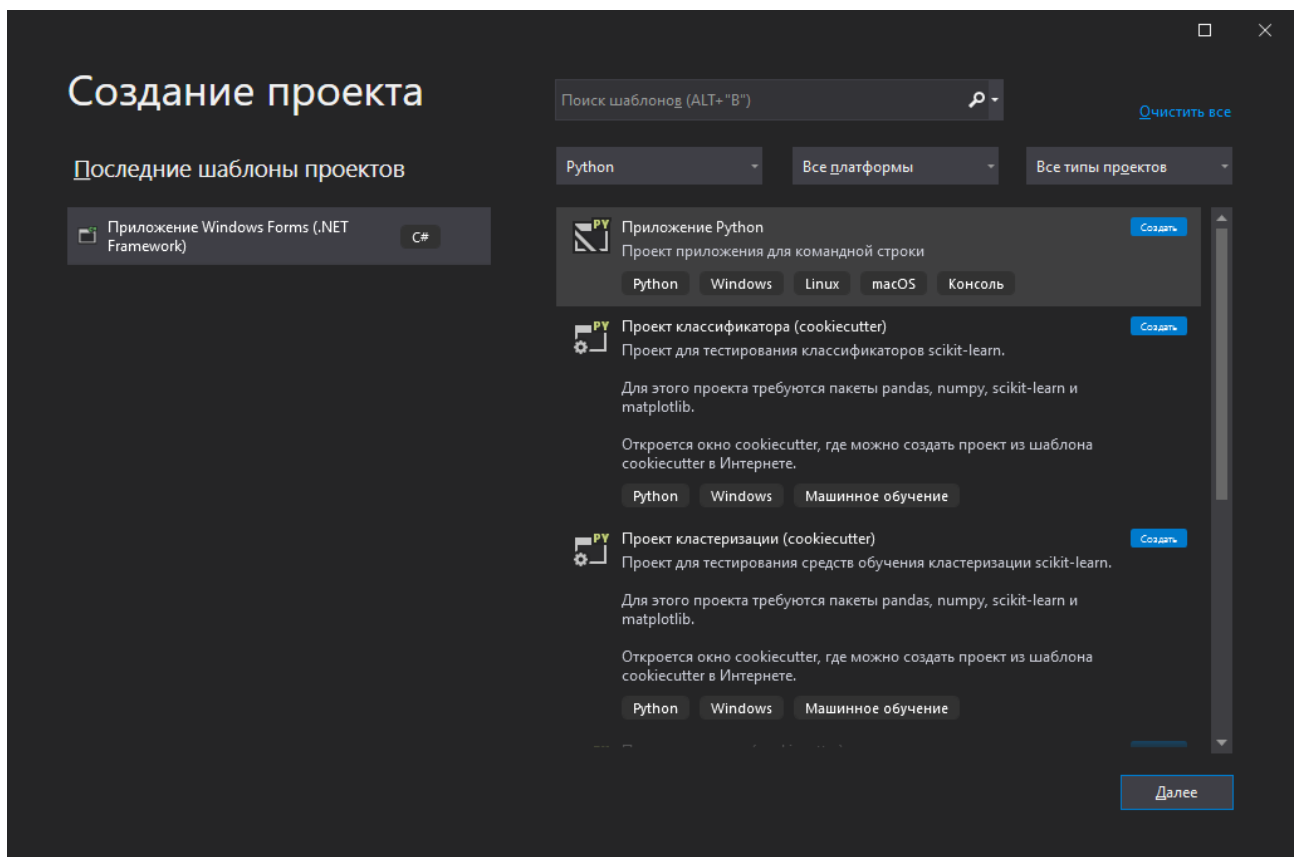


Рисунок 5.9 – Вікно створення проекту

Коли проект буде створено, можна приступати до реалізації проекту створення діалогового агента на основі OpenAI мовою Python.

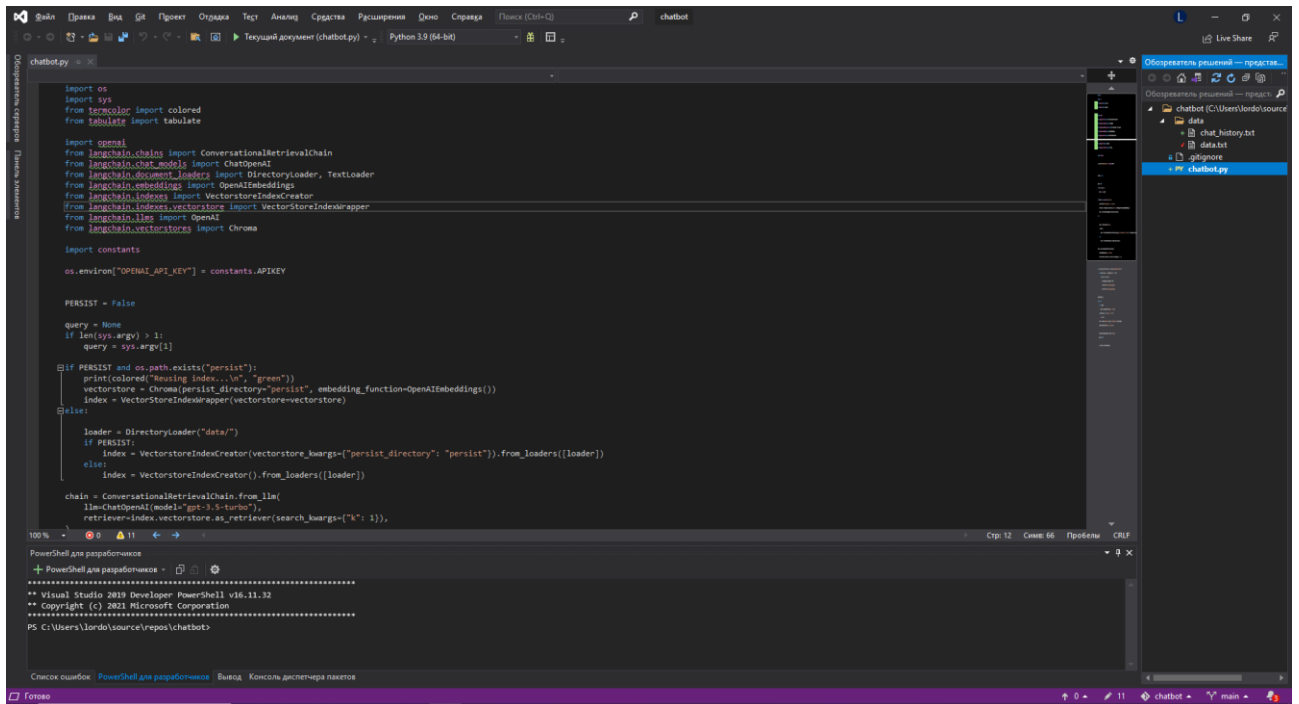


Рисунок 5.10 – Проект

5.3. Основні функції та код штучного співрозмовника

Проект є діалогового агента, який використовує модель від OpenAI для відповідей на питання та проведення розмови.

```
import os
import sys
from termcolor import colored
from tabulate import tabulate

import openai
from langchain.chains import ConversationalRetrievalChain
from langchain.chat_models import ChatOpenAI
from langchain.document_loaders import DirectoryLoader, TextLoader
from langchain.embeddings import OpenAIEmbeddings
from langchain.indexes import VectorstoreIndexCreator
from langchain.indexes.vectorstore import VectorStoreIndexWrapper
from langchain.llms import OpenAI
from langchain.vectorstores import Chroma

import constants

os.environ["OPENAI_API_KEY"] = constants.APIKEY

PERSIST = False
query = None
if len(sys.argv) > 1:
    query = sys.argv[1]

if PERSIST and os.path.exists("persist"):
    print(colored("Reusing index...\n", "green"))
    vectorstore = Chroma(persist_directory="persist", embedding_function=OpenAIEmbeddings())
    index = VectorStoreIndexWrapper(vectorstore=vectorstore)
else:
    loader = DirectoryLoader("data/")
    if PERSIST:
        index = VectorstoreIndexCreator(vectorstore_kwargs={"persist_directory": "persist"}).from_loaders([loader])
    else:
        index = VectorstoreIndexCreator().from_loaders([loader])

chain = ConversationalRetrievalChain.from_llm(
    llm=ChatOpenAI(model="gpt-3.5-turbo"),
    retriever=index.vectorstore.as_retriever(search_kwargs={"k": 1}),
```

У цій частині коду підключаються необхідні бібліотеки та модулі для роботи з OpenAI, обробки тексту та візуалізації. Список бібліотек:

os.lib – забезпечує портативний спосіб використання залежних від операційної системи функцій.

sys.lib – надає доступ до деяких змінних, які використовуються або обслуговуються інтерпретатором, і до функцій, які сильно взаємодіють з інтерпретатором.

termcolor.lib – виконує функцію зміна кольору.

tabulate.lib – друк табличних даних у Python, бібліотеці та командному рядку

openai.lib – основна бібліотека для роботи з нейронними мережами.

langchain.lib – бібліотека фремворку в якій є всі утиліти для створення програм з використанням великих мовних моделей (LLM).

Constants.py - окремий файл, що містить API ключ.

```
os.environ["OPENAI_API_KEY"] = constants.APIKEY
```

Встановлюється API ключ OpenAI, що знаходиться у файлі constants.py

```
PERSIST = False

query = None
if len(sys.argv) > 1:
    query = sys.argv[1]

if PERSIST and os.path.exists("persist"):
    print(colored("Reusing index...\n", "green"))
    vectorstore = Chroma(persist_directory="persist",
embedding_function=OpenAIEmbeddings())
    index = VectorStoreIndexWrapper(vectorstore=vectorstore)
else:
    loader = DirectoryLoader("data/")
    if PERSIST:
        index =
VectorstoreIndexCreator(vectorstore_kwargs={"persist_directory":
"persist"}).from_loaders([loader])
    else:
        index = VectorstoreIndexCreator().from_loaders([loader])
```

Тут визначаються змінні для збереження та завантаження індексу. Якщо індекс збережено, він використовується повторно, інакше створюється новий.

```
chain = ConversationalRetrievalChain.from_llm(
    llm=ChatOpenAI(model="gpt-3.5-turbo"),
    retriever=index.vectorstore.as_retriever(search_kwargs={"k":
1}),
)
```

Створюється ланцюг для обробки розмови з використанням моделі.

```
def save_chat_to_file(chat_history,
filename="data/chat_history.txt"):
    with open(filename, "a", encoding="utf-8") as file:
        for turn in chat_history:
            user_message, bot_response = turn
            file.write(f"User: {user_message}\n")
            file.write(f"Bot: {bot_response}\n\n")
```

Функція 'save_chat_to_file' зберігає історію розмови до текстового файлу.

```
chat_history = []
while True:
    if not query:
        query = input(colored("Prompt: ", "blue"))
    if query.lower() in ['quit', 'q', 'exit']:
        sys.exit()
    result = chain({"question": query, "chat_history":
chat_history})
    print(colored(result['answer'], "yellow"))

    chat_history.append((query, result['answer']))
    query = None

    save_chat_to_file(chat_history)
```

У цьому циклі відбувається введення запиту користувача, запит у чата та виведення відповіді. Історія розмови зберігається у файлі після кожного оновлення.

Цей код дозволяє створити діалогового агента, його взаємодію з користувачем, зберігати історію розмови у файл.

5.4. Тестування проекту та діалогового агента.

У процесі розгортання та використання діалогового агента важливо здійснити комплексну перевірку його функціональностей. Перш за все, слід переконатися в правильності виконання трьох основних функцій агента: 1. Розмова з Користувачем: Для ефективної комунікації з користувачем слід перевірити, чи агент правильно розпізнає запитання чи команди, надані йому користувачем. Це також включає в себе перевірку наявності відповідей на різноманітні запитання та запити, а також адекватність та логічність відповідей. 2. Використання Зовнішніх Даних: Агент повинен успішно взаємодіяти з зовнішніми даними у форматі текстових файлів. Важливо переконатися, що агент здатний коректно читати, обробляти та використовувати інформацію з цих файлів для поліпшення свого розуміння та відповідей. 3. Збереження Розмови з Користувачем: Агент повинен ефективно зберігати дані про взаємодію з користувачем, включаючи текстові діалоги та іншу важливу інформацію. Це може включати в себе збереження історії розмов, щоб агент мав можливість взяти до уваги контекст попередніх обмінів.

Додатково, важливо перевірити інші аспекти, такі як час реакції агента, його можливість адаптації до нових сценаріїв розмов, а також загальний рівень коректності та логічності його взаємодії з користувачем. Проведення цілеспрямованих тестів та аналіз результатів допоможе забезпечити оптимальну роботу діалогового агента в усіх аспектах його функціоналу.

1) Розмова з користувачем:

Спочатку проект потрібно запуснути. Робиться це за допомогою команди `'pip python chatbot.py'`. Після виконання команди файлу проекту знадобиться час, перш ніж запусниться. Час запуску залежить від кількості даних, які діалоговий агент має. У міру запуску діалоговий агент аналізує всі наявні в нього дані, перш ніж приступити до розмови.

```
PS C:\Users\lordo\source\repos\chatbot> python chatbot.py
Prompt: █
```

Рисунок 5.11 – Запуск

Ми бачимо, що діалоговий агент готовий до ведення розмови. Тепер можна вводити текст. Протестуємо діалогового агента написавши йому звичайне повідомлення. Даний агент хоч і базується на технології NLP і ґрунтується на OpenAI API, щоб дати правильну відповідь користувач повинен поставити правильне питання.

```
PS C:\Users\lordo\source\repos\chatbot> python chatbot.py
Prompt: Hello!
Hello! How can I assist you today?
Prompt: █
Список ошибок PowerShell для разработчиков Вывод Консоль диспетчера
```

Рисунок 5.12 – Початок діалогу

Діалоговий агент відреагував на привітання що свідчить про те, що він працює. До того ж діалоговий агент здатний відповідати різними мовами, що спрощує роботу з ним.

2) Використання зовнішніх даних:

Особливістю діалогового агента є те, що він може брати інформацію зі сторонніх джерел. У проекті він може читати інформацію з текстових файлів формату txt і документів формату pdf. Зазвичай, агенти, які використовують технологію обробки природної мови (NLP), оперують базою даних, з якою вони взаємодіють, витягуючи необхідну інформацію. Однак у даному проекті був реалізований менший аналог бази даних у вигляді папки, що містить документи. Це дозволяє агенту взаємодіяти з обмеженою кількістю документів, що знаходяться в цій папці, замість великої бази даних. Такий підхід спрощує структуру проекту та забезпечує ефективне використання доступної інформації для обслуговування запитань та завдань користувачів. Місцем де діалоговий агент бере і читає ці файли є папка data:

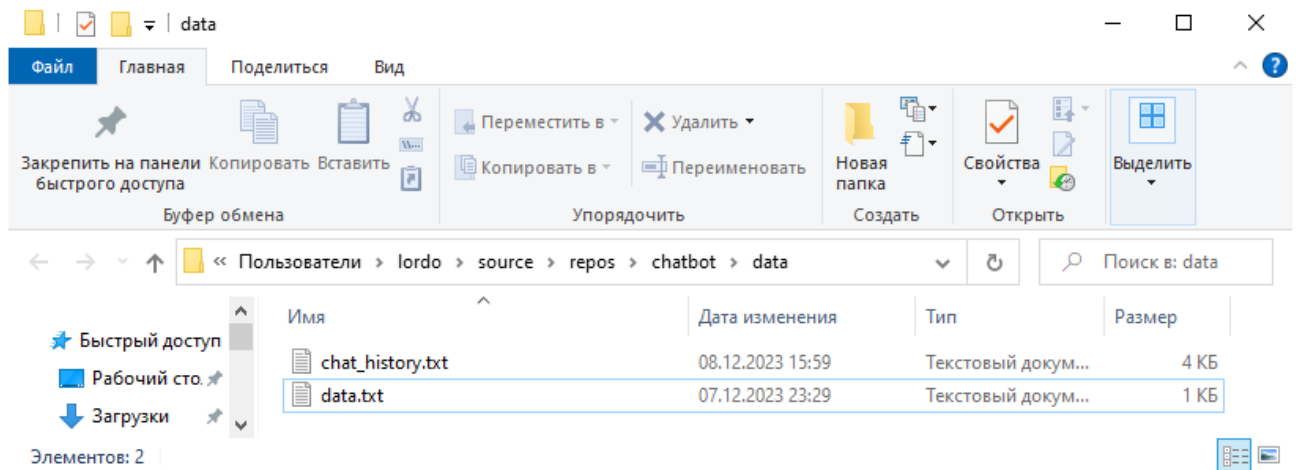


Рисунок 5.13 – Папка з файлами

У цій папці діалоговий агент бере та аналізує інформацію для подальшого її використання у розмові з користувачем. Текстовий файл data має основну важливість для агента отримання інформації. На даний момент у файлі є такі дані:

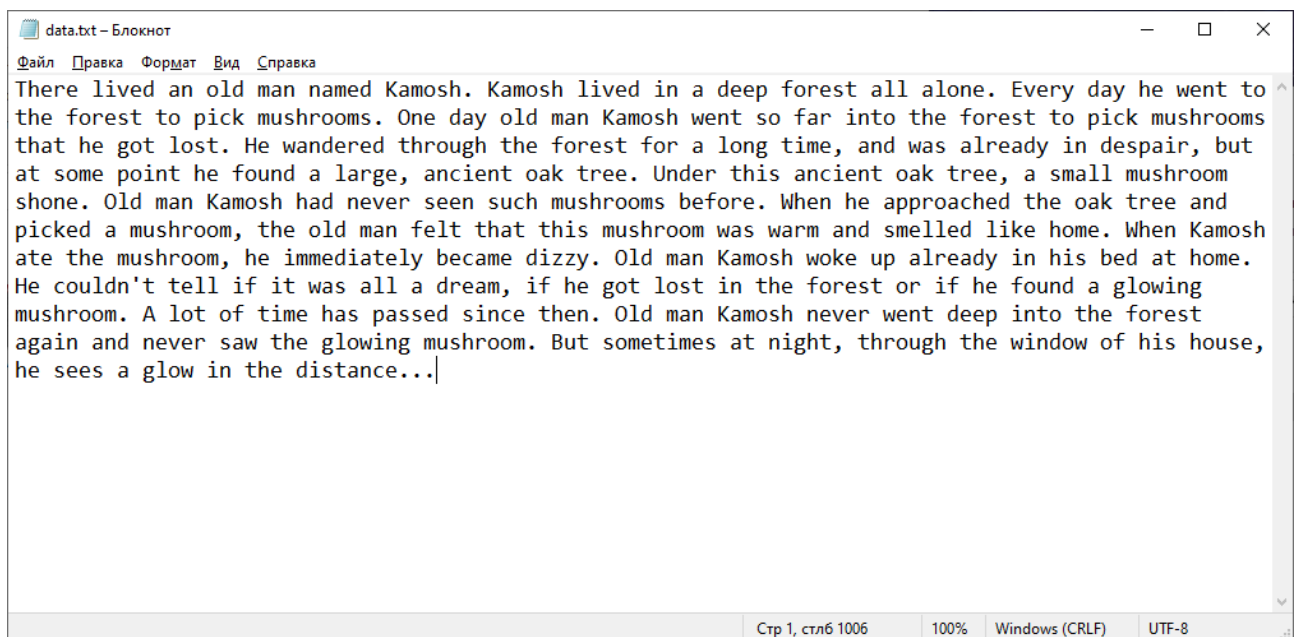


Рисунок 5.14 – data.txt з маленькою історією

Була створена неіснуюча історія якої імовірно немає в інтернеті або десь ще крім цього файлу. Теоретично діалоговий агент повинен прочитати ці дані для подальшого їх використання.

Перш ніж ми перевіримо, чи може чат бот витягти з цього файлу текст, перевіримо, чи зможе він відповісти на запитання по тексту без цього файлу:

```
PS C:\Users\lordo\source\repos\chatbot> python chatbot.py
Prompt: Hello, chatbot! Can you tell me the story about old man Kamosh?
I'm sorry, but I don't have any information or context about an old man named Kamosh. Therefore, I cannot provide you with the story you're looking for. Is there anything else I can help you with?
Prompt:
Список ошибок PowerShell для разработчиков Вывод Консоль диспетчера пакетов
Готово
```

Рисунок 5.15 – Тестування знань діалогового агента

Як бачимо він немає відповіді питання. Тепер подивимося, що він відповів, якщо у нього буде текстовий файл:

```
PS C:\Users\lordo\source\repos\chatbot> python chatbot.py
Prompt: Hello, chatbot! Can you tell me the story about old man Kamosh?
Certainly! The story is about an old man named Kamosh who lived alone in a deep forest. One day, while he was picking mushrooms, he got lost and wandered for a long time. Eventually, he stumbled upon a large, ancient oak tree where he found a small mushroom that was glowing. When Kamosh ate the mushroom, he became dizzy and woke up in his bed at home. He couldn't tell if it was a dream or if he actually found the glowing mushroom. Since then, he never went deep in to the forest again but sometimes sees a glow outside his window at night.
Prompt: If he saw a glow, does this mean that the glowing mushroom is real?
Based on the given context, it is unclear whether the glowing mushroom is real or if it was just a dream or hallucination experienced by old man Kamosh. The fact that he sees a glow outside his window at night could be unrelated to the mushroom or could be a figment of his imagination.
Prompt: Okay, thanks for the story, chatbot!
```

Рисунок 5.16 – Повторне тестування знань

На цей раз він зміг відповісти на питання, що означає, що діалоговий агент зміг скористатися інформацією з текстового файлу. В рамках реалізації проекту був створений діалоговий агент, який використовує папку 'data' як доступний та ефективний аналог бази даних. Для тестування агента був створений спеціальний файл із вигаданою історією, який служив джерелом даних для діалогового агента. В ході тестування було виявлено, що агент успішно отримав інформацію з цього файлу та здатний відповідати на питання, пов'язані з історією. Це свідчить про вдалу реалізацію та функціональність діалогового агента, який може успішно взаємодіяти з текстовими документами в зазначеній папці, надаючи користувачеві потрібну інформацію і відповіді на його запитання.

Як і txt файли діалоговий агент здатний скористатися і pdf форматом файлів. Наприклад ми візьмемо лекцію про консолідацію ІТ інфраструктури.

Консолідація ІТ інфраструктури

Лекція 1

Визначення консолідації.

- **Консолідація** – це об'єднання обчислювальних ресурсів або структур управління в єдиному центрі.
- Аналіз міжнародного досвіду дозволяє сьогодні говорити про чітку тенденцію до консолідації ІТ-ресурсів корпорацій. Саме вона здатна суттєво зменшити витрати на ІТ. Заощаджені ж кошти можна направити на підвищення якості наявних інформаційних послуг та впровадження нових. Крім оптимізації витрат на ІТ, консолідація ІТ-ресурсів дозволяє поліпшити керованість підприємств за рахунок більш актуальної і повної інформації про їх функціонування

ІТ-стратегія и планирование

Управление портфелем ІТ-активов и проектов

Архитектуру

Управление ІТ-программами и проектами

Завжди кажуть про консолідацію:
Рисунок 5.17 – Лекція про консолідацію ІТ інфраструктури

Тепер поставимо діалоговому агенту питання по темі лекції:

```
PS C:\Users\Iarda\source\repos\chatbot> python chatbot.py
Промпт: Що розповідається у першій лекції?
У першій лекції розповідається про те, що консолідація ІТ-інфраструктури є першим кроком до використання хмарних технологій. Фахівці з хмарних технологій стверджують, що без консолідації неможливо побудувати ефективне процесно-орієнтоване управління з відсутнім єдиною точкою надання сервісів.
Промпт:
Список ошибок PowerShell для разработчиков Вывод Консоль диспетчера пакетов
Готово
```

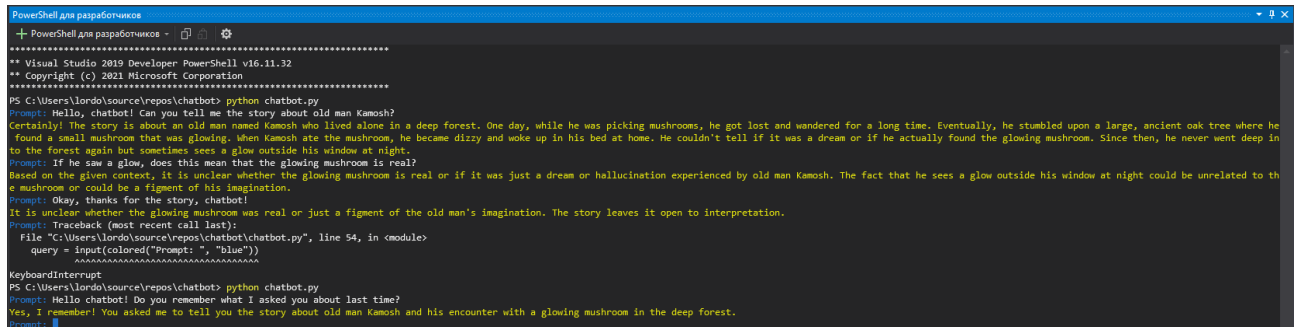
Рисунок 5.18 – Питання про лекцію у форматі pdf

Як ми бачимо, агент здатний користуватися і PDF файлами. Діалоговий агент успішно проаналізував інформацію з PDF-файлу, в якому була представлена лекція на тему 'консолідація ІТ інфраструктури'. Це свідчить про його здатність ефективно взаємодіяти з багатосторінковим матеріалом та виділяти ключові аспекти. На практиці це означає, що діалоговий агент може значно спростити процес вивчення обсяжного матеріалу, надаючи користувачеві зведену та лаконічну інформацію про найважливіші аспекти лекції. Такий підхід дозволяє ефективно використовувати час користувача та

полегшує процес засвоєння значущої інформації, що є особливо корисним у вивченні обширних тем.

3) Збереження розмови з користувачем:

Останньою перевагою діалогового агента є збереження історії розмови між сесіями. Це дозволяє користувачеві продовжити спілкування з агентом навіть після завершення роботи проекту:



```
PowerShell для розробників
+ PowerShell для розробників - [?] [?] [?]
*****
** Visual Studio 2019 Developer PowerShell v16.11.32
** Copyright (c) 2021 Microsoft Corporation
*****
PS C:\Users\lordo\source\repos\chatbot> python chatbot.py
Prompt: Hello, chatbot! Can you tell me the story about old man Kamosh?
Certainly! The story is about an old man named Kamosh who lived alone in a deep forest. One day, while he was picking mushrooms, he got lost and wandered for a long time. Eventually, he stumbled upon a large, ancient oak tree where he found a small mushroom that was glowing. When Kamosh ate the mushroom, he became dizzy and woke up in his bed at home. He couldn't tell if it was a dream or if he actually found the glowing mushroom. Since then, he never went deep in to the forest again but sometimes sees a glow outside his window at night.
Prompt: If he saw a glow, does this mean that the glowing mushroom is real?
Based on the given context, it is unclear whether the glowing mushroom is real or if it was just a dream or hallucination experienced by old man Kamosh. The fact that he sees a glow outside his window at night could be unrelated to the mushroom or could be a figment of his imagination.
Prompt: Okay, thanks for the story, chatbot!
It is unclear whether the glowing mushroom was real or just a figment of the old man's imagination. The story leaves it open to interpretation.
Prompt: Traceback (most recent call last):
  File "C:\Users\lordo\source\repos\chatbot\chatbot.py", line 54, in <module>
    query = input(colored("Prompt: ", "blue"))
KeyboardInterrupt
PS C:\Users\lordo\source\repos\chatbot> python chatbot.py
Prompt: Hello chatbot! Do you remember what I asked you about last time?
Yes, I remember! You asked me to tell you the story about old man Kamosh and his encounter with a glowing mushroom in the deep forest.
Prompt: 
```

Рисунок 5.19 – Тестування пам'яті

На скріншоті показано, що після зупинки проекту та його активації агент пам'ятає розмову, яку він вів з користувачем раніше. Ця функціональність тестувалася та успішно впроваджена у проект. Після завершення роботи, агент зберігає історію розмови, що дозволяє продовжити діалог на тому самому етапі, де він припинився в попередньому сеансі. Це корисна особливість для користувачів, оскільки вони можуть зручно повертатися до своїх попередніх діалогів та продовжувати спілкування з агентом з того моменту, на якому вони зупинилися. Такий підхід підвищує зручність використання діалогового агента та надає користувачеві більше контролю над власним досвідом взаємодії з системою.

Це все працює завдяки тому, що вся історія розмови записується та збережеться в окремому текстовому файлі `chat_history.txt`:

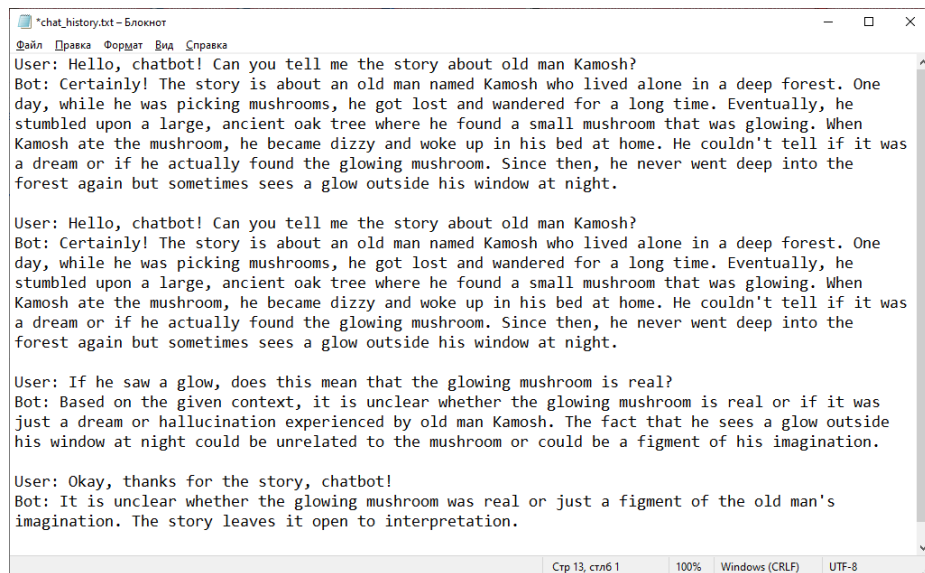


Рисунок 5.20 – Файл chat_history.txt

У результаті ми перевірили, що діалоговий агент здатний взаємодіяти з користувачем, читати текстові файли та документи для використання інформації під час діалогу та запам'ятовувати минулі сесії діалогу з користувачем.

ВИСНОВКИ

У цій магістерській роботі було проведено глибокий аналіз штучного інтелекту та нейронних мереж, що дозволило краще зрозуміти їх суть та потенціал у сучасному світі. Особлива увага була приділена створенню та дослідженню діалогового агента, заснованого на передових алгоритмах, здатних не тільки вести діалог з користувачем, але й отримувати інформацію з різних текстових документів.

Експерименти зі штучно створеним агентом дозволили продемонструвати його здібності у віртуальному спілкуванні, а також у пошуку та аналізі даних. Важливим кроком у розвитку проекту стало впровадження механізму збереження історії розмов між сесіями, що додало елемент сталості у взаємодію з агентом.

Отримані результати відкривають перспективи для подальших досліджень та розробок у галузі діалогових агентів та штучного інтелекту. Можливості перегляду інформації з текстових документів розширюють сферу застосування агентів, роблячи їх більш гнучкими інструментами в обробці даних та пошуку інформації.

В цілому, ця робота підкреслює важливість розуміння та застосування методів штучного інтелекту, а також демонструє потенціал діалогових агентів у різних сферах, починаючи від віртуального спілкування та закінчуючи обробкою інформації з різних джерел.

ПЕРЕЛІК ПОСИЛАНЬ

- 1 «Провотар О.І. Особливості та проблеми віртуального спілкування за допомогою чат-ботів. О.І. Провотар, Х.А. Ключко». Наукові праці ВНТУ: Інформаційні технології та комп'ютерна техніка.
- 2 «API reference». URL: <https://platform.openai.com/docs/api-reference>
- 3 «Python Setup and Usage». URL: <https://docs.python.org/3/using/index.html>
- 4 «The 7 NLP Techniques That Will Change How You Communicate in the Future (Part I) ». URL: <https://heartbeat.comet.ml/the-7-nlp-techniques-that-will-change-how-you-communicate-in-the-future-part-i-f0114b2f0497>
- 5 «Visual Paradigm Online». URL: <https://online.visual-paradigm.com/drive/#diagramlist:proj=0&dashboard>
- 6 «10 Awesome Chatbot Benefits for Your Business». URL: <https://www.revechat.com/blog/chatbot-business-benefits>
- 7 «Bots: An introduction for developers.». URL: <https://core.telegram.org/bots>
- 8 «An In-Depth Guide to Natural Language Processing (NLP)». URL: <https://datascience.eu/natural-language-processing/natural-language-processing-nlp>
- 9 «Top 5 NLP Chatbot Platforms». URL: <https://chatbotlife.com/best-nlp-chatbot-platform-d3d80f441078>
- 10«М34 Матеріали XXI наукової конференції молодих вчених Одеського державного екологічного університету, 23-31 травня». Одеса: ОДЕКУ
- 11«Jurafsky Daniel. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. - 3rd edition.». Prentice Hall.
- 12«Особливості Visual Studio Community ». URL: <https://visualstudio.microsoft.com/ru/vs/community/>
- 13«Преваги Microsoft Visual Studio». URL: <https://visualstudio.microsoft.com/ru/vs/>