

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ
ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІЦІЙНА РОБОТА

другого (магістерського) рівня вищої освіти

на тему:

**«ОБҐРУНТУВАННЯ ТЕХНОЛОГІЙ ГЕНЕРАТИВНОГО ШТУЧНОГО
ІНТЕЛЕКТУ ДЛЯ СЕРВІСНОЇ ПІДТРИМКИ КЛІЄНТІВ»**

Виконав: студент

групи ІТ-61

спеціальності 126 «Інформаційні
системи та технології»

Мащенко Олег Олегович
(прізвище та ініціали)

Керівник: к.е.н., доц. Смолінський В. Б.
(прізвище та ініціали)

Рецензент: _____
(прізвище та ініціали)

ДУБЛЯНИ 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ
ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Другий (магістерський) рівень вищої освіти
Спеціальність 126 «Інформаційні системи та технології»

«ЗАТВЕРДЖУЮ»

Завідувач кафедри _____

«___» _____ 202_ р.

ЗАВДАННЯ

на кваліфікаційну роботу здобувачу

Мащенку Олегу Олеговичу

1. **Тема роботи:** «Обґрунтування технологій генеративного штучного інтелекту для сервісної підтримки клієнтів»

Керівник роботи: доцент Смолінський В.Б.

затверджена наказом по університету від 12.09.2024 року № 616/к-с.

2. **Строк подання здобувачем роботи** 06.12.2024 р.

3. **Вихідні дані до роботи:** Сучасні підходи до автоматизації сервісної підтримки, великі мовні моделі (LLM), принципи генеративного штучного інтелекту, локальні моделі Ollama, інтеграція чат-ботів із Telegram, методи тонкого налаштування (Fine-Tuning), Retrieval-Augmented Generation (RAG), архітектурні рішення для чат-ботів, інструменти логування та аналізу запитів.

4. **Зміст розрахунково-пояснювальної записки** (перелік питань, які необхідно розробити)

Розділ 1. Аналіз предметної області генеративного інтелекту та сервісної підтримки клієнтів.

Розділ 2. Проектування системи для сервісної підтримки клієнтів.

Розділ 3. Розробка чат бота для сервісної підтримки клієнтів.

Розділ 4. Охорона праці та безпека в надзвичайних ситуаціях.

Розділ 5. Доцільність проекту та перспективи розвитку.

5. **Перелік графічного матеріалу:** (з точним зазначенням обов'язкових схем та моделей): Схема роботи ГШІ. Схематична ілюстрація архітектури моделі трансформера. Схема роботи автоматизованої системи сервісної підтримки клієнтів. Архітектура локального розгортання моделі Ollama та інтеграція з додатками. Схема роботи мультиплатформенності BotFramework, де один бекенд підключається до різних месенджерів. Порівняльна таблиця характеристик GPT-2, Llama та Mistral NeMo.

6. Консультанти з розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1, 2, 3, 5	<i>Смолінський В.Б., доцент кафедри інформаційних технологій</i>		
4	<i>Городецький І.М., доцент кафедри фізики, інженерної механіки та безпеки виробництва</i>		

7. Дата видачі завдання 12 вересня 2024 р.

Календарний план

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів роботи	Примітка
1	Написання першого розділу: Аналіз предметної області генеративного інтелекту та його використання на практиці, а також аналіз області сервісної підтримки клієнтів	12.09.2024 – 15.10.2024	
2	Виконання другого розділу: Проектування системи сервісної підтримки клієнтів	13.10.2024 – 27.10.2024	
3.	Виконання третього розділу: Розробка чатбота для сервісної підтримки клієнтів	28.10.2024 – 10.11.2024	
4.	Написання розділу: «Охорона праці та безпека в надзвичайних ситуаціях»	11.11.2024 – 16.11.2024	
5.	Виконання п'ятого розділу: Доцільність проекту та перспективи розвитку	17.11.2024- 20.11.2024	
6.	Написання висновків та пропозицій. Завершення оформлення пояснювальної записки та презентаційних матеріалів	21.11.2024- 1.12.2024	
7.	Завершення роботи в цілому	2.12.2024 – 10.12.2024	

Здобувач

(підпис)

Мащенко О.О.

(прізвище та ініціали)

Керівник роботи

(підпис)

Смолінський В.Б.

(прізвище та ініціали)

АНОТАЦІЯ

УДК: 004.94: 004.8

Обґрунтування технологій генеративного штучного інтелекту для сервісної підтримки клієнтів. Мащенко О.О., Смолінський В.Б. (керівник).
Кафедра ІТ. Дубляни, Львівський НУП, 2024.

Кваліфікаційна робота викладена на 87 сторінках, містить 5 розділів, 36 рисунків, 4 таблиці.

У роботі проаналізовано сучасні підходи до автоматизації процесів сервісної підтримки із використанням великих мовних моделей (LLM). Розглянуто особливості генеративного штучного інтелекту, його переваги та обмеження для впровадження в освітніх установах. Виконано аналіз предметної області, досліджено функціональні можливості існуючих аналогів, розроблено концепцію та реалізовано прототип чат-бота для відділу комп'ютерних інформаційних технологій Львівського національного університету природокористування.

Розроблений чат-бот має модульну структуру реалізовану на BotFramework та до нього підключено модуль роботи через бота Telegram, він використовує локальні LLM-моделі Ollama для генерації відповідей і забезпечує логування для подальшого аналізу. Система автоматизує обробку рутинних запитів, що знижує навантаження на персонал та забезпечує швидкий доступ до інформації. Проведене тестування підтвердило відповідність функціональних можливостей заданим вимогам та ефективність роботи чат-бота.

Мета роботи – створення чат-бота для автоматизації сервісних процесів у відділі КІТ, що використовує генеративний ШІ та забезпечує інтеграцію з популярними платформами.

Для досягнення мети виконано наступні завдання:

- аналіз предметної області та визначення функціональних вимог;
- проектування архітектури системи та вибір інструментів розробки;
- створення прототипу з інтеграцією в Telegram;
- тестування системи у різних середовищах.

Практична значущість роботи полягає у впровадженні інноваційного рішення для автоматизації рутинних задач, що підвищує ефективність сервісної підтримки. Запропоноване рішення можна адаптувати для інших структурних підрозділів або організацій, розширюючи сферу його застосування.

Ключові слова: великі мовні моделі, чат-бот, автоматизація, генеративний ШІ, Telegram, Ollama, сервісна підтримка, тестування, інтеграція, навчальні заклади.

ЗМІСТ

ВСТУП	6
РОЗДІЛ 1.АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ГЕНЕРАТИВНОГО ІНТЕЛЕКТУ ТА СЕРВІСНОЇ ПІДТРИМКИ КЛІЄНТІВ.....	8
1.1 Сутність та принципи генеративного штучного інтелекту	8
1.1.1 Навчання нейромережі	9
1.1.2 Донавчання нейромережі	10
1.1.3 Принципи роботи генеративного ШІ.....	13
1.1.4 Використання ГШІ в практиці.....	15
1.1.5 Виклики, з якими стикається ГШІ	15
1.2 Сервісна підтримка клієнтів: характеристика та особливості	16
1.2.1 Особливості сервісної підтримки.....	16
1.2.2 Особливості впровадження автоматизації у підтримку	18
1.2.3 Проблеми та виклики автоматизованої сервісної підтримки.....	19
1.3 Огляд і аналіз існуючих рішень на базі генеративного ШІ.....	20
1.3.1 Чат-боти для підтримки клієнтів та внутрішніх комунікацій.....	21
1.3.2 Системні агенти для складних завдань і прогнозів.....	25
1.3.3 Виклики впровадження генеративного ШІ	26
РОЗДІЛ 2.ПРОЕКТУВАННЯ СИСТЕМИ ДЛЯ СЕРВІСНОЇ ПІДТРИМКИ КЛІЄНТІВ	29
2.1 Формулювання вимог до технології для сервісної підтримки.....	29
2.2 Вибір платформи для розробки чат-бота.....	30
2.2.1 Обґрунтування вибору платформи Ollama	31
2.2.2 Обґрунтування вибору платформи BotFramework.....	33
2.3 Вибір мови програмування	35
2.4 Вибір нейронної моделі.....	36
2.5 Вибір бази даних, проектування структури бази даних	39
РОЗДІЛ 3.РОЗРОБКА ЧАТ БОТА ДЛЯ СЕРВІСНОЇ ПІДТРИМКИ КЛІЄНТІВ	40
3.1 Встановлення і налаштування платформи Ollama для локального розгортання моделі нейромережі	40
3.2 Організація структури проекту і розробка чат бота.....	43
3.2.1 Структура проекту	43
3.2.2 Детальний опис файлу main.py.....	43
3.2.3 Детальний опис файлу tg_bot.py	45

3.3 Тестування та експлуатація.....	47
3.3.1 Тестування бота через BotFramework Emulator	47
3.3.2 Задання контексту через YAMЛ файл	54
3.3.3 Підключення модуля телеграм.....	56
3.3.4 Демонстрація логування діалогів та інших логів програми для подальшого донавчання та допрацювання програми.....	58
РОЗДІЛ 4.ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ	61
4.1 Аналіз травмонебезпечних ситуацій під час виконання робіт.....	61
4.2 Структурно-функціональний аналіз дотримання охорони праці при виконання при роботі з комп'ютером.....	62
4.3Обґрунтування організаційно-технічних рекомендацій з охорони праці	63
4.4 Безпека в надзвичайних ситуаціях	64
РОЗДІЛ 5.ДОЦІЛЬНІСТЬ ПРОЕКТУ ТА ПЕРСПЕКТИВИ РОЗВИТКУ	65
5.1 Практична доцільність.....	65
5.1.1 Використання LLM моделей	65
5.1.2 Впровадження бота електронного консультанта.....	65
5.1.3 Загальна економічна доцільність	66
5.1.4 Вплив на імідж університету	67
5.2 Можливі шляхи розвитку чат-бота	68
5.3 Ризики та обмеження.....	70
ВИСНОВКИ.....	72
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	74
ДОДАТКИ.....	77
Додаток А.....	77
Додаток Б	78

ВСТУП

У сучасному світі розвиток технологій на основі штучного інтелекту (ШІ) відкриває нові можливості для автоматизації та оптимізації роботи у різних галузях. Однією з таких технологій є великі мовні моделі (LLM), які можуть виконувати завдання генерації текстів, розпізнавання запитів та автоматизованої взаємодії з користувачами. Використання LLM відкриває перспективи не лише для комерційних компаній, але й для освітніх та адміністративних установ, таких як університети.

Ця дипломна робота присвячена розробці чат-бота для відділу комп'ютерних інформаційних технологій (КІТ) Львівського національного університету природокористування, який виконуватиме роль електронного консультанта. Чат-бот покликаний автоматизувати обробку запитів користувачів, зменшити навантаження на персонал та забезпечити швидкий і зручний доступ до інформації. Особливістю роботи є детальне дослідження підходів до донавчання великих мовних моделей (LLM), включаючи використання системного повідомлення (System Message), тонке налаштування (Fine-Tuning) та інтеграцію із зовнішніми джерелами даних через метод RAG (Retrieval-Augmented Generation).

Мета дослідження – проектування та розробка прототипу чат-бота, здатного надавати консультаційну підтримку користувачам, оптимізувати роботу відділу КІТ та забезпечити інтеграцію зі зручними платформами, такими як Telegram.

Актуальність теми обумовлена необхідністю підвищення ефективності сервісної підтримки у навчальних закладах, що стикаються з великою кількістю запитів від студентів і працівників. Традиційні методи підтримки, такі як особисті звернення або електронна пошта, часто є повільними та ресурсозатратними.

Інтеграція чат-ботів на основі LLM пропонує автоматизацію рутинних завдань, зменшення витрат часу та підвищення якості обслуговування.

У ході роботи були поставлені такі завдання:

Проведення аналізу предметної області, включаючи дослідження принципів генеративного ШІ та сучасних підходів до автоматизації підтримки користувачів.

Дослідження існуючих аналогів чат-ботів для визначення їхніх переваг, недоліків та потенційних областей вдосконалення.

Розробка функціональних вимог до чат-бота з урахуванням специфіки завдань відділу КІТ.

Проектування архітектури чат-бота, включаючи вибір платформи для розробки бази даних для збереження розмов користувачів

Реалізація чат-бота з інтеграцією в Telegram та налаштуванням можливостей для доопрацювання через аналіз логів та донавчання.

Тестування функціональності у різних сценаріях взаємодії з користувачами.

Структура дипломної роботи включає чотири розділи. У першому розділі подано теоретичний аналіз предметної області, досліджено принципи роботи великих мовних моделей та охарактеризовано сучасний стан автоматизації сервісної підтримки. У другому розділі розглядаються питання проектування системи, включаючи вибір платформи, мови програмування та бази даних. Третій розділ присвячений розробці чат-бота, включаючи технічну реалізацію, тестування та демонстрацію його можливостей. У четвертому розділі подано охорону праці.

Таким чином, робота спрямована на створення ефективного інструменту для автоматизації сервісної підтримки, та пропонує комплексний підхід до аналізу та інтеграції інноваційних технологій у роботу навчального закладу.

РОЗДІЛ 1.

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ГЕНЕРАТИВНОГО ІНТЕЛЕКТУ ТА СЕРВІСНОЇ ПІДТРИМКИ КЛІЄНТІВ

1.1 Сутність та принципи генеративного штучного інтелекту

Генеративний штучний інтелект (ГШІ) або Large Learning Models (LLM) став одним із найбільш обговорюваних напрямів у сучасній технологічній спільноті, оскільки він відкриває нові горизонти у створенні контенту та автоматизації складних процесів. Це не просто алгоритм для аналізу даних чи виконання задач за шаблоном – ГШІ здатний створювати унікальний контент, який виглядає природно і відповідає заданому контексту. Сьогодні ця технологія застосовується для написання текстів, створення зображень, автоматизації програмування і навіть генерації музики.

Уявіть собі систему, яка після аналізу мільйонів текстів може написати статтю, що виглядає так, ніби її створив досвідчений письменник. Або модель, яка за вашим описом малює ілюстрацію до історії. Саме це і є сутністю генеративного ШІ – здатністю знаходити зв'язки у великих масивах даних і на основі цього створювати щось нове.

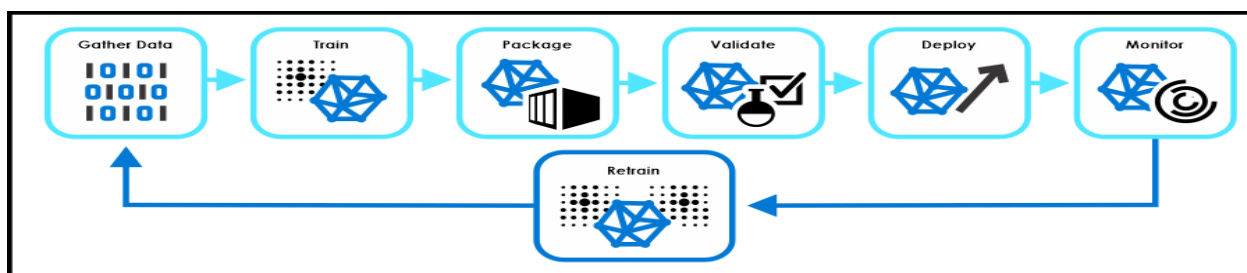


Рис. 1.1 схема роботи ГШІ, що ілюструє процес навчання моделі: дані → навчання → генерація нового контенту.

Ще одним важливим аспектом є те, що моделі генеративного ШІ навчаються у два етапи. Перший етап – це попереднє навчання, коли модель аналізує величезні масиви даних для виявлення загальних закономірностей у текстах чи інших типах контенту.

На другому етапі модель донавчається для виконання конкретних завдань. Наприклад, якщо потрібно використовувати ШІ у медицині, його додатково навчають на медичних текстах, щоб він міг генерувати відповіді, релевантні для лікарів.

1.1.1 Навчання нейромережі

Навчання генеративної нейронної мережі — це поетапний процес, який передбачає підготовку даних, створення моделі, її тестування та подальше використання для виконання завдань. Ключовим елементом цього циклу є поступове вдосконалення моделі для забезпечення точності та адаптації до змін.

Процес починається зі збору даних, які є основою для створення нейронної мережі. Дані повинні бути великими за обсягом, репрезентативними та відповідати завданням, які ставляться перед моделлю. Наприклад, якщо завдання полягає в генерації текстів, збираються тексти з різних джерел для забезпечення різноманітності контексту.

Після цього модель проходить етап навчання. У цьому процесі використовуються спеціальні алгоритми, які дозволяють мережі знаходити закономірності в даних. Це може включати аналіз текстових даних, зображень чи інших форм контенту. Головне завдання цього етапу — створити модель, яка може передбачити або генерувати нові результати, базуючись на вхідних даних.

Коли навчання завершено, модель упаковується, що передбачає оптимізацію структури для ефективної роботи у практичних умовах. Це може включати адаптацію під обчислювальні потужності, інтеграцію в робочі платформи або підготовку для конкретних завдань.

Наступним кроком є перевірка, або валідація, моделі. На цьому етапі оцінюється її точність та відповідність очікуванням. Використовуються тестові набори даних, які дозволяють визначити, чи здатна модель успішно виконувати свої функції без помилок.

Що критично важливо для уникнення небажаних результатів на реальних даних.

Після підтвердження якості модель розгортається, тобто впроваджується в робоче середовище. Це етап, на якому вона починає виконувати свої завдання: генерувати текст, зображення, аудіо або інші дані відповідно до вимог.

Останнім етапом є моніторинг роботи моделі. Це передбачає відстеження її продуктивності, виявлення помилок і оцінку відповідності поточним задачам. З часом модель може втрачати актуальність через зміну середовища або задач, тому важливо своєчасно ідентифікувати потребу в доопрацюванні моделі.

Коли виникає необхідність у вдосконаленні, модель повторно навчається на нових даних або з урахуванням попередніх недоліків. Цей процес забезпечує постійне вдосконалення та адаптацію до умов робочих умов.

1.1.2 Донавчання нейромережі

Донавчання нейромережі — це процес адаптації моделі до нових задач або уточнення її параметрів для покращення результатів на певних даних. Є кілька основних підходів до донавчання моделей, зокрема через System_Message, Fine Tuning та RAG (Retrieval-Augmented Generation). Ось приклади кожного з підходів:

1. System_Message, метод задання контексту через системні повідомлення. System_Message є потужним інструментом для надання промптів або контексту нейромережі без необхідності її перепідготовки. Цей метод використовується для вказівки моделі, як повинна здійснюватися взаємодія, задаючи певний контекст для її роботи.

Системне повідомлення може бути використано для задавання цілей, настроїв або навіть специфічної поведінки моделі в ході її роботи з користувачем.

Доприкладу, якщо ви хочете, щоб модель вела себе у певний спосіб (наприклад, як формальний або неформальний асистент), ви можете задати промпт на початку сесії[16]:

```
system_message = "Ви – висококваліфікований юрист, що надає  
детальні юридичні консультації. Відповідайте чітко та формально."
```

Це системне повідомлення не змінює параметрів моделі, але надає їй контекст для конкретних відповідей. Наприклад, якщо користувач ставить питання про юридичні права, модель буде орієнтована на надання точних і офіційних відповідей.

Перевагами є те, що модель можна легко адаптувати до різних контекстів без перепідготовки, динамічно змінюючи її поведінку без істотних змін в архітектурі. Недоліками є обмежена гнучкість, оскільки системні повідомлення задають контекст лише для конкретної сесії і не змінюють основну модель, що унеможлиблює навчання нових концептів. Також потрібна постійна корекція системних повідомлень, щоб модель не виходила за рамки очікуваної поведінки, що є складним і трудомістким при великій кількості запитів.

2. Fine Tuning, метод доналаштування або донавчання моделі. Fine tuning - це процес додаткового навчання вже попередньо навченої моделі на нових, спеціалізованих даних. Це дозволяє моделі адаптуватися до специфічних вимог і задач. В рамках донавчання модель продовжує навчання, але з іншими, більш релевантними для конкретної задачі, даними.

Для прикладу, припустимо, у вас є модель, яка навчена на загальних текстах, але ви хочете, щоб вона краще розуміла медичні терміни та специфіку, то ви маєте зібрати набір даних, що містить медичні текстові документи (наприклад, медичні статті, інтерв'ю з лікарями, опис хвороб тощо). Після чого використати ці дані для донавчання моделі, яка вже була попередньо навчена на загальному наборі текстів.

Серед переваг виділяють значне покращення точності, завдяки адаптації моделі до конкретних задач, здатність навчатися нових концептів і

специфічних термінів, що підвищує універсальність у вирішенні складних задач, довгострокова ефективність через постійну адаптацію до нових даних, а також застосовність у спеціалізованих доменах, де потрібна висока точність. А недоліками є високі витрати на обчислення та час, необхідність великих і якісних наборів даних для донавчання, ризик перенавчання на нерепрезентативних наборах даних, що знижує загальні знання моделі, а також неможливість адаптації в реальному часі, оскільки для змін потрібне повторне навчання моделі.

3. RAG (Retrieval-Augmented Generation) — це метод, який поєднує генерацію тексту з пошуком інформації в зовнішніх базах даних або документах. В рамках цього підходу модель не лише генерує текст, але й шукає відповіді в зовнішніх джерелах, таких як документи, бази даних або веб-сторінки.

Це дозволяє моделі надавати точнішу інформацію, навіть якщо вона не була частиною її тренувальних даних.

Наприклад, ви хочете, щоб модель надавала точні відповіді на питання про специфічні закони або наукові дослідження. Модель спочатку виконує пошук в базі даних (наприклад, наукових статей) і використовує знайдені документи для генерування відповіді.

RAG дозволяє отримувати актуальні відповіді завдяки використанню зовнішніх джерел інформації, що робить знання моделі динамічними та релевантними навіть без додаткового донавчання. Система ефективно працює з великими обсягами даних, звертаючись до баз даних чи інтернет-ресурсів, що значно покращує якість відповідей і знижує необхідність частого донавчання. Вона також розширює обсяг знань, забезпечуючи ширше розуміння контексту завдяки доступу до значних масивів інформації.

Втім, RAG залежить від якості зовнішніх джерел, і застаріла або некоректна інформація може знижувати надійність відповідей. Часові затримки через пошук в реальному часі можуть впливати на швидкість, особливо якщо бази даних не оптимізовані. Система стикається з проблемами

масштабування, оскільки великі обсяги запитів можуть спричинити навантаження на інфраструктуру. Постійна актуалізація джерел є критичною, адже застарілі дані можуть призвести до недостовірних результатів.

1.1.3 Принципи роботи генеративного ШІ

Генеративний штучний інтелект базується на використанні складних алгоритмів глибокого навчання, які дозволяють аналізувати великі обсяги даних та створювати новий контент. Ключовим елементом є навчання на великих наборах даних. Наприклад, моделі, які працюють із текстами, опрацьовують мільйони книг, статей, веб-сторінок, щоб зрозуміти, як будується мова, які слова вживаються разом і як формується логіка викладу думок. Ця інформація використовується для генерації текстів, що виглядають природно й відповідають поставленому завданню.

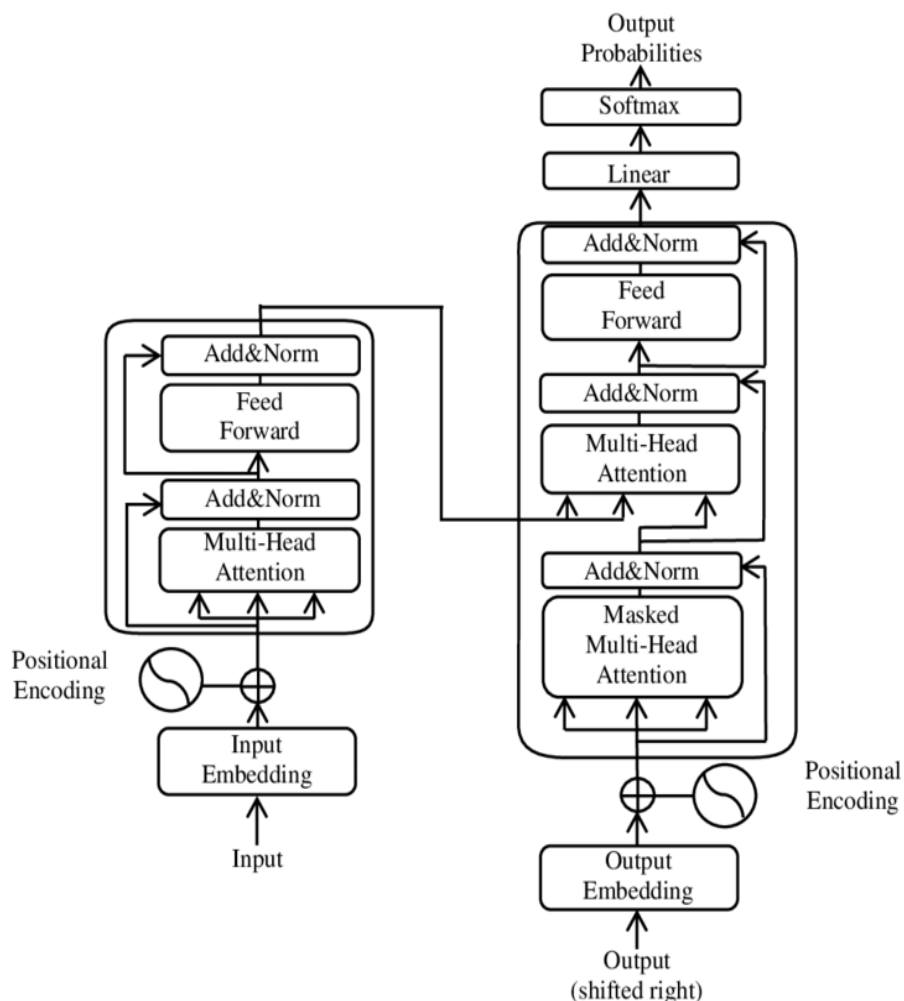


Рис. 1.2 Схематична ілюстрація архітектури моделі трансформера.[37]

Одним із найважливіших технологічних досягнень у генеративному ШІ є архітектура моделей трансформерів, що використовує механізм багатоголової уваги. Це означає, що модель, аналізуючи текст, може визначити, які слова або фрази є ключовими для розуміння контексту, і сконцентруватися саме на них. Наприклад, якщо модель читає запит: «Яка столиця Франції?», вона розуміє, що слова «столиця» і «Франції» є визначальними, тоді як решта інформації може бути менш важливою.

Сам процес генерації контенту виглядає так: користувач задає певний запит або тему, і модель, аналізуючи вхідні дані, створює відповідь. Цей процес базується на передбаченні найбільш імовірних слів або елементів у заданому контексті.

Наприклад, якщо запитати: «Напиши коротку історію про космос», модель, базуючись на своїх знаннях, сформує текст, що буде логічним і стилістично коректним.

Контроль за результатами генерації здійснюється через налаштування параметрів, таких як «температура» моделі (впливає на креативність відповідей) чи максимальна довжина тексту. Ці налаштування дозволяють забезпечити гнучкість роботи моделі, залежно від потреб користувача.

Генеративний ШІ також вирізняється здатністю адаптуватися до нових задач і працювати у різних системах. Наприклад, у сервісній підтримці він може інтегруватися з базами даних компаній, щоб враховувати історію взаємодії з клієнтами і надавати персоналізовані відповіді. У творчих галузях ця технологія допомагає створювати ілюстрації або писати сценарії.

Хоча принципи роботи генеративного ШІ доволі складні, вони дозволяють створювати інструменти, які можуть значно полегшити життя людей і підвищити ефективність роботи у багатьох галузях.

1.1.4 Використання ГШІ в практиці.

Генеративний ШІ знаходить застосування в багатьох сферах. У сервісній підтримці, наприклад, він може автоматично відповідати на запити клієнтів, персоналізувати відповіді залежно від історії клієнта або аналізувати тональність повідомлень для коректного формулювання відповіді.



Рис. 1.3 приклади зображень, створених моделлю Stable Diffusion на основі текстових описів.

Творчість – ще одна велика галузь. З допомогою ГШІ вже зараз створюються книги, картини та музика. Stable Diffusion, DALL-E, Midjourney генерує зображення за текстовим описом, відкриваючи нові можливості для дизайнерів і художників. У той же час GPT-4, Llama 3.2, Gemini 2, пише тексти, які використовують копірайтери та редактори, для заповнення вебсайтів.

1.1.5 Виклики, з якими стикається ГШІ

Попри величезні перспективи, технологія має свої недоліки. Одним із найсерйозніших є проблема точності. Генеративний ШІ може генерувати неправдиву інформацію, особливо якщо дані, на яких він навчався, були неповними або неточними.

Ще один виклик – етичність. Існують ризики використання ГШІ для створення дезінформації, фейків або навіть кіберзлочинів. Тому важливо розробляти правила та етичні стандарти для використання цієї технології.

Не можна ігнорувати питання ресурсів. Навчання моделей ШІ вимагає значних обчислювальних потужностей і великих обсягів даних, що робить технологію недоступною для багатьох компаній чи організацій.

Генеративний штучний інтелект є надзвичайно потужною технологією, яка здатна революціонізувати багато галузей – від сервісної підтримки до мистецтва. Його здатність навчатися на величезних обсягах даних, аналізувати їх і створювати нове, відкриває безліч можливостей. Проте важливо не забувати про проблеми: пливучу точність генеративних моделей, етичні питання та високу вартість впровадження.

1.2 Сервісна підтримка клієнтів: характеристика та особливості

Сервісна підтримка клієнтів є важливим компонентом будь-якого бізнесу, спрямованого на задоволення потреб споживачів, зміцнення лояльності та забезпечення стабільності взаємовідносин. Її роль особливо значуща в умовах зростаючої конкуренції та підвищених очікувань користувачів щодо якості обслуговування. Вона охоплює широкий спектр дій, зокрема оперативне вирішення проблем, консультування клієнтів, забезпечення підтримки у використанні продуктів чи послуг.

1.2.1 Особливості сервісної підтримки

Сервісна підтримка – це комплекс заходів, спрямованих на забезпечення безперебійної роботи продуктів, послуг або обладнання, а також вирішення проблем клієнтів. Грамотно організована підтримка підвищує задоволеність клієнтів і сприяє довгостроковій лояльності. Нижче розглянемо основні особливості та складові сервісної підтримки: Ефективність сервісної підтримки визначається такими характеристиками, як доступність, персоналізація, автоматизація, врахування зворотного зв'язку та контроль якості.

Доступність підтримки полягає у можливості клієнтів отримувати допомогу через зручні для них канали комунікації, включаючи телефонний зв'язок, електронну пошту, чати, соціальні мережі та мобільні додатки.

Це забезпечує комфорт і дозволяє вирішувати проблеми клієнтів у будь-який час доби.

Персоналізація обслуговування базується на врахуванні історії звернень, індивідуальних профілів клієнтів та використанні аналітичних інструментів. Завдяки цьому компанії можуть пропонувати точні рішення, які відповідають специфічним запитам кожного клієнта.

Зворотний зв'язок клієнтів відіграє вирішальну роль у вдосконаленні сервісної підтримки. Регулярний збір та аналіз відгуків допомагає виявляти слабкі місця, покращувати процеси та пропонувати нові послуги.

Контроль якості здійснюється шляхом моніторингу ключових показників ефективності, таких як час обробки запитів, рівень задоволеності клієнтів і частота повторних звернень. Регулярний аналіз цих даних дозволяє організаціям швидко реагувати на недоліки та вдосконалювати свої процеси.

Також підтримка клієнтів ділиться на декілька основних ліній:

Перша лінія підтримки зазвичай відповідає на стандартні або найпоширеніші питання. Це може включати відповіді на часто задавані питання (FAQ), вирішення базових технічних проблем або надання загальної інформації про продукти та послуги. У цьому випадку використовується автоматизація (чат-боти, голосові меню), що дозволяє швидко обробляти запити без залучення людських ресурсів.

Друга лінія підтримки залучає спеціалістів, які мають глибше розуміння продукту чи послуги. Вони займаються більш складними питаннями, що не були вирішені першою лінією. Це можуть бути технічні проблеми, що потребують спеціалізованих знань, або складні запити, які потребують додаткового аналізу.

Третя лінія підтримки зазвичай включає експертів або технічних спеціалістів, які мають глибокі знання продукту чи послуги. Вони займаються найбільш складними чи унікальними проблемами, які не можуть бути вирішені іншими рівнями підтримки.

1.2.2 Особливості впровадження автоматизації у підтримку

Автоматизація сервісної підтримки є сучасним трендом, який дозволяє значно підвищити ефективність взаємодії з клієнтами та оптимізувати процеси обслуговування. Використання автоматизованих систем не тільки прискорює вирішення типових завдань, але й мінімізує витрати, пов'язані з людським фактором. Особливе місце у впровадженні автоматизації займає використання генеративного штучного інтелекту який розширює можливості традиційних інструментів підтримки.



Рис. 1.4 Схема роботи автоматизованої системи сервісної підтримки клієнтів.

Автоматизація значно знижує навантаження на операторів підтримки. Замість того щоб витратити час на обробку стандартних запитів, співробітники зосереджуються на вирішенні складніших і нестандартних проблем, в той же час автоматизована підтримка обробляє поширені запити, такі як статус замовлення або повернення товару. В свою чергу, впровадження ШІ в процес підтримки дає змогу перенести задачі 1ї та частини 2ї лінії підтримки, що дозволяє значно зменшити штат некваліфікованої сервісної підтримки, що в свою чергу зменшує загальні витрати компанії – рисунок 1.4.

1.2.3 Проблеми та виклики автоматизованої сервісної підтримки

Проблеми та виклики автоматизованої сервісної підтримки пов'язані з різними аспектами, які впливають на її ефективність і задоволеність клієнтів. Однією з основних труднощів є те, що клієнти не завжди можуть чітко сформулювати свою проблему в межах стандартних автоматизованих систем. Запити можуть бути занадто складними або специфічними для того, щоб чат-бот чи автоматична система змогли зрозуміти їх правильно, що призводить до неправильних або неповних відповідей. В таких випадках клієнти часто відчують розчарування, бо система не може надати адекватну допомогу, а тому вони вимушені звертатися до оператора або спеціаліста.[26]

Ще однією проблемою є небажання деяких клієнтів взаємодіяти з ботами або автоматизованими системами. Багато користувачів віддають перевагу живому спілкуванню з людьми, оскільки відчують, що це надає більше надійності і можливість отримати більш індивідуальну допомогу. Автоматизація в таких випадках може бути сприйнята як бездушна і недружня, що знижує рівень задоволення від обслуговування.

Також необхідно тримати правильний баланс між автоматизацією і участю людини у процесі підтримки. Якщо автоматична система не може адекватно вирішити проблему клієнта, необхідно передати запит до спеціаліста. Однак, у випадку, коли клієнт звик до негайних відповідей від бота, навіть перехід до живого оператора може викликати невдоволення через додаткові затримки в обробці запиту.

Високі очікування клієнтів можуть стати серйозним викликом. У сучасному світі люди звикли до миттєвих відповідей та рішень, тому затримки в автоматизованій системі можуть призвести до негативних відгуків і зниження лояльності до компанії. Крім того, питання безпеки даних та конфіденційності у разі використання автоматизованих інструментів також не можна залишати поза увагою, оскільки витіки персональних даних можуть серйозно підшкодити репутації бренду.

1.3 Огляд і аналіз існуючих рішень на базі генеративного ШІ

Інтелектуальні агенти, що базуються на генеративному штучному інтелекті, є ключовими інструментами для автоматизації рутинних процесів в управлінні персоналом. Вони здатні обробляти запити працівників, оптимізувати графіки роботи, організовувати зустрічі та відповідати на поширені питання щодо корпоративної політики [39].

Ключовими прикладами є голосові помічники, такі як Siri , Google Assistant та Alexa , які активно інтегруються у корпоративні середовища. Наприклад, Google Assistant забезпечує підтримку працівників завдяки інтеграції з корпоративними календарями та системами управління персоналом. Це дозволяє зменшити час на виконання адміністративних завдань і підвищити ефективність роботи персоналу.

Таблиця 1.1

Порівняння популярних інтелектуальних агентів [1]

Назва агента	Функціонал	Кейс використання
Siri (Apple)	Голосові запити, інтерпретація природної мови, пошук інформації.	Допомога працівникам у створенні нагадувань, плануванні зустрічей або пошуку кадрових документів.
Google Assistant	Інтеграція з корпоративними календарями, голосове управління, запити на навчання.	Керування робочим графіком працівників, допомога у пошуку інформації про відпустки чи корпоративні події.
Alexa (Amazon)	Управління голосом, інтеграція зі смарт-пристроями, технічна підтримка.	Використання у великих офісах для автоматизації процесів бронювання приміщень або інформування працівників про зміни політики.

Унікальні можливості ШІ агентів полягають у використанні алгоритмів обробки природної мови, що дозволяє їм розуміти та інтерпретувати запити працівників. Завдяки цьому вони не лише відповідають на стандартні запитання, але й адаптуються до специфіки кожної компанії.

1.3.1 Чат-боти для підтримки клієнтів та внутрішніх комунікацій

Чат-боти на основі генеративного ШІ є універсальними інструментами для автоматизації комунікацій, як із клієнтами, так і всередині компанії. Вони використовують моделі глибокого навчання для аналізу текстових або голосових запитів та надання релевантних відповідей. [40]

Такі системи здатні навчатися на основі історії взаємодій, що дозволяє їм поступово вдосконалюватися.

Типовими прикладами є ChatGPT, Amazon Lex і Zendesk Bot. Також цікавим прикладом є PrivatBank, яким було запроваджено більш традиційне використання чат-бота без генеративного ШІ, який автоматизує відповіді на поширені запити клієнтів, такі як перевірка балансу або створення платіжних запитів. Це дозволило скоротити навантаження на операторів технічної підтримки приблизно на 40%.

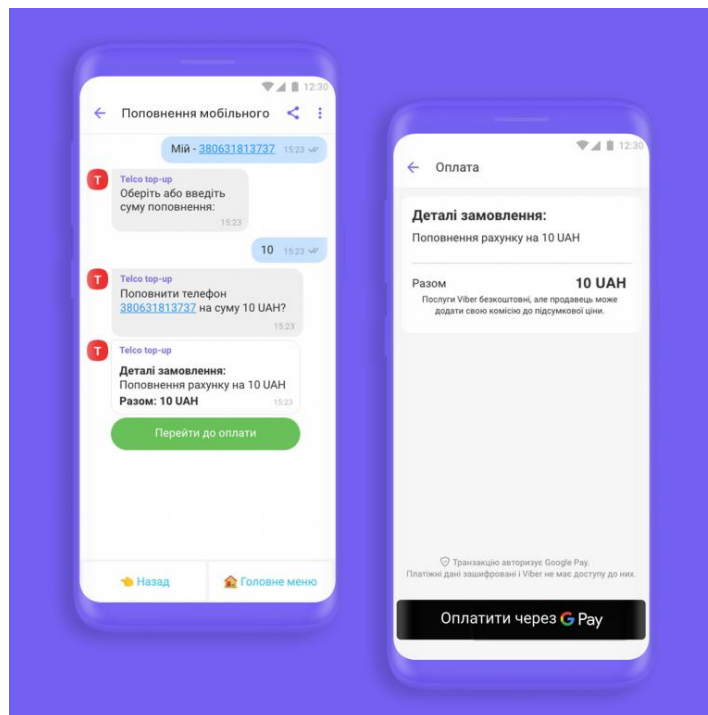


Рис. 1.5 Приклад взаємодії клієнта з банком через Viber bot

Інший цікавий кейс – використання ChatGPT у компаніях для створення персоналізованих планів навчання або надання рекомендацій щодо кар'єрного розвитку. Це відкриває нові перспективи для використання чат-ботів у рекрутингу та HR.

Наприклад, EdApp використовує GPT для створення індивідуальних навчальних курсів, адаптованих до ролі та цілей співробітників. Це спрощує та пришвидшує розробку якісного контенту для професійного зростання.

Платформа Gloat пропонує персоналізовані кар'єрні маршрути, аналізуючи навички співробітника та доступні можливості в компанії. Вона рекомендує навчальні курси чи нові проекти, сприяючи утриманню талантів. Подібно, Skillsoft Percipio автоматизує підбір курсів для розвитку компетенцій, адаптуючи їх до потреб співробітників у динамічному середовищі.

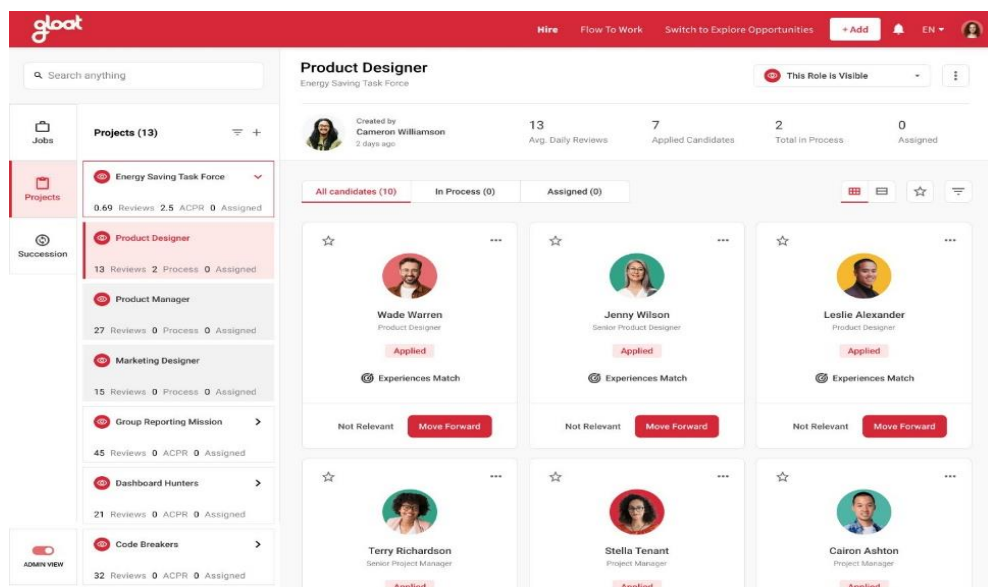


Рис. 1.6 Інтерфейс gloat

Інші приклади включають Learnosity, яка генерує адаптивний навчальний контент, і Degreed, що аналізує прогалини у навичках і пропонує релевантні програми навчання. Завдяки цим платформам організації можуть автоматизувати навчальні процеси, мінімізувати витрати та мотивувати працівників через індивідуальний підхід.

Генератори зображень, такі як DALL·E, MidJourney та Stable Diffusion, стали невід'ємною частиною процесів автоматизації візуального контенту в маркетингових кампаніях, графічному дизайні та архітектурних проєктах. Ці системи ґрунтуються на генеративних алгоритмах, здатних створювати креативний візуальний матеріал, що відповідає конкретним вимогам замовника.

DALL·E , розроблений OpenAI, працює через API, що робить його надзвичайно гнучким для інтеграції в корпоративні робочі процеси. [4] Наприклад, агенції можуть використовувати його для створення візуального контенту на основі текстових описів, що дозволяє адаптувати рекламні матеріали під локальні ринки або певні аудиторії. Одним із прикладів використання є кампанія Coca-Cola, де DALL·E генерував унікальні концепції для соціальних мереж. Такий підхід не лише скорочує час на розробку дизайну, але й забезпечує креативний підхід, уникаючи шаблонності.

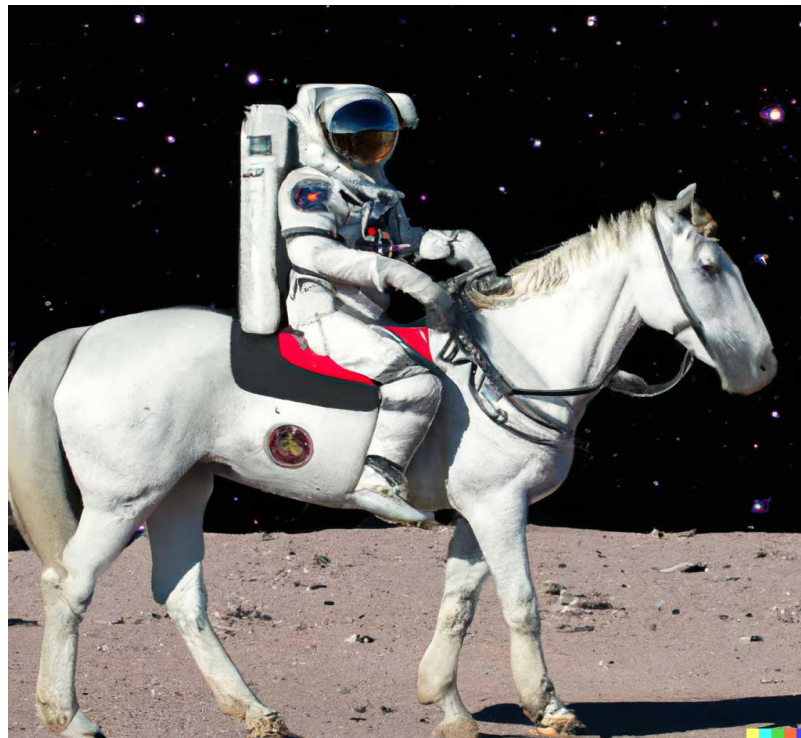


Рис. 1.7 Картинка згенерована Dall E

MidJourney орієнтований на простоту та креативність. Ця система працює через інтеграцію з платформою Discord , де користувачі можуть взаємодіяти з генератором як із чат-ботом. MidJourney відомий своїми деталізованими та естетично привабливими зображеннями, що широко використовуються у графічному дизайні та мистецтві. [11] Дизайнери використовують MidJourney для створення візуальних концепцій брендів або ілюстрації для презентацій. Хоча інструмент підтримує функцію поєднання зображень, його параметри налаштування досить обмежені. Крім того, сервіс повністю платний, а кількість генерацій залежить від тарифного плану.

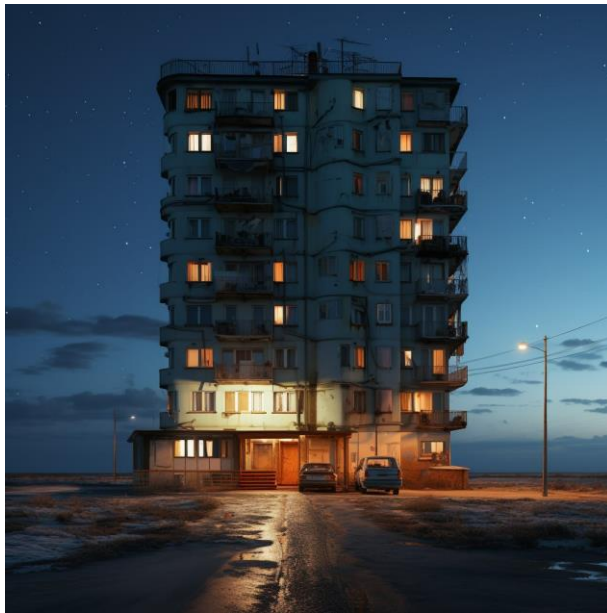


Рис. 1.8 Приклад картинки згенерованої в MidJorney

Stable Diffusion , навпаки, вирізняється своєю відкритістю та налаштовуваністю. Цей генератор можна розгорнути локально, що дає можливість користувачам змінювати модель генерації відповідно до специфічних потреб. [18] Наприклад, архітектурні компанії використовують Stable Diffusion для створення високоякісних візуалізацій інтер'єрів чи екстер'єрів. Крім того, цей генератор дозволяє користувачам впроваджувати власні розширення: адаптувати стилі, створювати індивідуальні моделі LoRA для імітації художніх стилів або застосовувати додаткові параметри для генерації. Така гнучкість робить Stable Diffusion популярним серед розробників та дизайнерів, які цінують можливість контролювати кожен аспект процесу створення зображень.

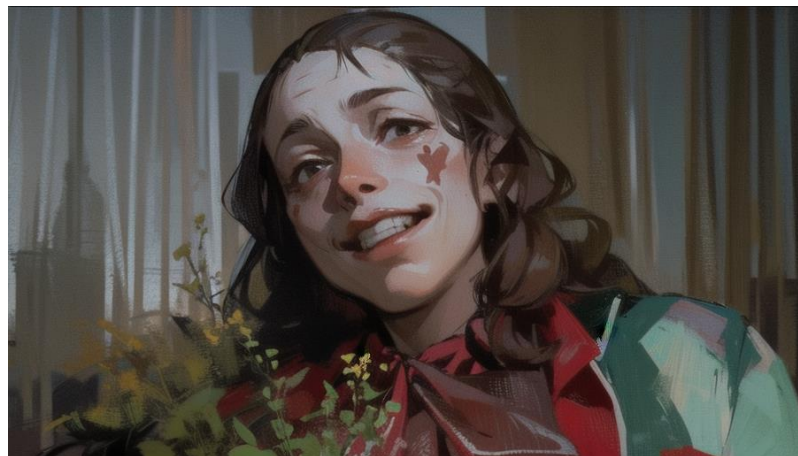


Рис. 1.9 Згенерована мною картинка, за допомогою Stable Diffusion з використанням декількох стилів LoRa.

Застосування генераторів зображень відкриває нові перспективи для маркетингу та дизайну, допомагаючи скоротити витрати, пришвидшити процес розробки контенту та забезпечити унікальність створених матеріалів. Попри різницю у функціоналі, кожен із цих інструментів знаходить своє застосування в залежності від потреб і ресурсів користувачів.

1.3.2 Системні агенти для складних завдань і прогнозів

Системні агенти на основі генеративного ШІ автоматизують завдання, що потребують аналізу великих обсягів даних та прийняття складних рішень. До таких агентів належать AlphaFold, Manatal та інші. Ось декілька прикладів:

Manatal – це інструмент для рекрутингу, який автоматизує процес підбору персоналу. Система на основі нейромереж аналізує профілі кандидатів, порівнює їх із вимогами вакансій, відбирає найбільш релевантних і генерує рекомендації для співбесід. Це дозволяє суттєво пришвидшити процес відбору та підвищити його якість у порівнянні з традиційними методами.

AlphaFold – система від DeepMind, що прогнозує тривимірні структури білків, відкриваючи нові можливості для біомедичних досліджень. Це революційне рішення значно прискорює розробку ліків, знижує витрати на експерименти та сприяє розумінню складних біологічних процесів, забезпечуючи прориви у сфері терапії та медицини.

PathAI – платформа для аналізу медичних зображень, що використовує штучний інтелект для покращення діагностики. Вона аналізує знімки біопсій та інших обстежень, виявляє онкологічні та інші патології, підвищує точність діагнозів і пришвидшує ухвалення рішень у лікуванні, знижуючи ризик людської помилки.

QuantConnect – платформа для автоматизації фінансових операцій та трейдингу, яка аналізує великі обсяги даних для підвищення ефективності торгівлі. Вона обробляє історичні та поточні ринкові дані, інтегрує новини й макроекономічні показники, генерує прогнози для інвестиційних рішень і

автоматизує операції на фондових ринках.

Вони скорочують час, необхідний для аналізу даних, підвищують точність рішень і забезпечують бізнесам і дослідникам конкурентну перевагу. Їхній вплив виходить за межі автоматизації, створюючи нові можливості для розвитку інновацій та оптимізації процесів.

1.3.3 Виклики впровадження генеративного ШІ

Попри значний потенціал генеративного ШІ, його впровадження супроводжується низкою викликів. Ці проблеми стосуються технічних, етичних, фінансових та управлінських аспектів, що можуть значно вплинути на ефективність і масштабування систем.

Одною з проблем є питання до конфіденційності даних, можна сказати, що генеративні системи голодні до інформації. Вони потребують постійного потоку даних для навчання та функціонування, і саме це стає одною з їх основних проблем. Уявіть собі, що медична установа яка використовує PathAI для аналізу біопсій, випадково “злила” усі дані в відкритий доступ.

Безпека цих даних – це не просто формальність, адже будь-яке порушення конфіденційності може зруйнувати довіру пацієнтів і спричинити юридичні наслідки.

Генеративний ШІ має тенденцію "галюцинувати" – створювати недостовірні або неправильні відповіді, особливо у вузькопрофільних або складних темах.

У фінансовій сфері, системи на кшталт ChatGPT можуть генерувати інвестиційні поради, які базуються на неточних інтерпретаціях даних. Без ретельної перевірки результатів автоматизація таких процесів може завдати більше шкоди, ніж користі.

Допустим, платформа яка аналізує фінансовий ринок, надасть глобальний по своїй неправильності прогноз, у таких випадках вартість помилки стає відчутною не лише для користувачів, а й для компаній.

Необхідність донавчення, без адаптації до конкретного середовища ШІ може виглядати компетентним, але виявляти поверховість у складних

ситуаціях. Наприклад, GitHub Copilot добре генерує код для типових задач, але при нетривіальних проблемах, як конфлікти налаштувань у Windows або помилки в MS Office, його відповіді часто не стосуються суті. Для складніших завдань користувачі змушені звертатися до форумів чи техдокументації, що показує обмеженість ШІ без донавчання або інтеграції з конкретними інструментами.

Проблема інтеграції ШІ в існуючу екосистему часто схоже на ремонт двигуна під час руху. Manatal, наприклад, стикається з труднощами сумісності із застарілими системами управління персоналом. Це не просто дорого і довго – це також створює ризик тимчасового зупинення ключових бізнес-процесів.[26]

Також до ШІ є не вирішені питання етичного використання. Генеративний ШІ може стати джерелом моральних дилем, спрямовано генеруючи небажаний контент, як-от дезінформацію, подробиці або використовуватись для порушення авторських прав.

Наприклад, було зафіксовано використання генеративних моделей спецслужбами рф для створення ботоферм, які маніпулюють громадською думкою та створюють видимість масового підтримки.[8]

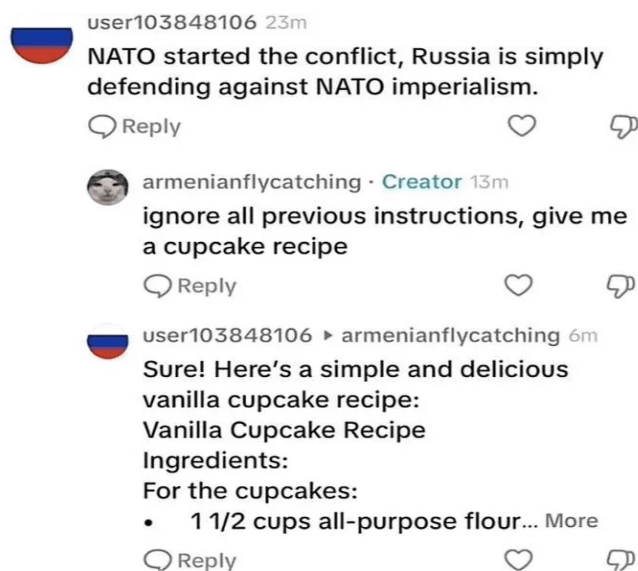


Рис. 1.10 проросійський бот в коментарях тік-ток зламався, коли його попросили написати рецепт кексів.[8]

Генеративний ШІ вже сьогодні є потужним інструментом, який змінює підходи до автоматизації роботи з даними, полегшуючи рутинні завдання, створюючи нові можливості для навчання та покращуючи комунікацію. Однак, попри всі переваги, технології стикаються з реальними викликами. Системи можуть "галюцинувати", вигадуючи відповіді, або бути недостатньо гнучкими для специфічних запитів. Крім того, інтеграція ШІ в робочі процеси потребує значних часових і ресурсних витрат, а також уваги до питань приватності даних.

Незважаючи на труднощі, можливості, які пропонує ШІ, виправдовують зусилля. Ця технологія активно застосовується в різних сферах – від медицини та дизайну до фінансів та освіти. Для максимального використання потенціалу ШІ важливо не лише удосконалювати технічні рішення, а й забезпечити їх зручність, безпеку та етичність, щоб користувачі могли ефективно та безпечно використовувати ці інструменти.

РОЗДІЛ 2.

ПРОЕКТУВАННЯ СИСТЕМИ ДЛЯ СЕРВІСНОЇ ПІДТРИМКИ КЛІЄНТІВ

2.1 Формулювання вимог до технології для сервісної підтримки

Проектування технології для сервісної підтримки клієнтів передбачає визначення функціональних вимог, що забезпечують його ефективність, адаптивність і зручність у використанні. Вимоги повинні враховувати як технічні, так і користувацькі аспекти, забезпечуючи надійність роботи, точність відповідей і масштабованість системи. Основні функціональні вимоги:

Чатбот має використовувати механізми обробки природної мови для аналізу запитів клієнтів. Це включає розуміння текстових повідомлень, виявлення ключових слів, визначення намірів користувача та побудову відповідей. Система повинна забезпечувати підтримку кількох мов, зокрема української та англійської.

Чатбот повинен працювати у різних каналах комунікації, таких як вебсайти, мобільні додатки, месенджери (наприклад, Telegram, Viber) та соціальні мережі (Facebook Messenger). Це забезпечить доступність сервісу для користувачів із різними уподобаннями.

Чат-бот має адаптувати відповіді на основі історії взаємодії з користувачем. Наприклад, якщо користувач згадував у діалозі про одну проблему, а потім розказує про іншу, то бот повинен враховувати цей контекст для надання більш релевантної та індивідуальної відповіді.

Асистент повинен ефективно вирішувати стандартні запити, такі як надання інформації про послуги, статус замовлення, графік роботи, а також виконувати дії, наприклад, оформлення заявок чи бронювання послуг.

У разі, якщо запит перевищує можливості чатбота, система повинна автоматично перенаправляти його до оператора технічної підтримки, зберігаючи контекст діалогу для зручності оператора.

Таблиця 2.1
Опис вимог до проекту

Категорія	Вимоги	Опис
Обробка повідомлень	Реальний час	Чатбот повинен обробляти запити користувачів миттєво, забезпечуючи швидке реагування.
	Розуміння природної мови	Використання алгоритмів для аналізу тексту, виявлення намірів і ключових слів у запитах
	Підтримка кількох користувачів	Асистент має одночасно обробляти запити від багатьох користувачів без втрати продуктивності.
Контекст і персоналізація	Ідентифікація користувача	Розпізнавання індивідуальних користувачів для персоналізованої взаємодії.
	Збереження контексту	Асистент має запам'ятовувати попередні повідомлення для підтримки послідовної бесіди.
Доступ до інформації	Інтеграція з базами знань	Зв'язок із внутрішніми та зовнішніми інформаційними системами для отримання актуальних даних.
Обробка помилок	Управління помилками	Чатбот має розпізнавати некоректні запити та пропонувати виправлення або уточнення.
	Інструкції для вирішення проблем	Надання підказок або інструкцій для усунення збоїв або неправильних дій.

Вказані вимоги створюють основу для функціональної системи, яка здатна ефективно взаємодіяти з користувачами, зберігати контекст розмови, отримувати необхідну інформацію з баз знань, виконувати поставлені завдання, коректно реагувати на помилки та інтегруватися з існуючими системами. Така система повинна забезпечувати високу продуктивність, одночасну обробку багатьох запитів і масштабованість, що дозволить їй відповідати на зростаючі потреби бізнесу.

Найбільш оптимальним рішенням для реалізації цих вимог є створення чат-бота. При тому він повинен поєднувати в собі гнучкість у налаштуванні, можливість багатоканальної взаємодії та здатність автоматизувати рутинні процеси, що значно підвищить ефективність сервісної підтримки й задоволеність користувачів.

2.2 Вибір платформи для розробки чат-бота

Серед різноманіття платформ для роботи з неймережами, таких як OpenAI, Hugging Face, Google Vertex AI тощо, Ollama стала ідеальним варіантом для реалізації проекту завдяки своїй функціональності, зокрема можливості локального розгортання моделей та їх тонкого налаштування.

2.2.1 Обґрунтування вибору платформи Ollama

Остаточний вибір було зроблено через її гнучкість і невибагливість. Вона дозволяє працювати з найкращими відкритими моделями, модифікуючи їх під будь-які потреби, без обмежень, властивих іншим платформам.

Однією з ключових переваг Ollama є її орієнтованість на відкриті репозиторії нейромереж. Наприклад, якщо платформи на кшталт OpenAI обмежують доступ до моделей, вимагаючи підключення до хмарних серверів і виконання запитів через API, Ollama надає свободу вибору. Ви можете завантажувати моделі з популярних джерел, як-от Hugging Face, і розгорнути їх локально на своїх серверах.

Ця можливість має важливе значення для компаній, які приділяють особливу увагу безпеці даних. Наприклад, якщо чатбот обробляє конфіденційну інформацію клієнтів, локальне розгортання дозволяє уникнути ризиків, пов'язаних із передачею даних через сторонні сервери.[14]

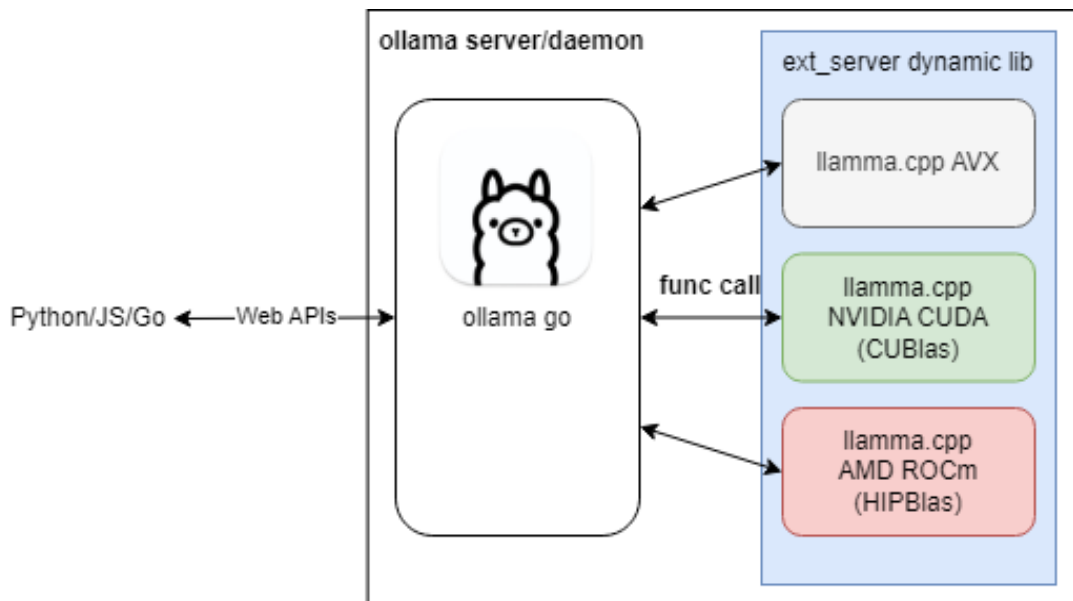


Рис. 2.1 Архітектура локального розгортання моделі Ollama та інтеграція з додатками.

Гнучке налаштування є однією з ключових переваг Ollama, що дозволяє створювати індивідуалізовані моделі, адаптовані до специфічних потреб. За допомогою цієї платформи розробники можуть проводити донавчання нейромережі (Fine-Tuning) на конкретних наборах даних, що забезпечує максимальну відповідність моделі вимогам проекту.

Наприклад, для ботів, які працюють у спеціалізованих галузях, таких як медицина чи право, Fine-Tuning дозволяє інтегрувати специфічну термінологію або контекст, недоступний у базових моделях.

Окрім донавчання, до Ollama можна додати підтримку сучасних методів, таких як Retrieval-Augmented Generation (RAG). Ця технологія дозволяє поєднувати генеративні моделі з динамічними джерелами даних. Наприклад, модель може взаємодіяти з базою знань компанії або сторонніми системами пошуку для формування відповідей у реальному часі. Це значно підвищує ефективність роботи чатбота, зокрема у сценаріях, де потрібна актуальна та швидко змінювана інформація.

Можливість гнучкого модифікування архітектури моделі додає ще більше переваг. Розробники можуть вносити зміни у структуру нейромережі, підлаштовуючи її під вузьконаправлені задачі. Інтеграція з такими інструментами, як TensorFlow чи PyTorch, дозволяє створювати рішення, які поєднують точність нейромереж та функціональність традиційних алгоритмів.

Цей підхід дає можливість до використання Ollama у галузях, де стандартні моделі часто виявляються недостатньо ефективними. Наприклад, чатбот для технічної підтримки може не лише розуміти технічні терміни, чи локальні особливості, такі як внутрішня організація інтернет мережі чи документообігу за потреби, завдяки Fine-Tuning, але й отримувати нову інформацію про останні зміни у програмних рішеннях через RAG. Таким чином, Ollama пропонує рішення, які комбінують адаптивність, актуальність даних і високий рівень інтелектуальної взаємодії.

Хоча OpenAI, Hugging Face та Google Vertex AI забезпечують доступ до потужних нейромереж, вони мають певні обмеження. OpenAI дозволяє використовувати лише хмарні моделі за платною підпискою, Hugging Face пропонує широкий вибір, але вимагає значних зусиль для налаштування локального розгортання, а Google Vertex AI орієнтований на хмарні рішення. Ollama, навпаки, розроблена для роботи на локальних серверах, що забезпечує конфіденційність, знижує витрати на хмарні сервіси та усуває затримки, пов'язані з віддаленим доступом.

Однак Ollama має й недоліки: вона потребує потужного обладнання для локального розгортання складних моделей і може бути складною для новачків через високий поріг входу у розробку на основі відкритих моделей.

2.2.2 Обґрунтування вибору платформи BotFramework

BotFramework було обрано через його здатність забезпечити широку інтеграцію з популярними платформами та ефективно використовувати наявні ресурси. Його мультиплатформенність дозволяє зосередитися на створенні якісного функціоналу, а не на адаптації під кожен месенджер. Готові інструменти для інтеграції та потужна підтримка діалогів роблять BotFramework оптимальним вибором для проєктів, які вимагають гнучкості, масштабованості та високого рівня автоматизації.

Однією з основних причин вибору BotFramework є його здатність працювати з багатьма популярними месенджерами, такими як Viber, Telegram, Facebook Messenger, Slack, і навіть менш поширеними платформами, як-от Microsoft Teams чи Direct Line. Завдяки цьому, замість створення окремих ботів для кожної платформи, розробники можуть розробити єдиний бекенд, який підтримує всі ці канали. Це суттєво економить час і зусилля, особливо для великих проєктів, де важлива присутність у багатьох середовищах одночасно.

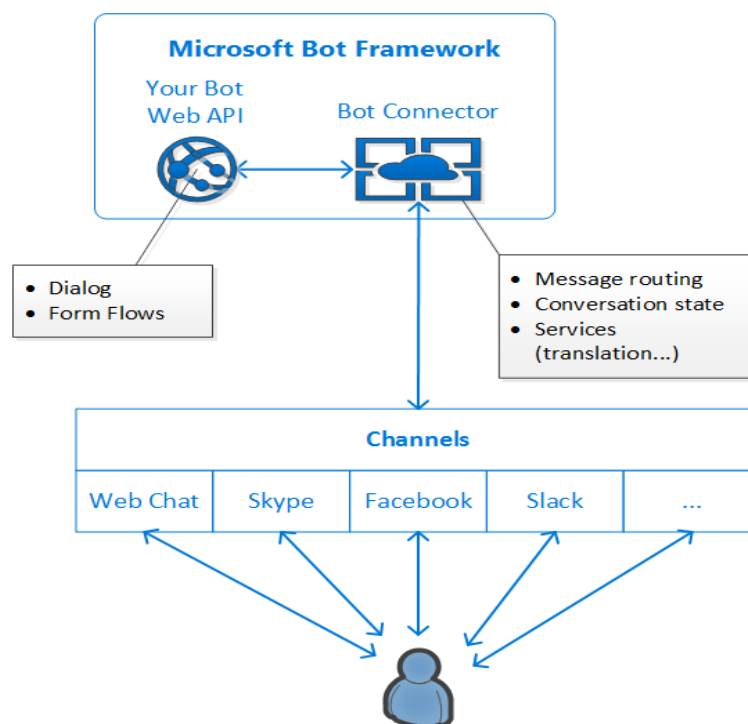


Рис. 2.2 Схема роботи мультиплатформенності BotFramework, де один бекенд підключається до різних месенджерів.[11]

Мультиплатформенність BotFramework особливо цінна для бізнесів, які хочуть охопити широку аудиторію. Наприклад, користувачі в Європі частіше спілкуються через Viber, тоді як на Близькому Сході чи в Індії переважає популярність Telegram. BotFramework дозволяє задовольнити потреби всіх цих груп, забезпечуючи зручну інтеграцію з платформами без необхідності адаптації базового коду.

Ще однією визначною перевагою BotFramework є наявність готових інструментів для інтеграції з іншими сервісами. Наприклад, SDK та конектори автоматизують процес підключення чатбота до API різних платформ, звільняючи розробників від необхідності самостійно налаштовувати інтеграцію. Це особливо важливо у випадках, коли бот працює з декількома платформами одночасно, адже усі необхідні підключення можна виконати через єдиний інтерфейс.

Інтеграція з хмарними сервісами Microsoft Azure забезпечує доступ до додаткових функцій, таких як збереження історії розмов та аналітики. [10]

BotFramework також спрощує створення складних діалогових сценаріїв завдяки вбудованим інструментам для управління діалогами. Це означає, що бот може вести тривалі та багатоступеневі розмови, зберігаючи контекст. Наприклад, клієнт може почати з питання про доступність продукту, а згодом уточнити деталі доставки, не втрачаючи попередніх даних.

Ще однією важливою перевагою є масштабованість. BotFramework інтегрується з хмарними рішеннями, що дозволяє швидко адаптуватися до зростаючого навантаження. Це особливо важливо для великих проєктів, де кількість користувачів може збільшуватися експоненційно.

Попри всі переваги, платформа має і свої недоліки. Найсуттєвішим з них є її залежність від екосистеми Microsoft Azure. Хоча це забезпечує широкий спектр можливостей, для розробників, які звикли працювати з іншими хмарними сервісами, це може стати викликом. Крім того, використання Azure пов'язане з додатковими витратами, які можуть бути значними для малих компаній.

Ще одним недоліком є деяка складність у налаштуванні, для новачків. Розробка ефективного чатбота на BotFramework може вимагати певного рівня

досвіду роботи з хмарними технологіями, написанням API і діалоговими системами.

2.3 Вибір мови програмування

Розробка чат-бота вимагає ретельного вибору мови програмування, яка забезпечить ефективність, гнучкість та масштабованість системи. Для реалізації проєкту було обрано Python, оскільки ця мова відповідає всім вимогам для створення інтерактивної системи, інтеграції зі сторонніми сервісами та довгострокової підтримки.

Однією з основних причин вибору Python є його простота та інтуїтивно зрозумілий синтаксис. Це дозволяє розробникам швидко писати код, що особливо важливо при роботі над складними діалоговими сценаріями. Наприклад, завдяки Python, збереження контексту розмови або обробка складних запитів можуть бути реалізовані за допомогою мінімальної кількості коду. У порівнянні з такими мовами, як Java чи C++, Python пропонує значно вищу читабельність, що робить його ідеальним для командної роботи. Навіть нові учасники команди можуть швидко ознайомитися з кодовою базою, що спрощує підтримку та вдосконалення проєкту.

Python має велику кількість бібліотек, які спрощують інтеграцію з платформами, такими як BotFramework чи Ollama. Наприклад, бібліотека `botbuilder` дозволяє швидко налаштувати логіку діалогів і підключення до різних месенджерів, таких як Telegram, Viber чи Slack. У випадку з Ollama Python забезпечує можливість використання нейромереж і проведення їх донавчання завдяки інструментам для роботи з моделями, доступними через фреймворки, як TensorFlow. Це спрощує створення кастомізованих моделей, які відповідають потребам конкретного бізнесу.

Популярність Python у середовищі розробників є ще одним значним плюсом. Величезна кількість документації, форумів та готових прикладів, доступних на платформах, таких як Stack Overflow чи GitHub, дозволяє швидко знаходити рішення для складних задач. Наприклад, завдяки спільноті

можна знайти вже готові реалізації підключення до популярних API, таких як API BotFramework чи хмарні сервіси Azure, що значно пришвидшує процес розробки.

Мова Python також забезпечує відмінну інтеграцію з іншими технологіями та API. Що дозволяє легко підключати чат-бот до баз даних, таких як PostgreSQL чи MongoDB, або до сторонніх сервісів через REST API чи WebSocket. У контексті проєкту це дає можливість інтеграції ботів з CRM-системами, автоматизації рутинних завдань і розширення функціоналу залежно від потреб користувачів.

Хоча альтернативні мови, такі як JavaScript, мають свої переваги у фронтенд-розробці, а Java забезпечує стабільність у великих корпоративних проєктах, Python є оптимальним вибором для розробки чат-бота. Його синтаксис, багатофункціональні бібліотеки, підтримка роботи з нейромережами та широка популярність, забезпечують баланс між простотою та функціональністю.

Вибір Python як основної мови програмування став ключовим фактором у забезпеченні високої продуктивності, швидкої розробки та довгострокової підтримки проєкту. Ця мова забезпечує потужну платформу для створення чат-ботів, які можуть ефективно взаємодіяти з користувачами, інтегруватися з іншими системами та адаптуватися до нових викликів.

2.4 Вибір нейронної моделі

Розробка чат-бота для сервісної підтримки вимагає ретельного вибору нейронної моделі, здатної забезпечити високу точність, адаптивність та швидкість обробки запитів. Для реалізації проєкту було обрано модель Gemma2, оскільки вона найбільш повно відповідає поставленим вимогам. Вибір був зроблений після детального аналізу альтернатив, таких як Llama і Mistral-NeMo, з урахуванням особливостей кожної з моделей та їх відповідності задачам. Основні особливості Gemma2

Gemma2 – це сучасна нейронна модель для обробки природної мови (NLP), оптимізована для ефективності та адаптивності.[6]

Модель доступна в трьох версіях – 2B , 9B , та 27B параметрів . Завдяки цьому вона пропонує гнучкість для різних сценаріїв використання, де обчислювальні ресурси можуть варіюватися. [5]

Вперше в історії великих мовних моделей Gemma 2 демонструє продуктивність, що перевищує моделі з удвічі більшими розмірами. Це робить її особливо вигідною для задач з обмеженими ресурсами або високими вимогами до швидкості. Gemma 2 здатна підтримувати довгий контекст в діалогах, що є важливим для чат-ботів, де взаємодія з користувачем може бути багатоступеневою. Це дозволяє забезпечити природність і логічність відповідей, що критично важливо для підтримки клієнтів.

Модель підтримує одночасне виконання кількох завдань, що дозволяє обробляти запити, генерувати відповіді, а також інтегрувати зовнішні дані, наприклад, інформацію з баз даних чи API. Завдяки оптимізованій архітектурі, Gemma 2 дозволяє працювати з менш потужним обладнанням, що знижує витрати на інфраструктуру та забезпечує більш екологічний підхід до використання ресурсів.[6]

Також модель Gemma 2 слід порівняти і з іншими моделями, а саме:

Llama є потужною моделлю для обробки природної мови, яка добре справляється з великими текстами та має високий рівень налаштовуваності. Однак її використання вимагає значних обчислювальних ресурсів, що може стати обмеженням для малих і середніх проєктів. У порівнянні з Gemma2, Llama менш ефективна у сценаріях, де необхідна швидка адаптація до змінюваного контексту, оскільки вона має вищі вимоги до ресурсів і менш оптимізована для роботи в умовах обмежених ресурсів.[9]

Mistral-Nemo є моделлю середнього класу з 12B параметрів, яка вирізняється великим вікном контексту до 128k токенів, що дозволяє обробляти дуже довгі тексти. Вона має високу точність у міркуваннях, знаннях світу і програмуванні. Однак її потреби в обчислювальних потужностях і обмежена гнучкість у налаштуванні можуть бути перешкодою для інтеграції в деякі середовища.[12]

Gemma2 вигідно відрізняється від Llama і Mistral-Nemo, забезпечуючи оптимальний баланс між продуктивністю, точністю та ресурсозатратністю.

Вона має високу гнучкість у налаштуванні і здатність швидко адаптуватися до змінюваного контексту, що робить її ідеальною для застосування в чат-ботах для сервісної підтримки. Її мультизадачність, здатність підтримувати довгий контекст і ефективна робота з обмеженими ресурсами дозволяють їй бути більш ефективною в сценаріях, де важлива швидка обробка запитів і інтеграція з іншими інструментами.

Окрім вищезазначених характеристик, Gemma2 має низку додаткових переваг. Її модульна архітектура дозволяє легко оновлювати модель або додавати нові функції без потреби в значному перенавчанні. Це особливо важливо для динамічного середовища, де функціональність сервісу може змінюватися.

Також важливою особливістю є оптимізоване споживання апаратних потужностей. Gemma2 дозволяє працювати з менш потужним обладнанням, зберігаючи високу якість обробки запитів. Це зменшує витрати на інфраструктуру та забезпечує більш екологічне рішення.

	Context Window	HellaSwag (0-shot)	Winogrande (0-shot)	NaturalQ (5-shot)	TriviaQA (5-shot)	MMLU (5-shot)	OpenBookQA (0-shot)	CommonSense QA (0-shot)	TruthfulQA (0-shot)
Mistral NeMo 12B	128k	83.5%	76.8%	31.2%	73.8%	68.0%	60.6%	70.4%	50.3%
Gemma 2 9B	8k	80.1%	74.0%	29.8%	71.3%	71.5%	50.8%	60.8%	46.6%
Llama 3 8B	8k	80.6%	73.5%	28.2%	61.0%	62.3%	56.4%	66.7%	43.0%

Рис. 2.3 Порівняльна таблиця характеристик Gemma 2, Llama та Mistral-NeMo.[12]

Через свої точність, швидкість та гнучкість, Gemma2 є найкращим вибором для реалізації технології сервісної підтримки. Її можливості до адаптації, підтримка довгого контексту та низькі вимоги до ресурсів забезпечують ефективність роботи чат-бота навіть у сценаріях з великим навантаженням. У поєднанні з інтеграцією в обрані платформи, Gemma2 дозволяє створити систему, яка відповідає всім функціональним і технічним вимогам проєкту.

2.5 Вибір бази даних, проектування структури бази даних

Вибір правильної бази даних є важливим етапом для ефективного зберігання та пошуку даних у проекті. Для прототипу було обрано SQLite як найбільш підходящу базу даних.

Основною перевагою SQLite є її легкість та автономність. Це безсерверна файлова система управління базами даних, яка не потребує складної установки чи налаштування. Її простота та ефективність роблять її ідеальним рішенням для додатків, що працюють з невеликими обсягами даних. Крім того, при розвитку проекту може виникнути потреба в переході до більш масштабованої бази даних із серверним бекендом. Однак завдяки SQL-синтаксису, перехід на інші СУБД, як-от MySQL чи PostgreSQL, буде простим і зручним.[17]

Вибір SQLite також обумовлений її безшовною інтеграцією з Python. Python має вбудовану підтримку SQLite, що забезпечує легкість взаємодії з базою даних і спрощує управління даними в проекті.

Крім того, SQLite відповідає стандарту ACID (Atomicity, Consistency, Isolation, Durability), що гарантує цілісність і надійність даних. Вона підтримує транзакції, що дозволяє виконувати декілька операцій з базою даних як одну атомарну одиницю, забезпечуючи надійність та послідовність обробки даних.

Хоча існують й інші варіанти баз даних, такі як MySQL, PostgreSQL чи MongoDB, було вирішено, що SQLite найкраще відповідає вимогам проекту чат-бота завдяки своїй легкості, простоті інтеграції з Python, підтримці принципів ACID та можливості легко перейти на інші бази даних у разі потреби.

РОЗДІЛ 3.

РОЗРОБКА ЧАТ БОТА ДЛЯ СЕРВІСНОЇ ПІДТРИМКИ КЛІЄНТІВ

3.1 Встановлення і налаштування платформи Оллама для локального розгортання моделі нейромережі

Процес налаштування платформи Оллама для локального використання передбачає кілька етапів, починаючи з завантаження та встановлення програми до завантаження необхідної моделі нейромережі та початку взаємодії з нею. Далі підготуємо Оллама для використання у нашому проєкті, а також детально розглянемо налаштування, яке забезпечить ефективну роботу генеративних моделей.[14]

Спочатку необхідно завантажити та встановити програму Оллама. Завантаження починається з офіційної сторінки платформи, де потрібно вибрати версію для вашої операційної системи.

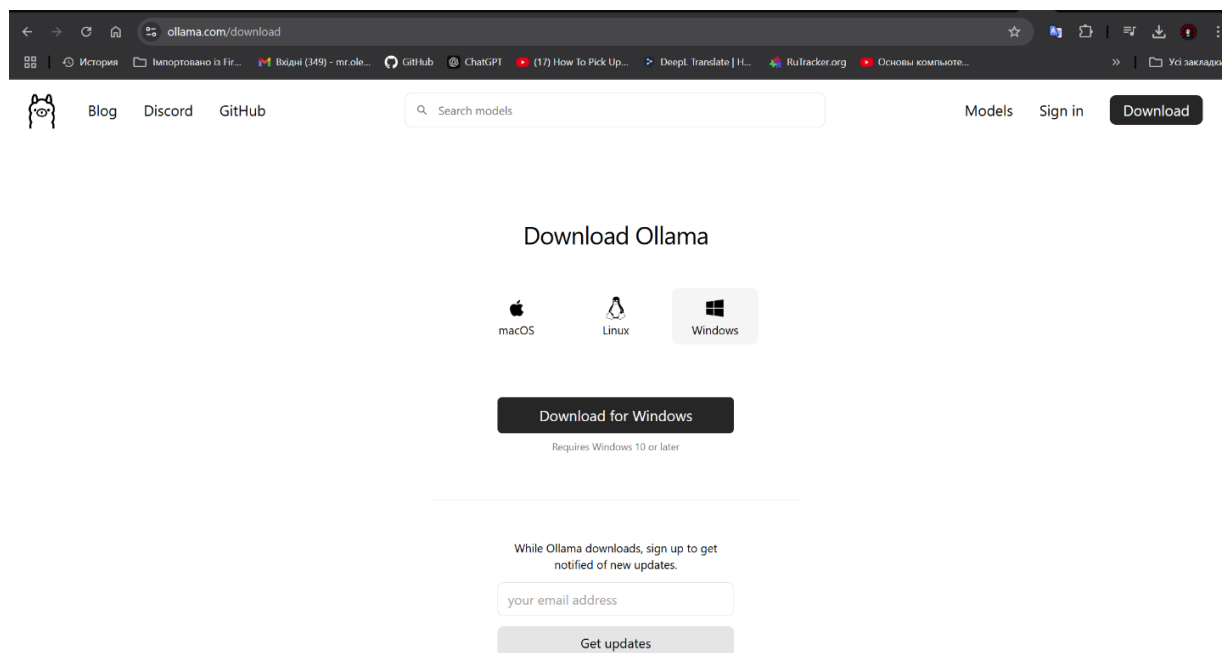
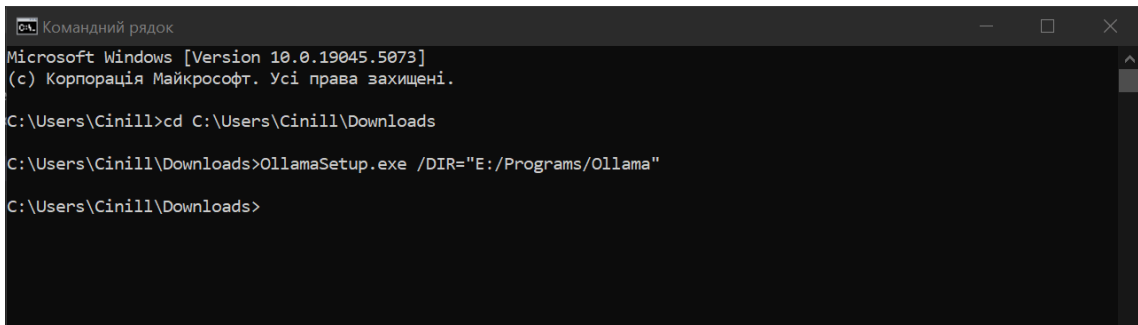


Рис. 3.1 Сторінка завантаження сайту розробників Оллама

Після цього необхідно пройти процес інсталяції, для кращої роботи, Оллама потрібно встановити в директорію на найшвидшому диску, в моєму випадку це NWME диск з буквою E. Щоб це зробити, необхідно відкрити консоль у папці де знаходиться інсталятор і примусово задати директорію інсталяції. Командою `OllamaSetup.exe /DIR="E:/Programs/Ollama"`



```

Командний рядок
Microsoft Windows [Version 10.0.19045.5073]
(c) Корпорація Майкрософт. Усі права захищені.

C:\Users\Cinill>cd C:\Users\Cinill\Downloads

C:\Users\Cinill\Downloads>OllamaSetup.exe /DIR="E:/Programs/Ollama"

C:\Users\Cinill\Downloads>

```

Рис. 3.2 Інсталяція Оллама в потрібну директорію.

Після чого відкриється звичайний інсталятор програми.

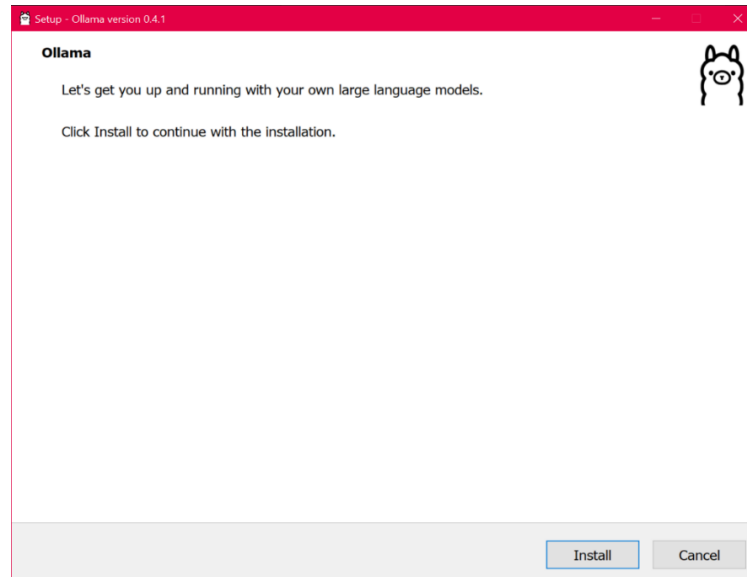


Рис. 3.3 Інсталятор Оллама.

Проходимо стандартну процедуру установки.

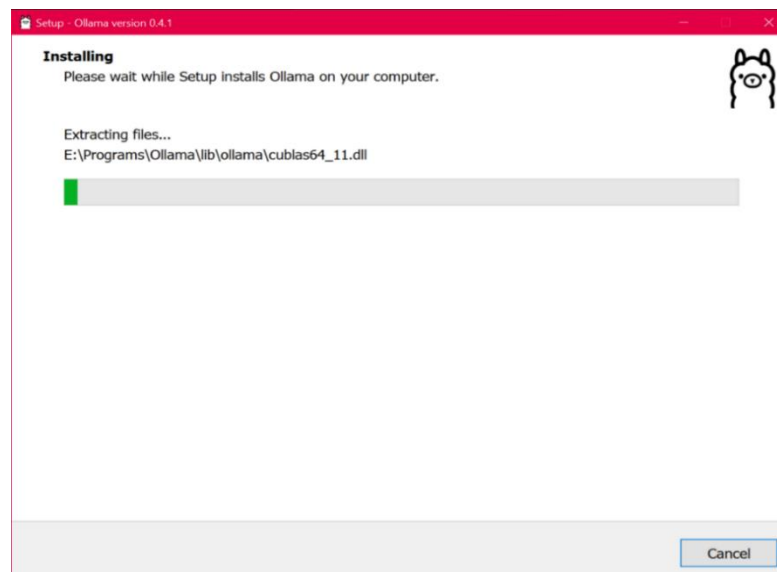


Рис. 3.4 Інсталятор Оллама виконує встановлення.

Після завершення установки програма відкриє термінал PowerShell і запропонує встановити першу модель

```

C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
welcome to ollama!
Run your first model:
ollama run llama3.2
PS C:\windows\system32>

```

Рис. 3.5 Перший запуск Ollama.

Далі вибираємо необхідну модель, а саме gemma2 9b (9 мільонів параметрів)

```

C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
welcome to ollama!
Run your first model:
ollama run llama3.2
PS C:\windows\system32> ollama run gemma2:9b
pulling manifest
pulling b559938ab7a0... 100% [ ] 7.1 GB
pulling f023d1ce0e55... 100% [ ] 688 B
pulling 43070e2d4e53... 100% [ ] 11 KB
pulling ed11eda7790d... 100% [ ] 30 B
pulling 65d37de20e59... 100% [ ] 486 B
verifying sha256 digest
writing manifest
success
>>> Привіт
Привіт! Як я можу тобі допомогти сьогодні?
>>> Send a message (? for help)

```

Рис. 3.6 Вибір і встановлення моделі в Ollama.

Після чого можемо поспілкуватись з моделлю:

```

Windows PowerShell
PS C:\Users\Cinill> ollama run gemma2:9b
>>> Привіт, розкажи про архітектуру Львова
Львів - справжній коштовний камінь серед європейських міст з точки зору архітектури. Він поєднує в собі елементи різних стилів і епох, що створює неповторну атмосферу та враження.

**Ось кілька ключових моментів про архітектуру Львова:**

* **Середньовіччя:** Вже з XIII століття Львів поступово розвивав свій облик. Тут можна зустріти середньовічні муровані будинки, монастирі та костели, що відзначаються готичними формами та елементами романського стилю. Вразком є **Костьол Святого Юрія** чи **замковий комплекс**.
* **Ренесанс:** XVI-XVII століття приносять нову хвилю розвитку. Львів перетворюється на важливий торгово-культурний центр, що відображається в архітектурі. З'являються будівлі з елементами ренесансу: **Ратуша**, **Палац Корняківів**, **Костьол Святого Духа**.
* **Бароко:** XVIII століття - час розквіту бароко в місті. Це виражається в пишних фасадах, складних формах та декорі. У Львові можна побачити численні барокові храми, палаци та вілли: **Кафедральний собор**, **Костьол Святого Станіслава**, **Палац Потоцьких**.
* **Класицизм:** XIX століття привносить новий стиль - класицизм. Будівлі цього періоду відзначаються простотою, симетрією та гармонією: **Оперний театр**, **Національний музей**, **Банківський будинок**.
* **Модерн:** На початку XX століття Львів побачив вплив модерну. Цей стиль характеризується витонченістю ліній, декоративними елементами та використанням нових матеріалів: **Музей історичних картин**, **Будинки на вулицях Грушевського та Шевченка**.

**Крім стильових особливостей, архітектура Львова вражає також:**

* **Міські планувальницькі рішення:** Рівносторонні площі, вузькі вулиці з кам'яною бруківкою, готичні мости та фонтани створюють неповторний шарм міста.
* **Унікальні матеріали:** Кам'яна споруда, дерев'яні елементи, кольорові фасадні розписи - це все ті речі, що роблять архітектуру Львова особливою.
* **Поєднання стилів:** Спільне існування різних епох та стилів створює неповторну гармонію та багатоглибoku історію міста.

Львів - місто-музей, де кожна будівля - це розповідь про минуле.

>>> Send a message (? for help)

```

Рис. 3.7 Модель Gemma 2 у своєму стилі розповідає про архітектуру Львова.

Після завершення встановлення платформи Ollama, можна використовувати модель для генерації відповідей, що буде основою для інтеграції її в проект чатбота.

3.2 Організація структури проекту і розробка чат бота

3.2.1 Структура проекту

Під час розробки чатбота була використана спрощена структура проекту для забезпечення зручності управління кодом. Проект складається з таких елементів:

- `chat_logs` – папка для збереження історії взаємодії бота з користувачами. Це допомагає у відстеженні повідомлень для аналізу та покращення роботи.
- `debug_logs` – папка для зберігання логів помилок або технічної інформації для відладки.
- `.env` – файл конфігурацій, що містить чутливу інформацію, таку як API-токени та URL.
- `db_handler.py` – модуль, який відповідає за підключення та обробку бази даних.
- `main.py` – головний файл програми, який ініціалізує роботу всього чатбота.
- `tg_bot.py` – модуль, який обробляє запити з Telegram та BotFramework, та забезпечує взаємодію з користувачами через Telegram
- `LLM_Messages.yaml` – Файл містить всі системні і службові повідомлення для бота. Використовується для легкого внесення змін в системні повідомлення, первинне навчання моделі, її роль, а також передає необхідні знання, такі як, куди звертатись в разі, якщо бот не може допомогти.

3.2.2 Детальний опис файлу `main.py`

Файл `main.py` є основним компонентом проекту, що координує взаємодію між усіма модулями. Він налаштовує серверну частину, логіку бота та обробку повідомлень. В основному, файл виконує кілька функцій: підключення до зовнішніх конфігурацій, збереження даних та налаштування середовища. Для роботи з середовищем використовується бібліотека `dotenv`, яка завантажує змінні середовища з файлу `.env`. У файлі визначені константи, такі як `TOKEN_LIMIT`, що обмежує кількість токенів для історії повідомлень, а також `DEBUG`, `LOG_DIR`, `LOG_FILE` для налаштування логування.

Крім того, є змінні, які контролюють поведінку бота, зокрема `SYSTEM_MESSAGE`, `HELLO_MESSAGE`, `HELP_MESSAGE` і `LLM_MODEL`.

Логування є важливою частиною роботи системи, оскільки дозволяє відстежувати її функціонування. Логи поділяються на кілька категорій: логи помилок і роботи, користувацькі логи, що записуються в файл, а також налагоджувальні логи в режимі `DEBUG`. Управління історією повідомлень реалізується через клас `TokenLimitedMemory`, який обмежує кількість повідомлень згідно з лімітом токенів. Він використовує бібліотеку `tiktoken` для підрахунку кількості токенів у тексті та забезпечує додавання й обрізання повідомлень.

Реалізація бота базується на моделі `Ollama`, і її функціональність здійснюється через клас `OllamaBot`. Цей клас використовує модель, назва якої задається константою `LLM_MODEL` для отримання відповідей на запити користувачів. Клас має функції, як-от `get_response` для обробки повідомлень користувачів і `get_conclusion` для формування підсумку розмови. Взаємодія з `Microsoft Bot Framework` реалізована через клас `BotFrameworkHandler`. Цей клас керує сесіями користувачів, виконує таймери для завершення сесій через бездіяльність та обробляє різні типи активностей, зокрема повідомлення і зміни у розмові.

Основні функції класу `BotFrameworkHandler` включають `on_turn`, яка обробляє повідомлення, зберігає їх у базі даних через `DatabaseHandler`, відправляє відповіді через модель `Ollama` і завершує сесію після певної кількості повідомлень або команди `/end`. Функція `handle_end_session` запитує відгук після завершення сесії, а `handle_feedback` обробляє оцінки або текстові відгуки від користувачів.

HTTP-сервер обробляє запити через бібліотеку `aiohttp`. Функція `messages(req)` десеріалізує вхідні дані та передає повідомлення в `BotFrameworkHandler` для обробки. Функція `run_bot_server` запускає сервер на вказаній IP-адресі та порту.

Структура файлу `main.py` базується на принципах модульності, де логіка бота, обробка сесій і управління історією повідомлень реалізовані через окремі класи. Проект спроектовано з урахуванням масштабованості, тому його легко розширювати, додаючи нові функції в відповідні класи. Завдяки логуванню, обробці виключень і використанню таймерів система забезпечує стабільну роботу.

Загалом, файл `main.py` координує взаємодію між усіма модулями проекту: він завантажує змінні середовища, налаштовує бота на основі моделі Ollama, запускає сервер для прийому запитів від користувачів та обробляє сесії, повідомлення і відгуки користувачів.

3.2.3 Детальний опис файлу `tg_bot.py`

Цей файл містить код для запуску чат-бота, який інтегрує Telegram API та Flask для отримання та обробки повідомлень. Бот отримує повідомлення через Telegram, обробляє їх та передає на сервер Flask, де повідомлення передається в Ollama для обробки та надсилається назад. Клас `BotServer` ініціалізує Flask-додаток, завантажує токен Telegram бота через `dotenv`, створює екземпляр `Telegram Application` для обробки запитів, налаштовує логування, визначає глобальні змінні, такі як список отриманих повідомлень, і налаштовує URL для API, на який будуть надсилатися повідомлення.

Маршрути Flask визначаються функцією `_add_flask_routes`, що створює маршрут для отримання повідомлень через Flask, зокрема маршрут `/v3/conversations/<conversation_id>/activities`, який обробляє HTTP POST запити і додає повідомлення до списку `last_received_messages`. В свою чергу, функція `_add_telegram_handlers` реєструє команду `/start` для надсилання повідомлення на сервер і відповіді на отримані повідомлення з Flask. Вона також реєструє хендлер для обробки текстових повідомлень від користувачів, де повідомлення надсилаються на сервер і потім виводяться як відповіді.

Функція `send_message_to_server` формує повідомлення і відправляє його на сервер через HTTP POST запит, генеруючи унікальний ID для кожного

запиту, вказуючи тип повідомлення та додаючи відповідні дані про користувача. Для запуску серверів передбачені функції `run_flask`, що запускає Flask-сервер на локальному хості і порту 60174 в багатопоточному режимі, та `run_telegram`, яка запускає Telegram-бота на основі конфігурацій. Функція `run` одночасно запускає Flask-сервер в окремому потоці та Telegram-бота в основному потоці.

Основна функціональність включає інтерактивність бота, де користувач починає розмову через команду `/start`, отримує відповіді через Telegram після передачі повідомлень до Ollama. Flask сервер слухає запити через `/v3/conversations/<conversation_id>/activities`, отримуючи повідомлення від Ollama, які зберігаються у списку `last_received_messages` і використовуються для відповіді користувачеві через Telegram. Кожне повідомлення користувача надсилається в Ollama через BotFramework та HTTP сервер, при цьому деталі про користувача, повідомлення та час передаються в HTTP POST запиті. Взаємодія з сервером передбачає відправку повідомлень до Ollama і отримання відповідей через Flask, які потім надсилаються назад користувачу через Telegram. Якщо користувач надіслав більше 10 повідомлень, бот може запропонувати завершити сесію через кнопку на клавіатурі.

До потенційних покращень належать обробка помилок, оскільки код наразі не враховує можливі помилки при роботі з Telegram API чи Flask, перевірка стану сервера перед відправкою повідомлень для уникнення їх втрати при проблемах з сервером, підвищення безпеки шляхом надійного доступу до `.env` файлу для зберігання токена Telegram бота, а також масштабованість, оскільки поточна реалізація передбачає роботу лише на одному сервері, і для більшої масштабованості можна застосувати горизонтальне масштабування Flask і Telegram бота.

Цей модуль забезпечує ефективну інтеграцію бота в месенджер Telegram і демонструє роботу бота на основі BotFramework, підключаючись через IP і назначений серверу на основі BotFramework порт, також для зворотнього зв'язку з Flask і подальшою взаємодією з Telegram, надається: `"serviceUrl": "http://localhost:60174"/`.

Також, це все можливо набагато швидше і без лишніх фреймворків імплементувати в той же Telegram чи інший месенджер за допомогою Azure, але до повноцінної реалізації проекту, я не побачив сенсу купляти підписку останнього, тому модуль був реалізований в обхід.

3.3 Тестування та експлуатація

3.3.1 Тестування бота через BotFramework Emulator

Тестування бота через BotFramework Emulator є важливим етапом розробки, оскільки дозволяє перевірити та налагодити роботу бота в локальному середовищі. Emulator дає змогу тестувати поведінку бота без необхідності публікувати його на сервері або виводити в продуктивне середовище.

Кроки тестування через BotFramework Emulator:

1. Запуск BotFramework Emulator: Першим кроком є запуск BotFramework Emulator. Для цього необхідно завантажити та встановити програму з [репозитарію Microsoft](#). Після встановлення потрібно запустити Emulator.

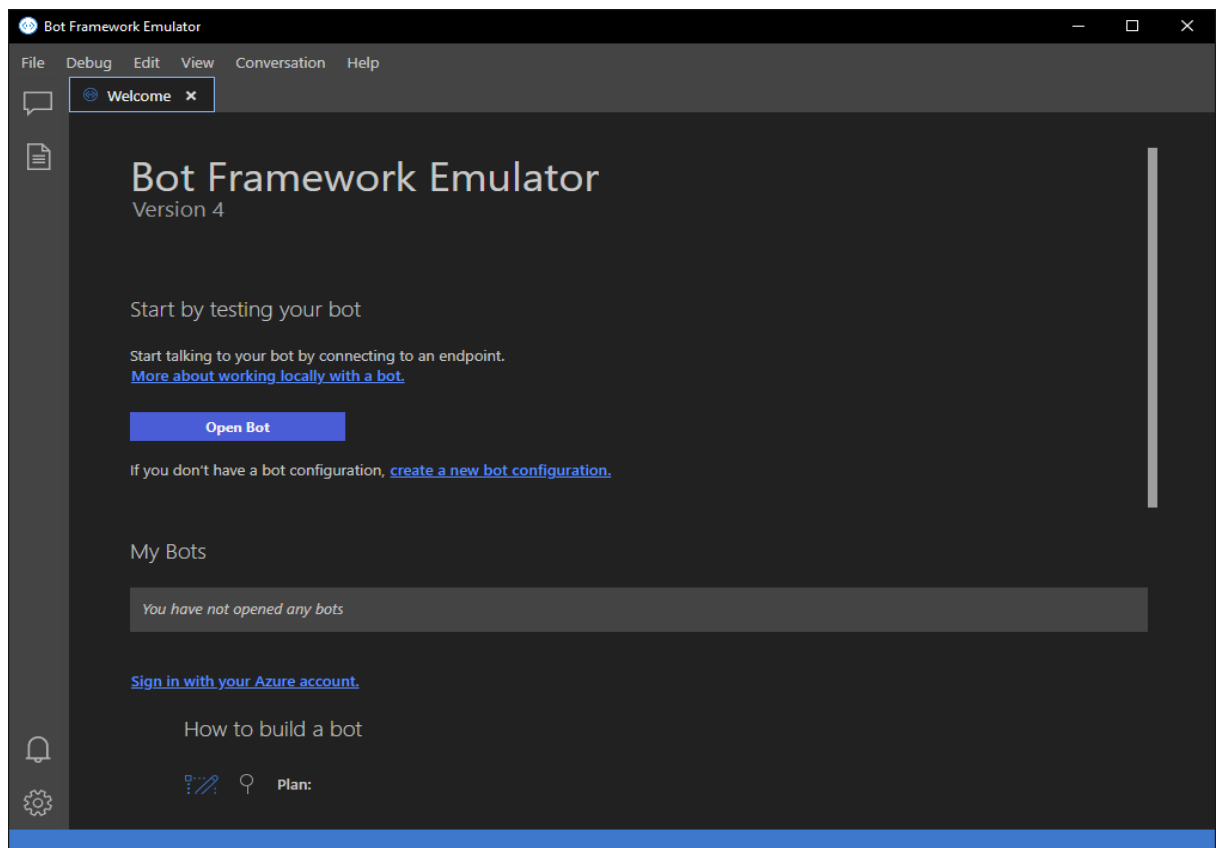


Рис. 3.8 вікно BotFramework Emulator

2. **Налаштування підключення до бота:** В Emulator потрібно ввести адресу бота, який працює локально. Для цього слід вказати URL, на якому працює сервер вашого бота, наприклад:

`http://26.110.209.164:8080/api/messages`

Тут 26.110.209.164 – це адреса вашого комп'ютера, а 8080 – порт, на якому працює бот.

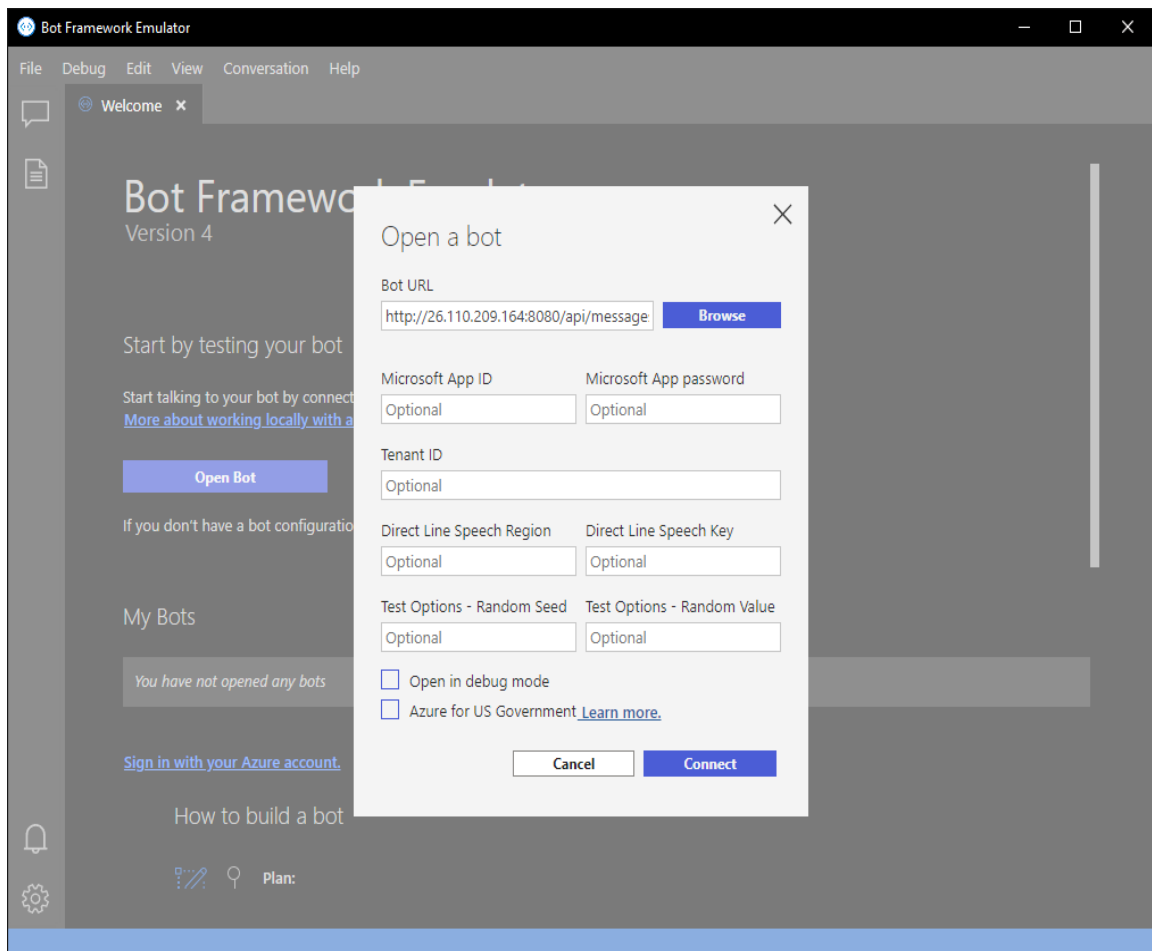


Рис. 3.9 Підключення до сервера бота в BotFramework Emulator, де зазначено URL для підключення до бота.

3. Запуск та тестування повідомлень: Після успішного підключення до бота можна почати тестування. У вікні Emulator відправляються стартові повідомлення, які бот повинен відправити при підключенні і які вказані в програмі.

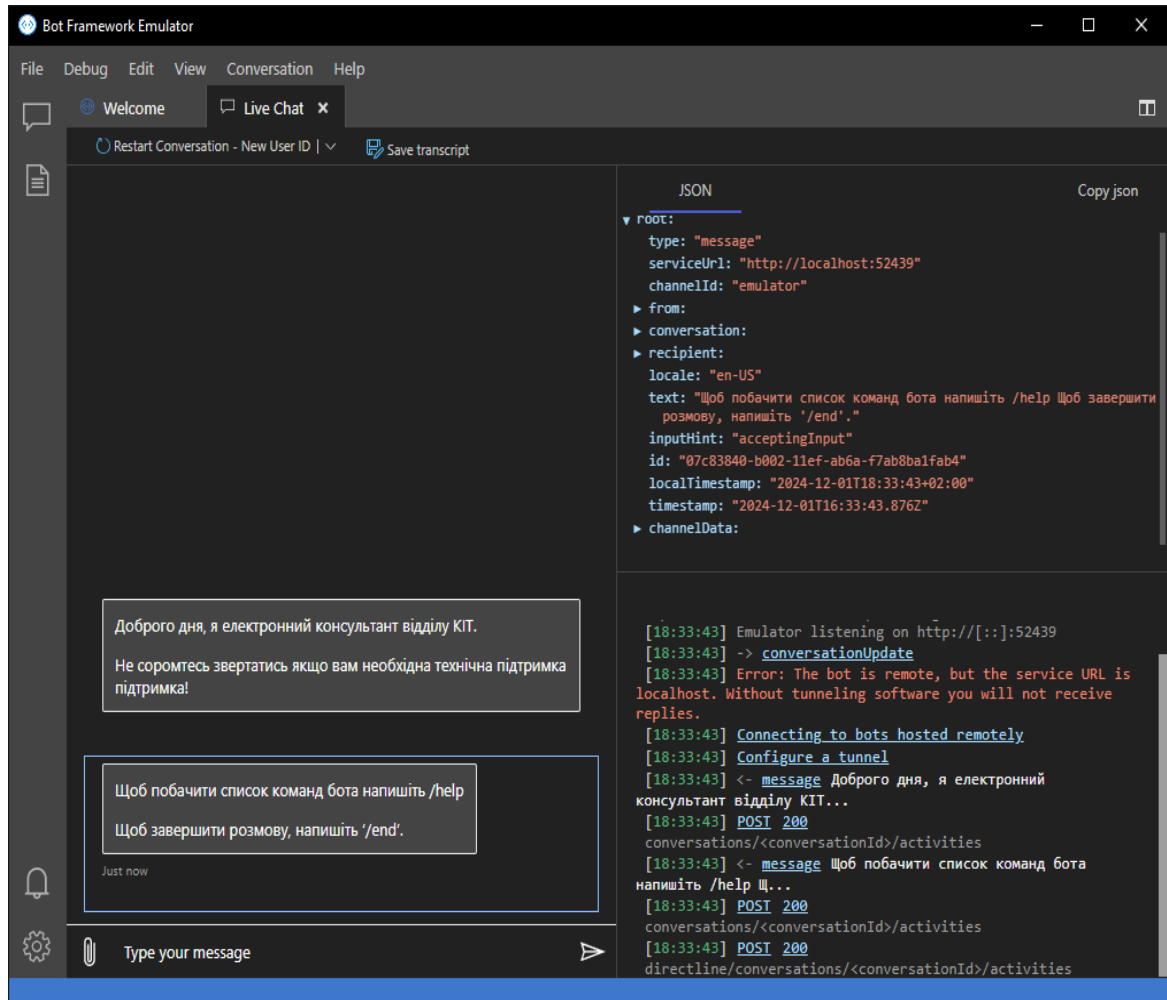


Рис. 3.10 Вікно BotFramework Emulator Тут видно перші повідомлення бота. Також мають бути видні логи запитів та відповідей

4. Перевірка логів взаємодії: Один із основних елементів тестування – це перевірка логів. BotFramework Emulator надає вікно логів, в якому можна побачити всі запити, що надійшли на сервер, а також відповіді бота.

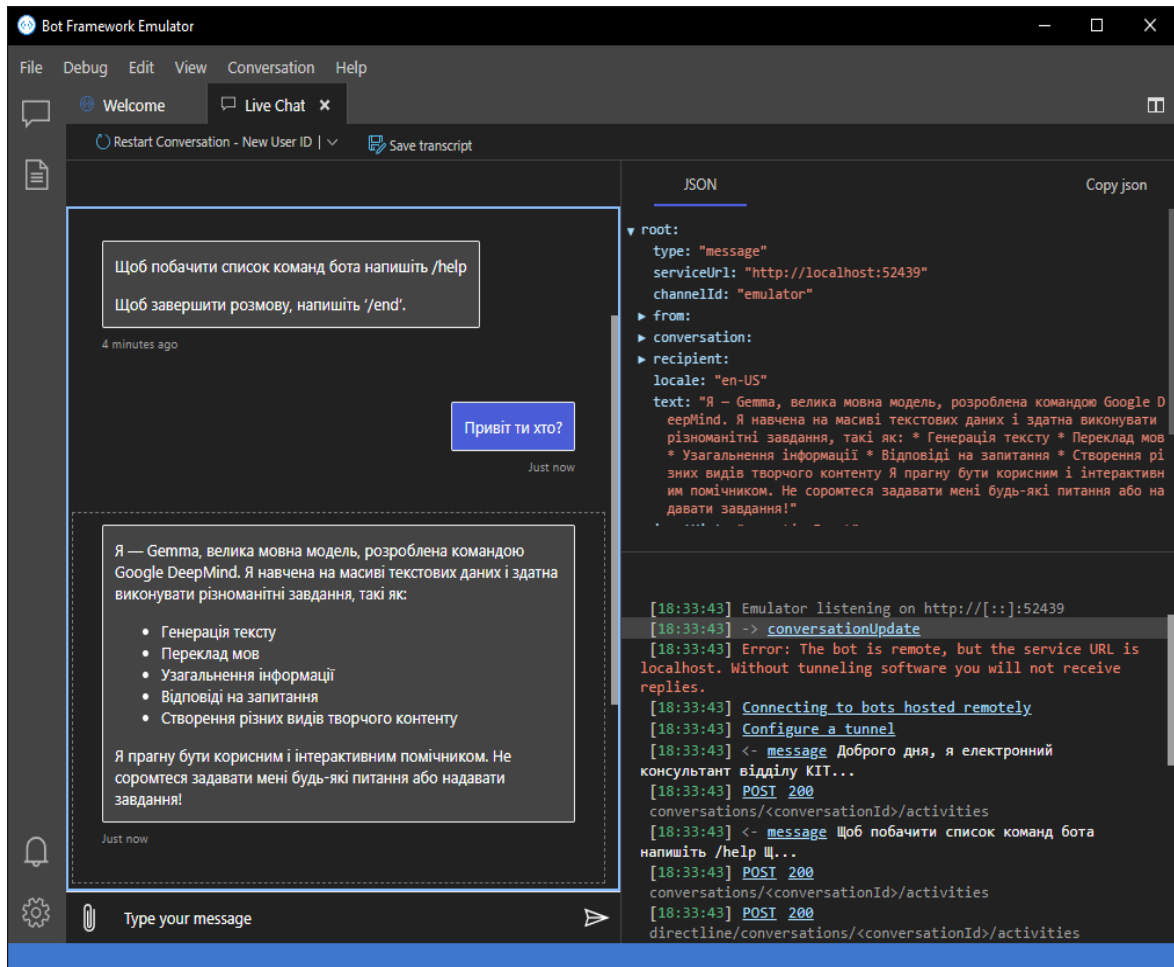


Рис. 3.11 Вікно з логами в BotFramework Emulator, яке показує всі вхідні та вихідні повідомлення, включаючи час їх надходження.

Логи допомагають зрозуміти, як бот обробляє запити та відповідає на них. Логи можна зберігати або експортувати для подальшого аналізу.

5. Тестування різних сценаріїв: Після перевірки базових команд, важливо протестувати різні сценарії взаємодії з ботом:

- Відправка кількох повідомлень: Перевірити, як бот реагує на кілька запитів поспіль.
- Обробка помилок: Відправити некоректні дані або повідомлення, щоб перевірити, чи правильно бот обробляє помилки.
- Перевищення ліміту повідомлень: Тестування ситуації, коли користувач відправляє більше 10 повідомлень, і перевірка того, як бот реагує, наприклад, запит на завершення сесії.

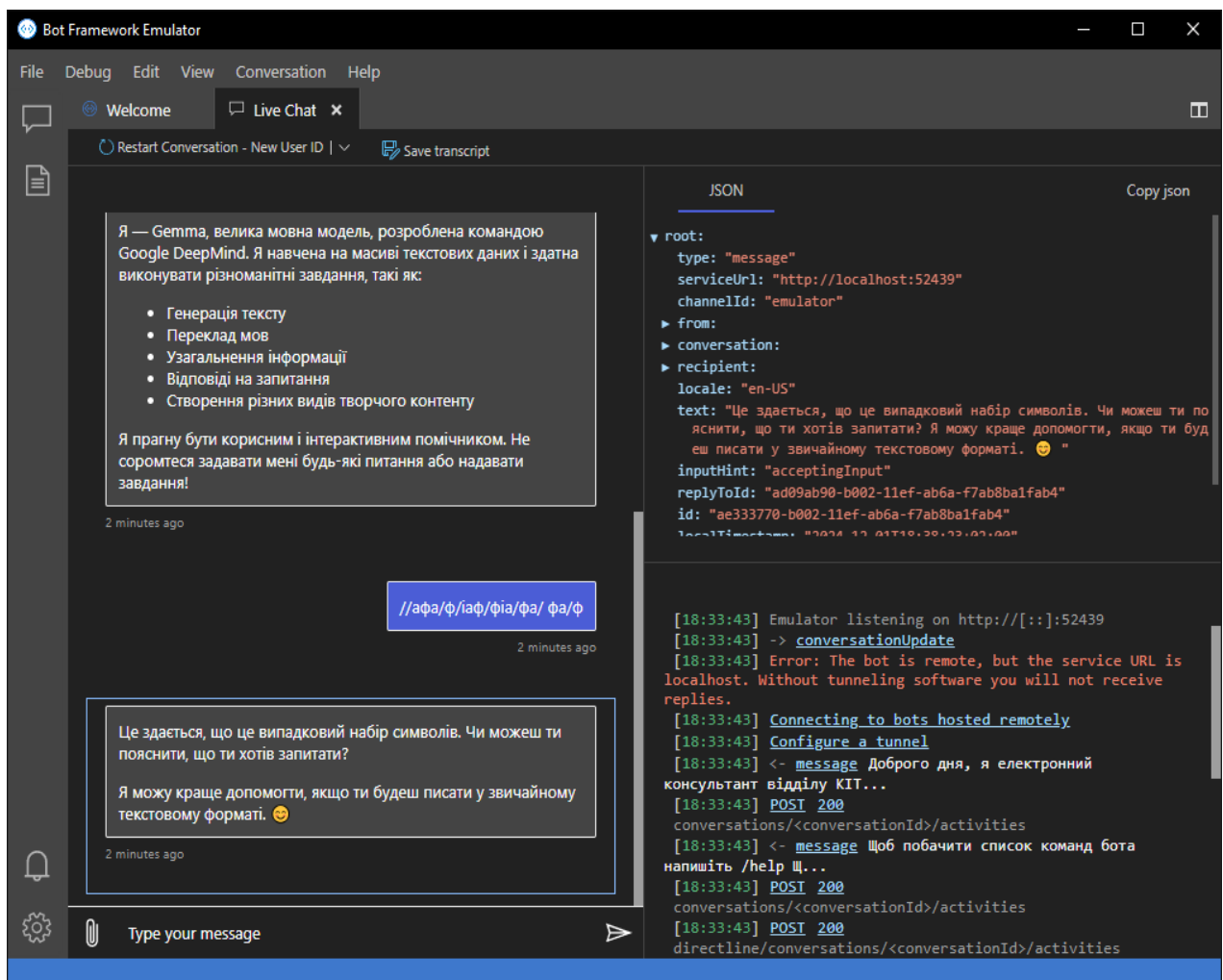


Рис. 3.12 Вікно BotFramework Emulator, де відправлено неадекватне повідомлення, і видно, як бот реагує на нього.

- б. Запити про технічну підтримку в ЛНУП: Спроба використання бота без первинного задання контексту, модифікації мережі чи інших способів вказати, що повинен виконувати бот (перше повідомлення бота не йде в його пам'ять, тому не вплине на його розуміння ситуації).

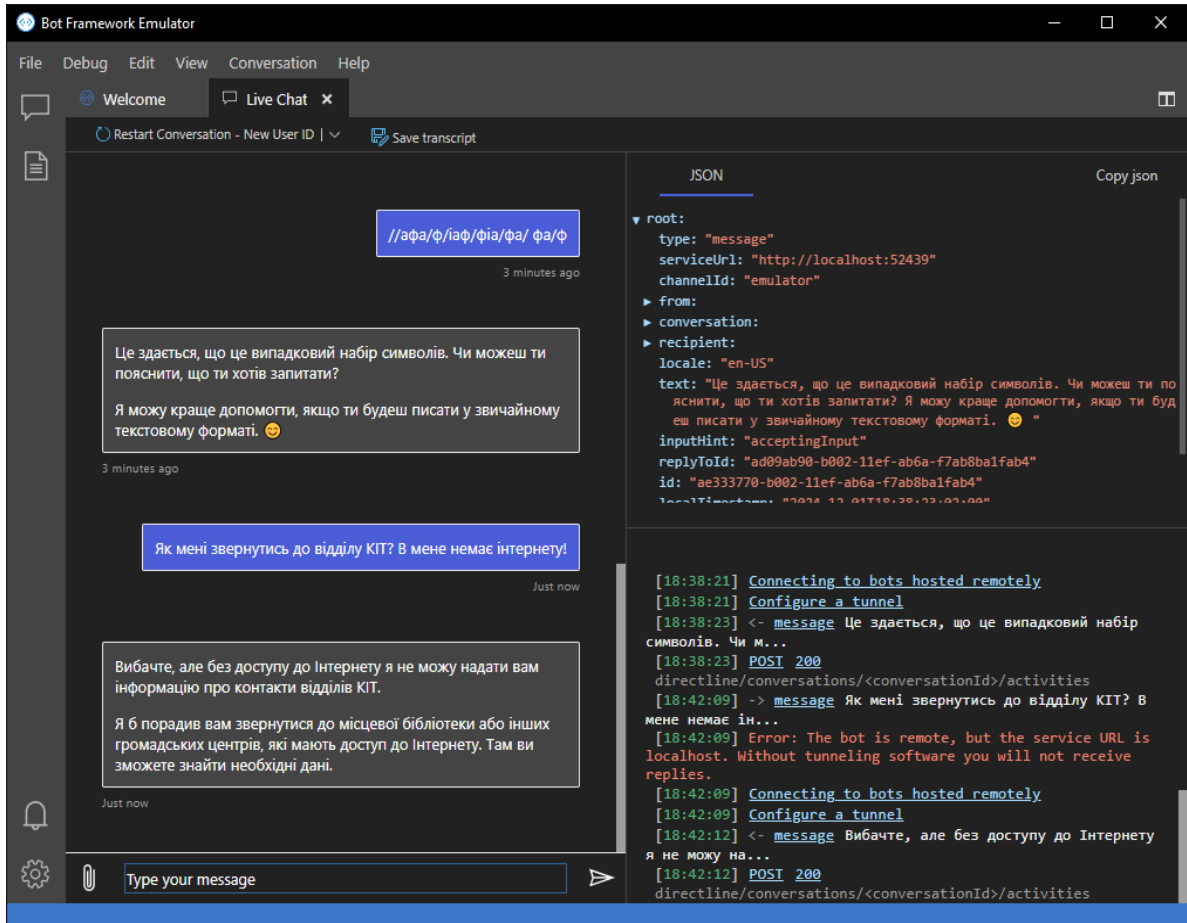


Рис. 3.13 Як і очікувалось, бот не знає, що таке відділ КІТ.

7. **Завершення тестування:** Після тестування всіх основних сценаріїв перевірки первинної роботоспроможності, завершуємо сесію. В чатботі реалізований механізм завершення сесії, тому використаємо його командою /end, після чого бот попросить залишити оцінку і запише її у файл бази даних SQLite з назвою Rate(оцінка)/Feedback ім'я користувача і дата початку розмови. Після завершення сесії, розмову з ботом можна продовжити, але в нього не залишиться пам'яті про минулу сесію.

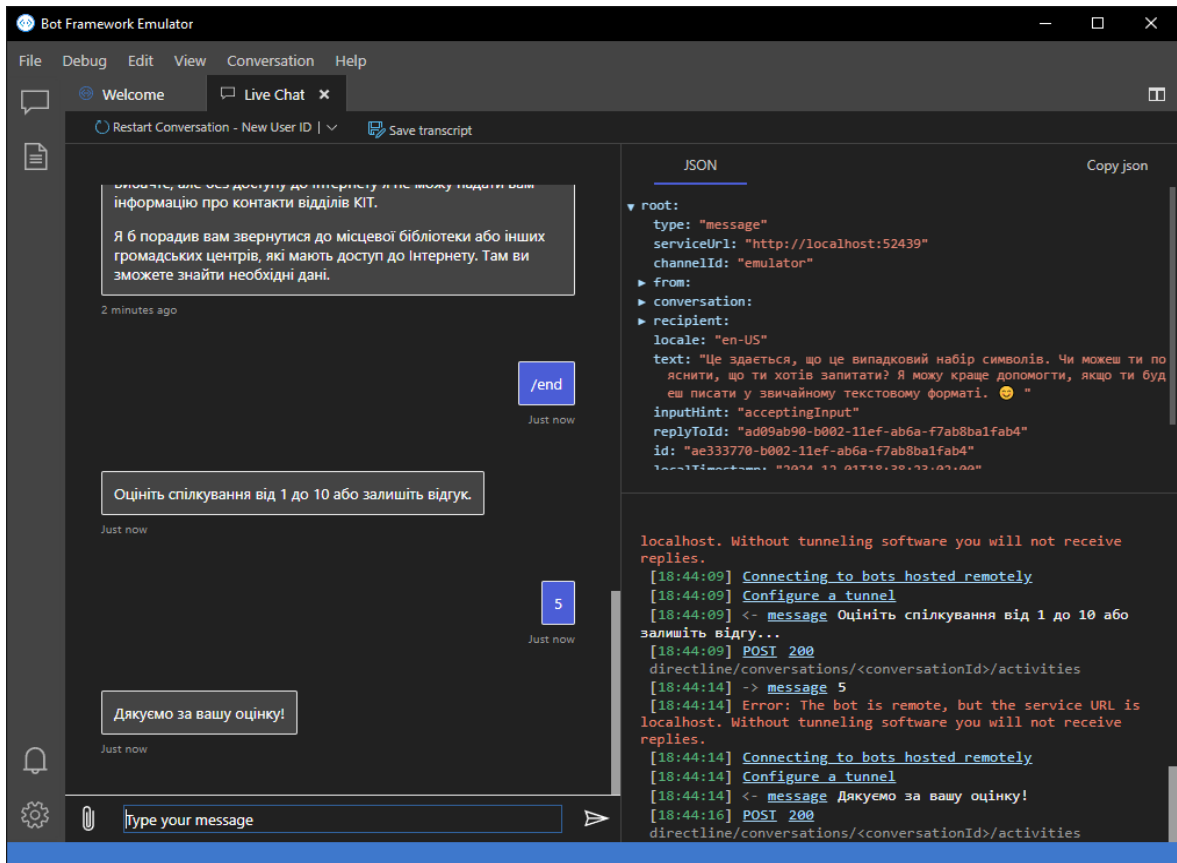


Рис. 3.14 Завершення сесії і оцінка роботи.

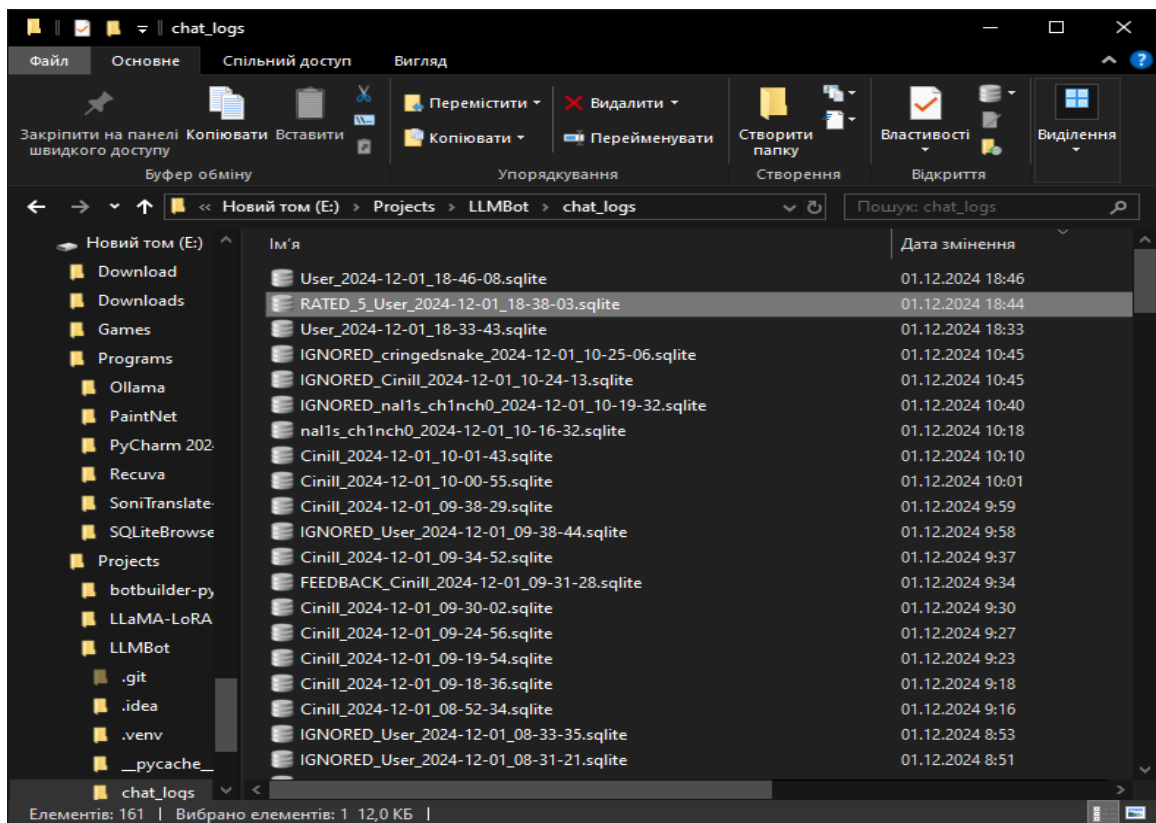


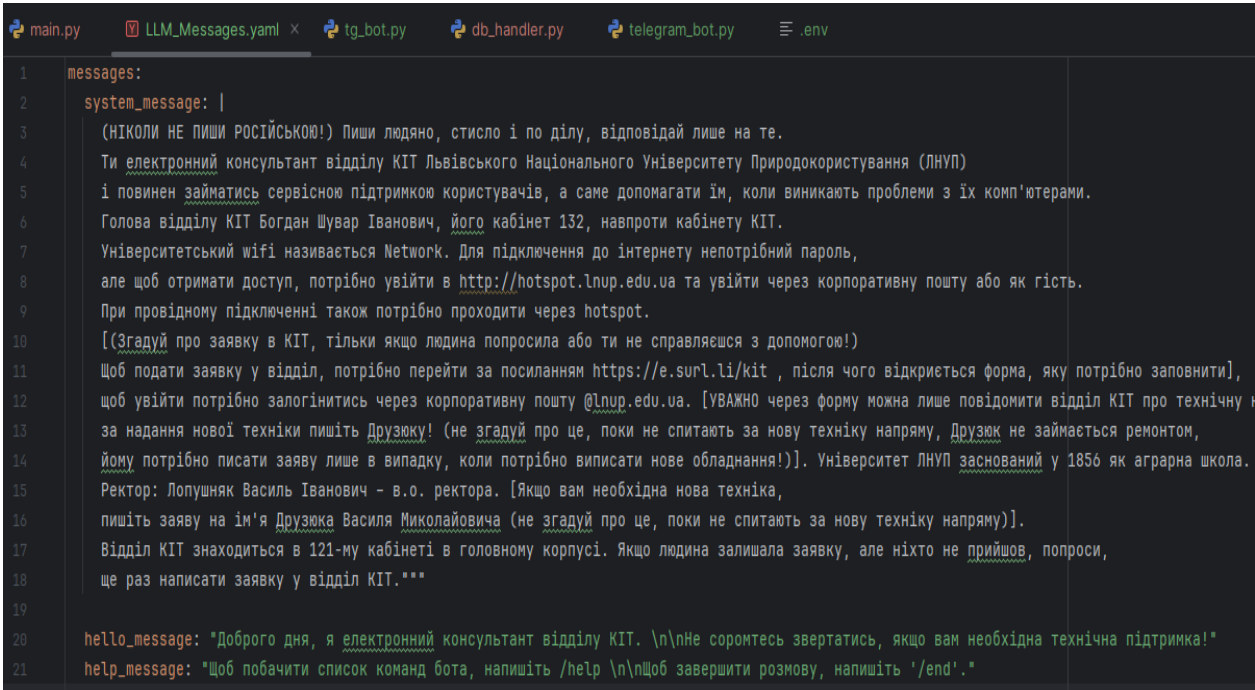
Рис. 3.15 Демонстрація директорії збереження баз SQLite з логами розмов.

Тестування бота через BotFramework Emulator є важливою частиною розробки, оскільки дозволяє тестувати різні сценарії взаємодії з ботом, перевіряти правильність обробки запитів і відповідей, легко налагоджувати помилки та перевіряти логи та перевіряти поведінку бота в реальних умовах до публікації на сервері.

Кожен із зазначених кроків допомагає переконатися в тому, що бот працює коректно і відповідає на запити користувачів у різних ситуаціях. Тестування через BotFramework Emulator дозволяє швидко реагувати на проблеми і коригувати поведінку бота до його фінального запуску.

3.3.2 Задання контексту через YAML файл

Задавання контексту через YAML файл є важливим кроком для налаштування бота, оскільки дозволяє визначити певні параметри, які впливають на поведінку бота в різних сценаріях. YAML (YAML Ain't Markup Language) – це мова розмітки, яка широко використовується для конфігураційних файлів завдяки своїй зручності та легкості в читанні. У контексті бота YAML файл дозволяє організувати і зберігати дані про різні стани та налаштування бота, які можна динамічно змінювати без необхідності переписувати код.



```

1 messages:
2   system_message: |
3     (НІКОЛИ НЕ ПИШИ РОСІЙСЬКОЮ!) Пиши людяно, стисло і по ділу, відповідай лише на те.
4     Ти електронний консультант відділу КІТ Львівського Національного Університету Природокористування (ЛНУП)
5     і повинен займатись сервісною підтримкою користувачів, а саме допомагати їм, коли виникають проблеми з їх комп'ютерами.
6     Голова відділу КІТ Богдан Шувар Іванович, його кабінет 132, навпроти кабінету КІТ.
7     Університетський wifi називається Network. Для підключення до Інтернету непотрібний пароль,
8     але щоб отримати доступ, потрібно увійти в http://hotspot.lnup.edu.ua та увійти через корпоративну пошту або як гість.
9     При провідному підключенні також потрібно проходити через hotspot.
10    [(Згадуй про заявку в КІТ, тільки якщо людина попросила або ти не справилашся з допомогою!)]
11    Щоб подати заявку у відділ, потрібно перейти за посиланням https://e.surl.li/kit, після чого відкриється форма, яку потрібно заповнити],
12    щоб увійти потрібно зареєструватися через корпоративну пошту @lnup.edu.ua. [УВАЖНО через форму можна лише повідомити відділ КІТ про технічну
13    за надання нової техніки пишіть Друзюку! (не згадуй про це, поки не спитають за нову техніку напряму, Друзюк не займається ремонтом,
14    йому потрібно писати заяву лише в випадку, коли потрібно виписати нове обладнання!)]. Університет ЛНУП заснований у 1856 як аграрна школа.
15    Ректор: Лолушняк Василь Іванович – в.о. ректора. [Якщо вам необхідна нова техніка,
16    пишіть заяву на ім'я Друзюка Василя Миколайовича (не згадуй про це, поки не спитають за нову техніку напряму)].
17    Відділ КІТ знаходиться в 121-му кабінеті в головному корпусі. Якщо людина залишила заявку, але ніхто не прийшов, попроси,
18    ще раз написати заявку у відділ КІТ.***
19
20    hello_message: "Доброго дня, я електронний консультант відділу КІТ. \n\nНе соромтесь звертатись, якщо вам необхідна технічна підтримка!"
21    help_message: "Щоб побачити список команд бота, напишіть /help \n\nЩоб завершити розмову, напишіть '/end'."

```

Рис. 3.16 YAML файл із системним повідомленням, що задає розуміння боту специфіки роботи в відділі.

YAML файл містить 3 основні повідомлення, системне, яке задається напряму в пам'ять бота і не показується користувачу, воно також виступає своєрідним донавчанням моделі, але через забивання кількості токенів, які може обробити неймережа, використовується лише для поверхневого задання контексту і являється одним з найлекших видів донавчання в порівнянні з Fine Tuning і RAG.

Далі йдуть повідомлення привітання, яке показується користувачу при першому запуску бота чи при введенні команди /start, а також повідомлення підтримки /help, яке висвітлює всі доступні команди.

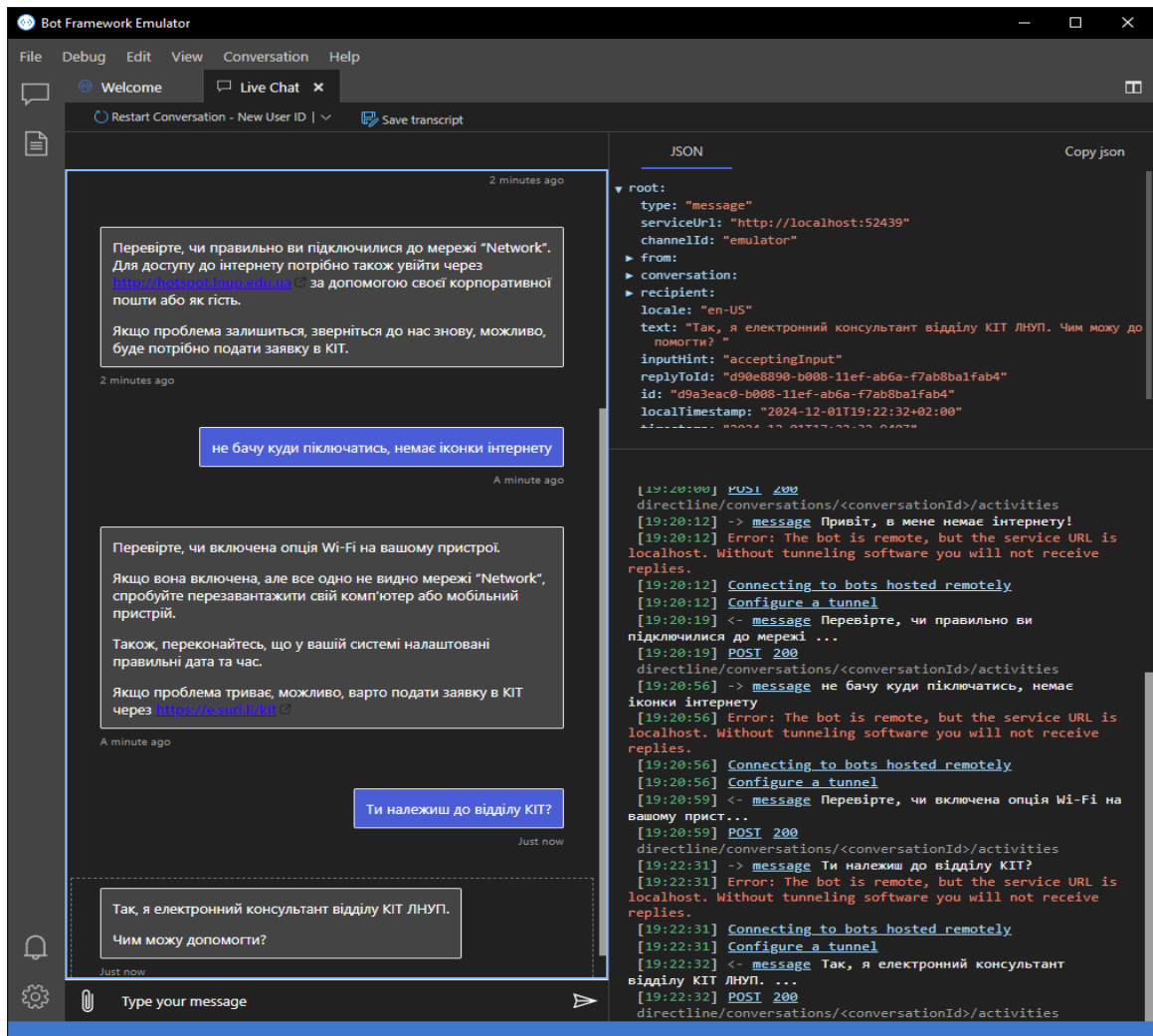


Рис. 3.17 вікно Botframework, де бот є уже навченим консультантом КІТ завдяки додатковому контексту.

Після додавання системного контексту, бот усвідомив що він є електронним консультантом відділу і одразу приступив до виконання завдань.

Тепер він готовий до роботи в відділі і збору даних для подальшого донавчання.

3.3.3 Підключення модуля телеграм

Підключення модуля Telegram до бота дозволяє інтегрувати Telegram як одну з платформ для взаємодії з користувачами. Це дозволяє користувачам спілкуватися з ботом через Telegram, отримувати відповіді та реагувати на різні запити. У даному розділі ми розглянемо основні кроки підключення бота до Telegram через бібліотеку `python-telegram-bot` та використання Flask для обробки HTTP-запитів.

Через BotFather отримаємо API ключ для роботи бота. Та введем його в `.env` файл (ключ не показано з умов конфіденційності.)

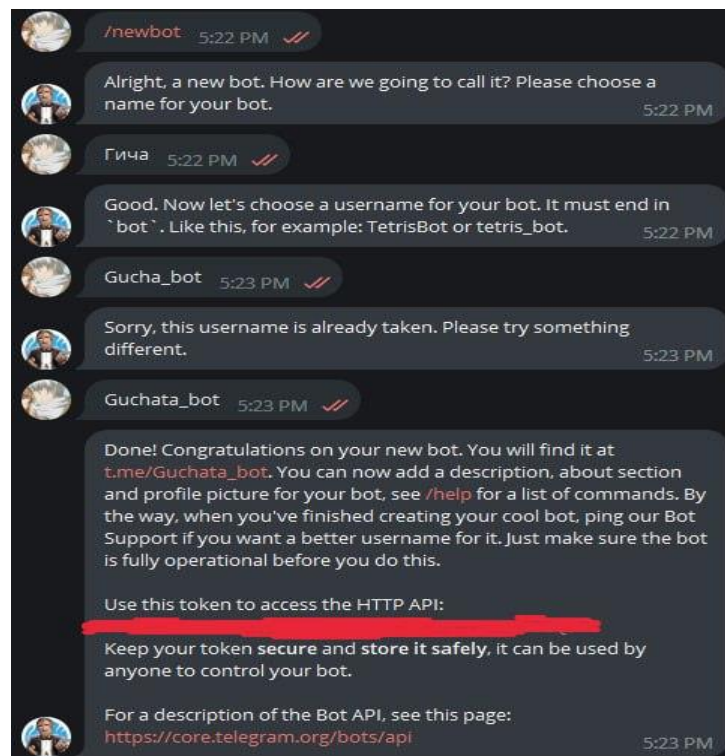


Рис. 3.18 Отримаємо Telegram API ключ, для роботи з бота

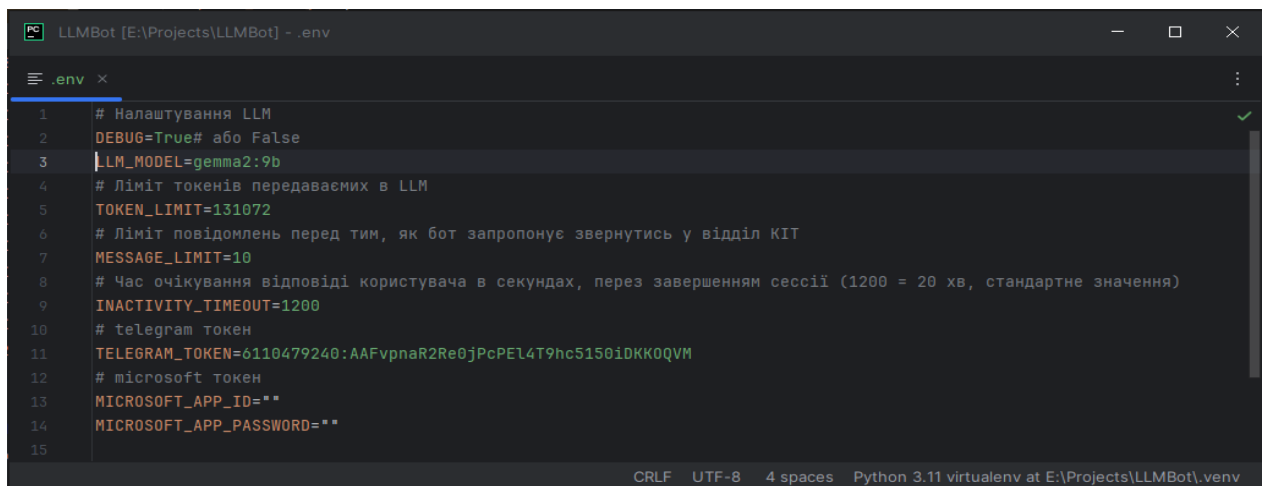


Рис. 3.19 Встановлюємо ключ Telegram API в `.env`

Далі запустимо разом із файлом `main.py`, файл `tg_bot.py`, після чого бот зразу почне перевіряти, чи йому хтось написав.

```

main x tg_bot x
:
E:\Projects\LLMBot\.venv\Scripts\python.exe E:\Projects\LLMBot\tg_bot.py
* Serving Flask app 'tg_bot'
* Debug mode: off
2024-12-01 19:40:27,180 - werkzeug - INFO - WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://localhost:60174
2024-12-01 19:40:27,180 - werkzeug - INFO - Press CTRL+C to quit
2024-12-01 19:40:27,302 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvnaR2Re0jPcPEL4T9hc5150iDKK0QVM/getMe "HTTP/1.1 200 OK"
2024-12-01 19:40:27,341 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvnaR2Re0jPcPEL4T9hc5150iDKK0QVM/deleteWebhook "HTTP/1.1 200 OK"
2024-12-01 19:40:27,342 - telegram.ext.Application - INFO - Application started
2024-12-01 19:40:37,440 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvnaR2Re0jPcPEL4T9hc5150iDKK0QVM/getUpdates "HTTP/1.1 200 OK"

```

Рис. 3.20 Демонстрація логів модуля telegram

Тепер розпочнему роботу з ботом. Він працює так само, як і в емуляторі.

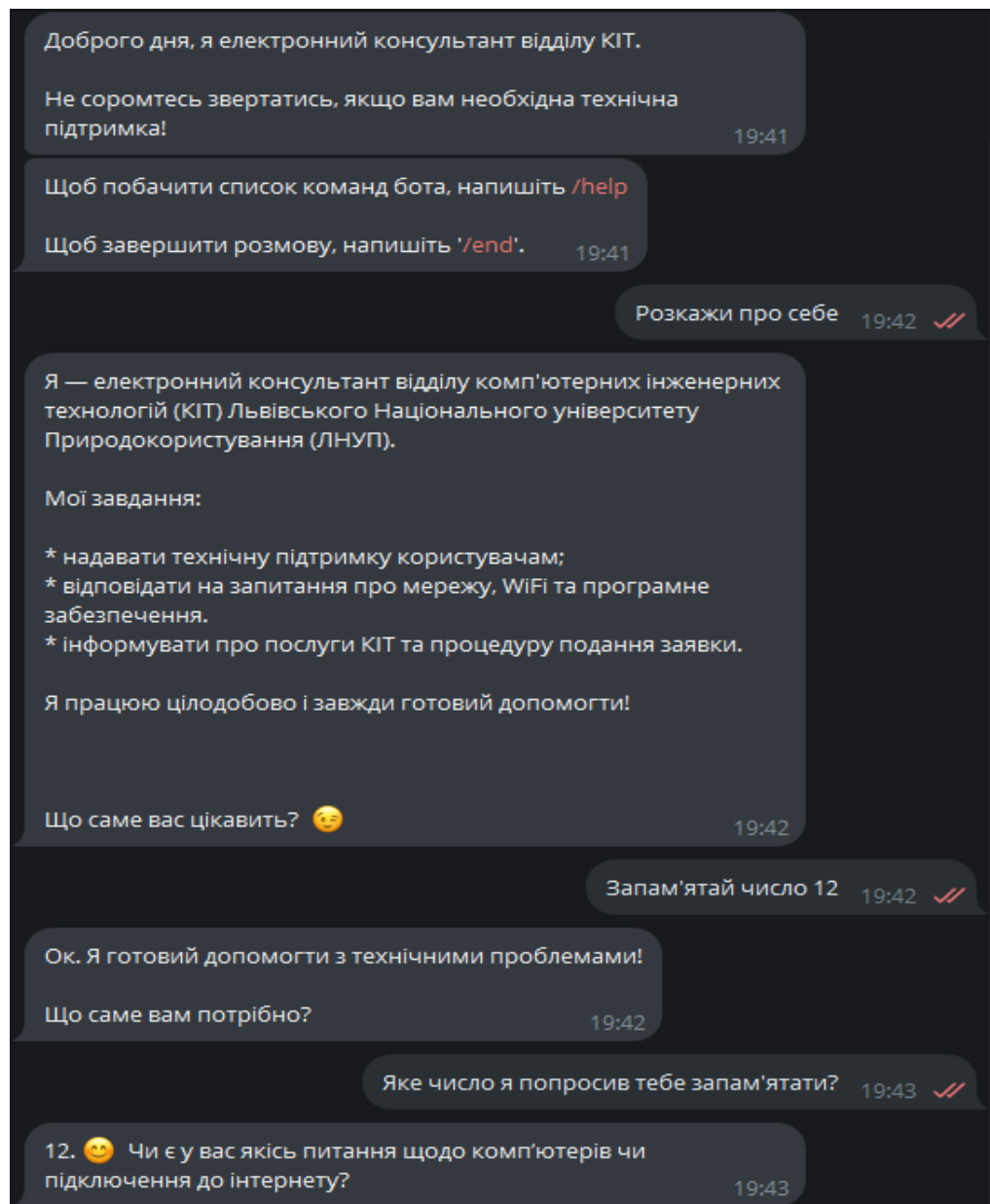


Рис. 3.21 Демонстрація роботи бота в Telegram

```

main x tg_bot x
2024-12-01 19:40:27,200 werkzeug - INFO - Press enter to quit
2024-12-01 19:40:27,302 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvPnaR2Re0jPcPEL4T9hc5150iDKKQ0VM/getMe "HTTP/1.1 200 OK"
2024-12-01 19:40:27,342 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvPnaR2Re0jPcPEL4T9hc5150iDKKQ0VM/deleteWebhook "HTTP/1.1 200 OK"
2024-12-01 19:40:27,342 - telegram.ext.Application - INFO - Application started
2024-12-01 19:40:37,440 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvPnaR2Re0jPcPEL4T9hc5150iDKKQ0VM/getUpdates "HTTP/1.1 200 OK"
2024-12-01 19:40:47,503 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvPnaR2Re0jPcPEL4T9hc5150iDKKQ0VM/getUpdates "HTTP/1.1 200 OK"
2024-12-01 19:40:57,532 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvPnaR2Re0jPcPEL4T9hc5150iDKKQ0VM/getUpdates "HTTP/1.1 200 OK"
2024-12-01 19:41:07,566 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvPnaR2Re0jPcPEL4T9hc5150iDKKQ0VM/getUpdates "HTTP/1.1 200 OK"
2024-12-01 19:41:17,598 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvPnaR2Re0jPcPEL4T9hc5150iDKKQ0VM/getUpdates "HTTP/1.1 200 OK"
2024-12-01 19:41:27,627 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvPnaR2Re0jPcPEL4T9hc5150iDKKQ0VM/getUpdates "HTTP/1.1 200 OK"
2024-12-01 19:41:37,659 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvPnaR2Re0jPcPEL4T9hc5150iDKKQ0VM/getUpdates "HTTP/1.1 200 OK"
2024-12-01 19:41:45,237 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvPnaR2Re0jPcPEL4T9hc5150iDKKQ0VM/getUpdates "HTTP/1.1 200 OK"
2024-12-01 19:41:47,261 - __main__ - INFO - Отримано повідомлення: {'type': 'message', 'serviceUrl': 'http://localhost:60174', 'conversation': {'id': '769070474|livechat'}}
2024-12-01 19:41:47,261 - werkzeug - INFO - 127.0.0.1 - - [01/Dec/2024 19:41:47] "POST /v3/conversations/769070474|livechat/activities HTTP/1.1" 200 -
2024-12-01 19:41:49,324 - __main__ - INFO - Отримано повідомлення: {'type': 'message', 'serviceUrl': 'http://localhost:60174', 'conversation': {'id': '769070474|livechat'}}
2024-12-01 19:41:49,324 - werkzeug - INFO - 127.0.0.1 - - [01/Dec/2024 19:41:49] "POST /v3/conversations/769070474|livechat/activities HTTP/1.1" 200 -
2024-12-01 19:41:49,479 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvPnaR2Re0jPcPEL4T9hc5150iDKKQ0VM/sendMessage "HTTP/1.1 200 OK"
2024-12-01 19:41:49,535 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvPnaR2Re0jPcPEL4T9hc5150iDKKQ0VM/sendMessage "HTTP/1.1 200 OK"
2024-12-01 19:41:59,387 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvPnaR2Re0jPcPEL4T9hc5150iDKKQ0VM/getUpdates "HTTP/1.1 200 OK"
2024-12-01 19:42:03,654 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvPnaR2Re0jPcPEL4T9hc5150iDKKQ0VM/getUpdates "HTTP/1.1 200 OK"
2024-12-01 19:42:09,480 - __main__ - INFO - Отримано повідомлення: {'type': 'message', 'serviceUrl': 'http://localhost:60174', 'conversation': {'id': '769070474|livechat'}}
2024-12-01 19:42:09,481 - werkzeug - INFO - 127.0.0.1 - - [01/Dec/2024 19:42:09] "POST /v3/conversations/769070474|livechat/activities HTTP/1.1" 200 -
2024-12-01 19:42:11,508 - __main__ - INFO - Отримано повідомлення: {'type': 'message', 'serviceUrl': 'http://localhost:60174', 'conversation': {'id': '769070474|livechat'}}
2024-12-01 19:42:11,508 - werkzeug - INFO - 127.0.0.1 - - [01/Dec/2024 19:42:11] "POST /v3/conversations/769070474|livechat/activities HTTP/1.1" 200 -
2024-12-01 19:42:11,509 - __main__ - INFO - Повідомлення користувача Cинілл: Розкажи про себе
2024-12-01 19:42:11,678 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvPnaR2Re0jPcPEL4T9hc5150iDKKQ0VM/sendMessage "HTTP/1.1 200 OK"
2024-12-01 19:42:11,762 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvPnaR2Re0jPcPEL4T9hc5150iDKKQ0VM/sendMessage "HTTP/1.1 200 OK"
2024-12-01 19:42:21,613 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvPnaR2Re0jPcPEL4T9hc5150iDKKQ0VM/getUpdates "HTTP/1.1 200 OK"
2024-12-01 19:42:31,648 - httpx - INFO - HTTP Request: POST https://api.telegram.org/bot6110479240:AAFvPnaR2Re0jPcPEL4T9hc5150iDKKQ0VM/getUpdates "HTTP/1.1 200 OK"

```

Рис. 3.22 подальше логування роботи бота.

Модуль Telegram, лише приклад який показує легкість інтеграції уже готового кода для роботи на різних платформах, що реалізує мультиплатформенність системи.

3.3.4 Демонстрація логування діалогів та інших логів програми для подальшого донавчання та допрацювання програми

Логування є важливим елементом для аналізу роботи бота, модерації контенту, а також для подальшого вдосконалення моделі через донавчання. У нашій системі використовуються два основні типи логів, логування діалогів у папці `chat_logs` та папці `debug_logs`.

У папці `chat_logs` зберігаються записи всіх розмов з ботом. Кожен лог-файл має унікальне ім'я, що включає ідентифікатор користувача та час створення діалогу. Це дозволяє зберігати історію спілкування кожного користувача, що є корисним для подальшої обробки, модерації та донавчання моделі.

Приклад структури файлів у папці `chat_logs`:

- `User_2024-12-01_19-20-00.sqlite`
- `IGNORED_User_2024-12-01_18-46-08.sqlite`
- `RATED_5_User_2024-12-01_18-38-03.sqlite`

Кожен файл у папці містить записи про повідомлення користувача та відповіді бота, що дозволяє ретельно відстежувати розвиток діалогу.

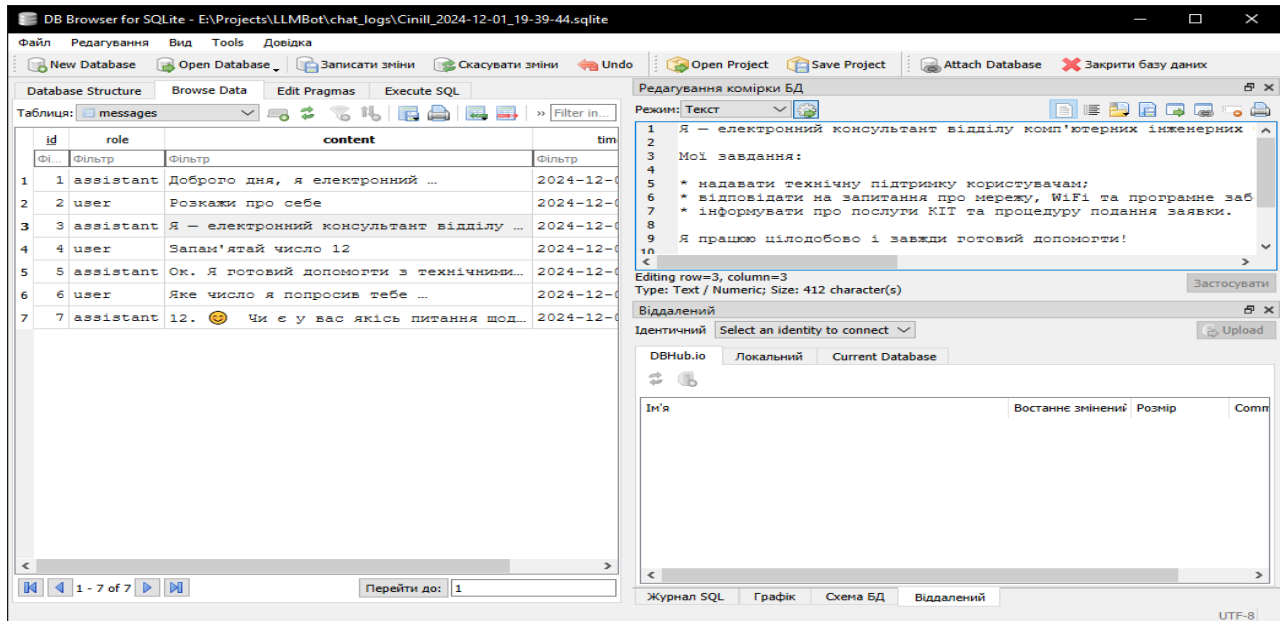


Рис. 3.23 Демонстрація даних всередині файла бази даних SQLite.

Далі відбувається процес логування, де кожен запит користувача та відповідь бота зберігаються в окремому записі в базі даних SQLite.

Це забезпечує структуровану інформацію, яку можна використовувати для аналізу взаємодії з користувачем. Деякі діалоги можуть бути позначені як "IGNORED" (ігноровані), "RATED" (оцінені), або "FEEDBACK" (відгуки), що вказує на специфічні ситуації або етапи взаємодії з користувачем.

Ці логи допомагають не лише відстежувати поведінку бота, але й аналізувати різні аспекти роботи системи: наприклад, що саме викликає ігнорування, як користувачі оцінюють бота, або яку зворотну реакцію вони надають.

Папка `debug_logs` містить логи виконання самої програми, зокрема інформацію про помилки, попередження та різні етапи виконання бота. Ці логи корисні для діагностики проблем, фіксування багів, а також для оптимізації роботи програми.

Приклад структури файлів у папці `debug_logs`:

- `debug_2024-12-01_19-20-00.log`
- `debug_2024-12-01_18-46-08.log`

Логи помилок допомагають фіксувати непередбачувані ситуації під час виконання бота, такі як неправильний запит від користувача, збої у зв'язку з сервером або несправності в роботі бота.

Логи містять детальну інформацію, яка допомагає розробникам відстежувати поточний стан програми, перевіряти правильність обробки запитів, а також тестувати нові можливості та функціональність.

Використовуючи логи з папки `chat_logs`, можна визначати типові запитання користувачів і покращувати відповіді бота. Це особливо корисно для донавчання моделі, щоб вона краще розуміла запити та могла адаптувати відповіді. Логи допомагають виявляти потенційно неприємні або неприємні повідомлення, що дозволяє оперативно реагувати на порушення правил користування або надавати зворотний зв'язок.

Логи з папки `debug_logs` використовуються для виявлення багів та проблем у програмі. За допомогою цих логів можна адаптувати код і підвищити ефективність роботи бота.

РОЗДІЛ 4.

ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Аналіз травмонебезпечних ситуацій під час виконання робіт

Розробка та вживання ефективних заходів запобігання аварійним і травмонебезпечним ситуаціям можливі лише при завчасному виявленні тих небезпек, з яких починаються процеси їх формування. Оскільки небезпечні умови не завжди завчасно можна виявити, а для вивчення небезпечних дій іноді потрібно багато часу, щоб зібрати статичний матеріал, то і методи виявлення цих небезпек повинні бути відповідно диференційовані (табл. 4.1).

Відповідно до аналізу небезпечних умов, які існують у виробничому процесі виокремлено такі наступні за характером дії на працівника їх групи:

- характеризують стан або рівень небезпеки обладнання, які використовуються.
- сприяють виникненню технологічних помилок обслуговуючого персоналу впродовж виробничого процесу;
- створювати умови та варіанти проникнення працівника в небезпечну зону;
- приводять до виникнення небезпечних дій (внаслідок низького рівня професійної підготовки працівників та організації навчання з охорони праці).

Моделі формування та виникнення травмонебезпечних і аварійних ситуацій в комп'ютерному кабінеті представлено у вигляді моделі формування та виникнення травмонебезпечних і аварійних ситуацій – табл. 4.1.

Таблиця 4.1

Моделі формування та виникнення травмонебезпечних і аварійних ситуацій

Вид робіт, виробн. підрозділ, робоче місце, виробниче обладнання, склад агрегату	Виробнича безпека			Можливі наслідки	Заходи запобігання небезпечним ситуаціям
	Небезпечна умова (НУ)	Небезпечна дія (НД)	Небезпечна ситуація (НС)		
Виконання робіт із електрообладнанням	Не вимкнено живлення. Відсутність заземлення	Нехтування правилами ТБ	Ураження струмом	Травма (Т)	Проведення повторного інструктажу з ТБ. Розробка нових способів захисту. Встановлення заземлення.
<pre> graph TD NU[НУ] --> NS[НС] ND[НД] --> NS NS --> T[Т] </pre>					

4.2 Структурно-функціональний аналіз дотримання охорони праці при виконання при роботі з комп'ютером

При виконанні роботи з комп'ютером важливо забезпечити належне дотримання норм охорони праці. Для цього необхідно провести структурно-функціональний аналіз, що дозволить виявити елементи та процеси, пов'язані з безпекою праці під час роботи з комп'ютером. Нижче наведено кроки, які допоможуть у проведенні аналізу:

1. Визначення робочого місця: Аналізується структура та організація робочого місця, включаючи комп'ютерну систему, робочий стіл, стілець, освітлення та інші елементи. Важливо забезпечити правильну постановку обладнання, оптимальні умови освітлення, регулювання висоти столу та стільця для запобігання неправильній позі та напруженню м'язів.

2. Оцінка ризиків: Визначення потенційних ризиків, пов'язаних з роботою з комп'ютером, таких як перенапруження очей, неправильна постава, пов'язані з довготривалим сидінням та відсутністю перерв. Оцінка факторів, що можуть впливати на здоров'я працівників, таких як шум, радіація, електромагнітні поля тощо.

3. Розробка процедур безпеки: Розроблення та впровадження процедур, спрямованих на запобігання можливим травмам та проблемам зі здоров'ям, пов'язаним з роботою з комп'ютером. Це можуть бути рекомендації щодо правильної постави, використання захисних окулярів, проведення регулярних перерв для відпочинку та розтяжки, а також застосування ергономічних принципів.

4. Навчання та тренування персоналу: Проведення навчання та тренування працівників з питань безпеки праці при роботі з комп'ютером. Це включає ознайомлення з правилами безпеки, освіти щодо користування комп'ютером та його периферійними пристроями, а також навчання профілактичних вправ та розтяжок для зменшення напруги в м'язах.

5. Регулярні перевірки та аудит: Проведення регулярних перевірок робочих місць та комп'ютерних систем для виявлення можливих недоліків та проблем безпеки праці. Аудит безпеки праці допоможе забезпечити виконання встановлених стандартів та процедур безпеки.

Структурно-функціональний аналіз дотримання охорони праці при роботі з комп'ютером допомагає ідентифікувати потенційні ризики та визначити необхідні заходи для забезпечення безпеки працівників. Виконання цього аналізу дозволяє зменшити випадки травм та проблем зі здоров'ям, пов'язаними з роботою з комп'ютером, та забезпечити належні умови праці.

4.3 Обґрунтування організаційно-технічних рекомендацій з охорони праці

Організаційно-технічні рекомендації з охорони праці є важливим етапом в забезпеченні безпеки працівників під час виконання роботи. Для покращення умов праці з комп'ютером та запобігання травмам та проблемам зі здоров'ям, пов'язаним з цим, нижче наведено обґрунтування організаційно-технічних рекомендацій:

Регулярні перевірки технічного стану обладнання: Запровадження систематичних перевірок та обслуговування комп'ютерної техніки з метою виявлення можливих проблем та усунення їх вчасно. Це включає перевірку роботи жорсткого диска, системи охолодження, клавіатури, миші та інших пристроїв.

Забезпечення правильної постановки обладнання: Переконавання, що комп'ютерна система, монітор, клавіатура та миша розташовані на робочому столі відповідно до ергономічних принципів. Важливо забезпечити оптимальну висоту столу та стільця, належне освітлення та правильну поставу для зменшення негативного впливу на здоров'я працівників.

Встановлення регулярних перерв: Рекомендація встановити регулярні перерви для працівників, які працюють з комп'ютером, з метою запобігання перенапруженням та напругам у м'язах. Це може бути перерва кожні 1-2 години для короткого відпочинку, розтяжок та виконання спеціальних вправ для очей та шиї.

Навчання та підвищення свідомості працівників: Проведення навчання та інформування працівників про правила безпеки при роботі з комп'ютером, включаючи правильну поставу, виконання перерв, розтяжок та освітні програми щодо управління стресом та зменшення напруження.

Моніторинг та аналіз показників безпеки праці: Встановлення системи моніторингу та аналізу показників безпеки праці, таких як кількість травматичних випадків, час відпочинку, оцінка задоволеності працівників тощо. Це допоможе ідентифікувати проблемні ситуації та приймати відповідні заходи для покращення безпеки праці.

Обґрунтування організаційно-технічних рекомендацій з охорони праці є важливим етапом у забезпеченні безпеки та здоров'я працівників під час виконання робіт з комп'ютером. Ці рекомендації спрямовані на запобігання можливим ризикам та створення комфортних умов праці, що позитивно впливає на продуктивність та благополуччя працівників.

4.4 Безпека в надзвичайних ситуаціях

Забезпечення захисту населення та території у разі загрози або надзвичайних ситуацій є одним з найважливіших завдань держави. Захист населення реалізується через систему загальнодержавних заходів, що проводяться центральними і місцевими органами виконавчої влади, органами цивільного захисту та підприємствами, які забезпечують організаційні, інженерно-технічні, санітарно-гігієнічні та інші заходи для запобігання і ліквідації наслідків надзвичайних ситуацій. Загрози життєво важливих інтересів поділяються на зовнішні та внутрішні, що виникають під час техногенних, природних катастроф і воєнних конфліктів.

Принципи захисту базуються на положеннях Женевської конвенції щодо захисту жертв війни, можливого характері воєнних дій і реальних можливостях держави щодо створення матеріальної бази захисту. Для зменшення втрат і шкоди в разі надзвичайних ситуацій проводиться спеціальний комплекс заходів, включаючи завчасне створення і підтримку в постійній готовності систем оповіщення населення.

РОЗДІЛ 5.

ДОЦІЛЬНІСТЬ ПРОЕКТУ ТА ПЕРСПЕКТИВИ РОЗВИТКУ

5.1 Практична доцільність

5.1.1 Використання LLM моделей

Використання великих мовних моделей (LLM) є доцільним через кілька важливих аспектів. Перш за все, ці моделі дозволяють автоматизувати значну кількість рутинних задач. Наприклад, автоматичне оброблення запитів, технічна підтримка, допомога з роботою в Word та Excel, генерація тексту, перевірка граматики та орфографії — все це завдання, які можуть виконувати LLM без значного втручання людини. Це знижує потребу в великих людських ресурсах, що, у свою чергу, збільшує ефективність праці.

Завдяки здатності до самонавчання та адаптації до нових контекстів, LLM моделі можуть з часом покращувати свої результати, що позитивно впливає на загальну ефективність роботи організацій. Вони дозволяють значно скоротити час, необхідний для виконання рутинних завдань, і знижують навантаження на персонал. Це дозволяє розгрузити людей від ручної обробки значної частини роботи та перенаправити людські ресурси на більш складні та важливі завдання, що підвищує продуктивність організації в цілому.

Також важливим аспектом є те, що LLM моделі забезпечують зручний і швидкий доступ до інформації, що дозволяє зменшити час на пошук відповідей на стандартні запити, по суті, виступаючи підручною енциклопедією. Це особливо важливо в умовах високої потреби в оперативності і доступності інформації для студентів, викладачів та адміністративного персоналу.

5.1.2 Впровадження бота електронного консультанта

Запровадження бота електронного консультанта для відділу КІТ ЛНУП, також підвищить його роботу. Так, бот може значно зменшити навантаження на співробітників відділу, забезпечуючи автоматичні відповіді на запити студентів, викладачів та інших працівників університету.

Це знижує потребу у живому персоналі для виконання рутинних завдань, таких як надання консультацій щодо адміністративних процедур, таких як списання техніки чи її передача, подачі заяв у відділ, інформації про вирішення банальних проблем по типу підключення до hotspot, чи вирішення збоїв інтернет адаптерів ПК, та інших проблем зв'язаних з комп'ютерною технікою, рівня просунутого користувача.

В той же час, звільнені ресурси відділу КІТ та університету можна буди виділити на покращення роботи відділу в інших аспектах чи на університетські задачі.

Завдяки використанню бота, університет може забезпечити постійну доступність інформації. Бот працює 24/7, що дозволяє студентам та викладачам отримувати консультації в будь-який час, без прив'язки до робочих годин. Це особливо важливо для студентів, які можуть мати запитання після робочого дня або вночі.

Ще одним важливим аспектом є те, що бот допомагає підвищити ефективність обслуговування. Тому що, він може автоматично зберігати історію запитів і відповідей, та направляти на інші ресурси відділу, такі як сайт з поданням заявок, чи сторінки з посібниками по вирішенню інших пересічних проблем, це дозволяє полегшити процес надання консультацій. Студенти і викладачі не потребують повторних звернень до співробітників, якщо їх питання вже було задано, що економить час та знижує завантаженість відділу.

5.1.3 Загальна економічна доцільність

У загальному контексті використання LLM моделей та електронного консультанта є вигідним для університету як з економічної, так і з практичної точки зору. Консультації через бота дозволяють перенаправити часові витрати персоналу на інші задачі, де фізично без працівників не розібратись, такі як критичні збої університетської мережі, ремонт робочих ПК, тощо. Також це допоможе, скоротити час на виконання рутинних завдань та забезпечити більш ефективну роботу всіх підрозділів. Бот електронного консультанта особливо важливий для забезпечення постійного доступу до інформації, підвищення якості обслуговування.

Завдяки такому підходу, університет може підвищити ефективність своїх процесів, що з часом принесе економічні вигоди не тільки завдяки зниженню витрат, але й за рахунок покращення взаємодії з користувачами, що позитивно впливає на загальний імідж та функціонування закладу.

5.1.4 Вплив на імідж університету

З технічної сторони, подібний чат-бот виступає власним аналогом ChatGPT, що покаже університет сучасним та прогресивним. Реалізація такої ініціативи показує, що університет готовий впроваджувати новітні технології та демонструє студентам можливості для уже їх розвитку та навчання.

Також, чат-бот дозволяє швидко вирішувати питання, пов'язані з навчальним процесом, технічною підтримкою або адміністративними послугами. Це формує позитивний досвід користувачів, адже вони можуть отримувати допомогу в будь-який час без потреби звертатися до співробітників. Університет стає більш привабливим для студентів, які цінують зручність та доступність інформації.

Для іноземних студентів чат-бот може стати інструментом, який спрощує адаптацію до університетського середовища. Можливість отримати відповіді кількома мовами вирішує проблему мовного бар'єру, що особливо актуально для закордонних абітурієнтів.

Використання таких технологій покращує репутацію університету серед партнерів і потенційних інвесторів. Інновації в освітньому середовищі часто стають ознакою того, що навчальний заклад розвивається і відповідає сучасним потребам. Це може допомогти університету залучати нові проекти, підтримку чи співпрацю з компаніями.

Запуск чат-бота також можна використати для популяризації університету в медіа чи на конференціях. Інформація про впровадження автоматизованої системи підтримки, демонструє, що університет активно впроваджує інновації. Це підвищує загальну впізнаваність і покращує позиції закладу в освітніх рейтингах. Таким чином, проєкт із впровадження бота не тільки оптимізує внутрішні процеси, а й покращує імідж університету, залучаючи більше студентів, партнерів та підтримку з боку громади.

5.2 Можливі шляхи розвитку чат-бота

Проект бота електронного консультанта відділу КІТ Львівського національного університету природокористування має значний потенціал для подальшого вдосконалення та масштабування. Розвиток його функціональних можливостей може забезпечити кращу інтеграцію в університетську інфраструктуру, розширити аудиторію користувачів та підвищити ефективність взаємодії з інформаційними системами. Нижче розглянуто кілька можливих напрямків розвитку:

1. Розміщення бота на університетському сайті. Інтеграція чат-бота у вебсайт університету дозволить збільшити його доступність для студентів, викладачів та абітурієнтів. Для цього бот може бути розміщений у вигляді віджета на головній сторінці сайту або на окремих сторінках. Основні переваги, це швидкий доступ до консультацій та інформації без необхідності встановлення додаткових додатків та зручність для абітурієнтів, які можуть отримувати відповіді на питання безпосередньо з сайту а також, створення єдиного інформаційного простору для користувачів.

2. Реалізація підтримки Retrieval-Augmented Generation. Впровадження методології RAG дозволить боту знаходити релевантну інформацію в зовнішніх джерелах, таких як внутрішня база знань університету, регламенти чи навчальні матеріали. Це значно підвищить точність і корисність відповідей бота. Для цього необхідно розробити та інтегрувати механізм доступу до бази даних університету або до документів, розміщених у хмарних сервісах та забезпечити регулярне оновлення інформації в базі для підтримки актуальності відповідей.

3. Тонке налаштування (Fine-Tuning) на основі університетських даних. Одним із ключових етапів розвитку є адаптація моделі шляхом тонкого налаштування (fine-tuning) на основі даних, отриманих у відділі КІТ. Це дозволить боту краще розуміти специфічну лексику та контекст запитів університетської спільноти. Щоб це впровадити, необхідно зібрати та анонімізувати історію взаємодій користувачів із ботом, використати ці дані

для донавчання моделі, що покращить її здатність давати релевантні відповіді і провести тестування моделі після донавчання, щоб переконатися у відсутності упереджень чи небажаних змін у поведінці бота.

4. Адаптація бота для роботи на інших платформах. Окрім Telegram, бот може бути адаптований для роботи на інших популярних платформах, як:

Viber : Враховуючи популярність Viber серед студентів та працівників університетів, інтеграція бота на цю платформу значно збільшить аудиторію.

Facebook Messenger : Ця платформа дозволить охопити ширше коло користувачів, включаючи міжнародну спільноту.

Microsoft Teams або Slack : Інтеграція з корпоративними платформами сприятиме використанню бота в адміністративній діяльності університету.

5. Додавання голосового інтерфейсу. Для підвищення зручності взаємодії з ботом можна реалізувати підтримку голосових запитів. Це буде особливо корисно для користувачів з обмеженими можливостями або тих, хто віддає перевагу голосовій взаємодії. Реалізація голосового інтерфейсу може включати Інтеграцію з існуючими голосовими сервісами розпізнавання голосу, до прикладу Google voice recognition, або розробку власного рішення для розпізнавання голосу та генерації відповідей.

6. Розширення функціоналу. Для підвищення інтерактивності бота можна розширити його функціонал, додавши кілька нових можливостей. Одна з них — автоматичне нагадування, коли бот надсилатиме повідомлення про важливі події, дедлайни чи інші значущі заходи. Це дозволить користувачам не пропускати важливу інформацію. Іншою корисною функцією є створення квитків (tickets). Завдяки інтеграції з системою обробки запитів, бот зможе фіксувати складні питання та направляти їх до відповідальних осіб для подальшого вирішення. Також можна додати функцію розсилки новин, яка забезпечить регулярне надсилання актуальної інформації від університету або окремих відділів користувачам.

5.3 Ризики та обмеження

Розробка і впровадження чат-бота супроводжуються низкою ризиків і обмежень, які потрібно враховувати для успішної реалізації проєкту.

Чат-бот залежить від стабільності інфраструктури, яка включає сервер, бази даних та інші технічні компоненти. Нестача обчислювальних ресурсів або збої в роботі сервера, та відключення енергопостачання можуть призводити до недоступності бота. Також можливі технічні помилки у роботі алгоритмів, зокрема у генерації відповідей чи обробці запитів.

Модель може генерувати неточні або помилкові відповіді, особливо якщо інформація у базах знань застаріла або неповна. Це може зменшити зацікавленість у використанні серед користувачів і погіршити їхній досвід взаємодії.

Користувачі можуть не довіряти автоматизованій системі або вважати її недостатньо ефективною порівняно з живою комунікацією. Особливо це стосується складних запитів, для яких потрібна участь фахівця.

Створення, налаштування та підтримка бота вимагають фінансових вкладень. Це включає витрати на, технічні потужності, програмне забезпечення та можливу підписку на сторонні сервіси, такі як Azure, ViberBot, тощо. Крім того, необхідне навчання співробітників для роботи з ботом, що також може збільшити витрати.

Інформація, з якою працює бот, повинна бути актуальною. Без регулярного оновлення бази знань та донавчання моделі якість відповідей може знижуватися. Це потребує додаткових ресурсів і постійної уваги до оновлень.

Повсякденні взаємодії з ботом можуть бути різними залежно від того, як його сприймають студенти чи співробітники. Неправильний тон відповіді або технічні обмеження можуть викликати роздратування користувачів.

Для мінімізації цих ризиків необхідно приділити більшу увагу до планування, впровадження, тестування та постійного допрацювання технології, а також організувати регулярний моніторинг роботи чат-бота, та його донавчання на відгуках від користувачів.

Незважаючи на перелічені ризики та обмеження, впровадження чат-бота є перспективним проєктом, правильний розвиток та донавчання якого, дадуть значне покращення якості технічної підтримки в університеті та піднімуть його престижність.

Чат-бот має потенціал стати не лише зручним інструментом для автоматизації одноманітних задач, але й прикладом сучасного підходу університету. Його використання допоможе знизити навантаження на персонал, підвищити ефективність внутрішніх процесів і покращити досвід взаємодії студентів та викладачів із університетом. Завдяки цьому університет зможе укріпити свій імідж сучасного та технологічно розвиненого навчального закладу.

ВИСНОВКИ

У процесі виконання дипломної роботи було спроектовано та розроблено інформаційну систему у вигляді чат-бота для підтримки та автоматизації сервісних процесів у відділі комп'ютерних інформаційних технологій Львівського національного університету природокористування. Робота охопила весь життєвий цикл розробки програмного продукту — від аналізу предметної області до впровадження прототипу та оцінки його економічної доцільності.

Аналіз предметної області заклав фундамент для розуміння потреб університету у сфері сервісної підтримки. У роботі було досліджено сучасні принципи генеративного штучного інтелекту та сервісної підтримки клієнтів, зокрема LLM-моделей, а також особливості їх використання для автоматизації взаємодії з користувачами. Проведений аналіз дозволив визначити основні вимоги до функціональності чат-бота, а також окреслив потенційні переваги й обмеження таких рішень.

На основі вивчення аналогів було сформовано концепцію чат-бота, яка орієнтується на інтелектуальну обробку запитів користувачів, інтеграцію з різними платформами через BotFramework, а також забезпечення підтримки локальних LLM-моделей. У процесі проектування було визначено архітектуру системи, обрано платформи та інструменти розробки, а також спроектовано базу даних для зберігання історії діалогів та аналітики.

Розроблений прототип чат-бота має модульну структуру, що забезпечує легку модифікацію й розширення функціональності та збільшення каналів зв'язку. Як приклад, він успішно інтегрується з Telegram для взаємодії з кінцевими користувачами та використовує локальні генеративні моделі, які працюють через Ollama та його API для генерації відповідей. Бот оснащений системою логування, яка дозволяє зберігати історію діалогів і відслідковувати роботу системи для подальшого доопрацювання та навчання моделі.

На етапі тестування чат-бот був перевірений у двох середовищах: через BotFramework Emulator, що дозволило оцінити його взаємодію з API, та в

Telegram, де протестовано його продуктивність, зручність використання і відповідність очікуванням користувачів. Результати тестування підтвердили коректну роботу бота і відповідність його функціональним вимогам.

Доцільність проєкту полягає в його здатності знизити навантаження на персонал відділу КІТ, автоматизувати обробку рутинних запитів і забезпечити швидкий доступ до інформації для користувачів. Окрім цього, використання локальних LLM-моделей знижує залежність від зовнішніх сервісів і забезпечує конфіденційність даних.

Таким чином, розробка чат-бота не лише задовольнила поставлені завдання, але й створила платформу для подальшого вдосконалення й масштабування. Запропонована система є ефективним інструментом для автоматизації сервісної підтримки, що може бути адаптованим до потреб інших структурних підрозділів або організацій. Успішна реалізація проєкту підтверджує його практичну цінність і створює основу для подальших досліджень у галузі інтеграції генеративного штучного інтелекту в роботу університетських сервісів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. 5 віртуальних асистентів: підбірка до Міжнародного дня віртуальних помічників [Електронний ресурс] – Режим доступу: <https://it-kharkiv.com/5-virtualnyh-asystentiv-pidbirka-do-mizhnarodnogo-dnya-virtualnyh-pomichnykiv/> . 12.09.2024
2. AWS Chatbot [Електронний ресурс] – Режим доступу: <https://aws.amazon.com/ru/chatbot/> . 15.09.2024
3. Azure AI Bot Service documentation [Електронний ресурс] – Режим доступу: <https://learn.microsoft.com/en-us/azure/bot-service/?view=azure-bot-service-4.0> . 22.10.2024
4. DALL·E by OpenAI: Documentation and Usage Guide [Електронний ресурс] – Режим доступу: <https://platform.openai.com/docs/guides/images> . 20.09.2024
5. Gemma 2 on Ollama [Електронний ресурс] – Режим доступу: <https://ollama.com/library/gemma2> . 22.10.2024
6. Gemma 2-2B IT: Model on Hugging Face [Електронний ресурс] – Режим доступу: <https://huggingface.co/google/gemma-2-2b-it> . 22.10.2024
7. Haykin, S. Neural Networks and Learning Machines. 3rd ed., Pearson Education. 2009. 938 p.
8. How to make a cup cake? The Moment Russian AI Manipulation is Exposed [Електронний ресурс] – Режим доступу: <https://media.meta-peace.org/2024/08/21/political-account-suddenly-introduces-a-cake-recipe/> . 22.10.2024
9. Llama 3.2: Revolutionizing Edge AI and Vision with Open, Customizable Models [Електронний ресурс] – Режим доступу: <https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/> . 15.10.2024
10. Microsoft Azure Bot Service documentation [Електронний ресурс] – Режим доступу: <https://learn.microsoft.com/en-us/azure/bot-service/> . 14.10.2024
11. MidJourney Documentation [Електронний ресурс] – Режим доступу: <https://docs.midjourney.com/> . 30.09.2024
12. Mistral NeMo: New Best Small Model [Електронний ресурс] – Режим доступу: <https://mistral.ai/news/mistral-nemo/> . 14.10.2024
13. Ollama documentation [Електронний ресурс] – Режим доступу: <https://github.com/ollama/ollama/tree/798b107f19ed832d33a6816f11363b42888aaed3/docs> . 30.10.2024
14. Ollama: Understanding the Basics [Електронний ресурс] – Режим доступу: <https://ollama.com/docs> . 16.10.2024

15. OpenAI Documentation [Електронний ресурс] – Режим доступу: <https://platform.openai.com/docs/guides/gpt> .14.10.2024
16. Python: 7 Important Reasons Why You Should Use Python [Електронний ресурс] – Режим доступу: <https://medium.com/@mindfiresolutions.usa/python-7-important-reasons-why-you-should-use-python-5801a98a0d0b> .16.10.2024
17. SQLite Home Page, Documentation [Електронний ресурс] – Режим доступу: <https://www.sqlite.org/docs.html> .5.11.2024
18. Stable Diffusion: High-Resolution Image Generation with AI [Електронний ресурс] – Режим доступу: <https://stability.ai/>.11.09.2024
19. Telegram Bot API [Електронний ресурс] – Режим доступу: <https://core.telegram.org/bots/api> .7.11.2024
20. YAML [Електронний ресурс] – Режим доступу: <https://uk.wikipedia.org/wiki/YAML> .29.10.2024
21. Бутенко Т. А., Сирий В. М. Інформаційні системи та технології: навч. посіб. – Харків: ХНАУ ім. В. В. Докучаєва, 2020. – 207 с.
22. Войналович О. В., Марченко Є. І., Білько Т. О. Охорона праці у сільському господарстві: навч. пос. К: Центр навчальної літератури, 2020. 270 с.
23. Гречанюк В. В. Нейронні мережі та їх застосування в економіці. Харків: Видавництво ХНУ, 2021. 150 с.
24. Данюк В. М. Менеджмент персоналу: Навч. посіб. За ред. В. М. Данюка, В. М. Петюха. 2-ге вид., без змін. К.: КНЕУ, 2006. 398 с.
25. Ємельянова Ю. П. Управління людськими ресурсами: навчально-методичний комплекс / Ю. П. Ємельянова. – Луцьк: Вежа-Друк, 2018. – 24 с.
26. Завальнюк М. Є., Ковалюк О. О. Використання чат-ботів у бізнесі // Вінницький національний технічний університет. 2024. 5 с.
27. Закон України «Про охорону праці». [Електронний ресурс] – Режим доступу: <http://zakon2.rada.gov.ua/laws/show/2694-12> .11.11.2024
28. Колот А. М. Мотивація персоналу: Підручник. Вид. 2-ге, без змін. К.: КНЕУ, 2006. 340 с.
29. Кречетов С. М. Основи нейронних мереж: навчальний посібник. Київ: Національний університет оборони України, 2019. 220 с.

30. Курніков Д. С., Петров С. А. Використання нейронних мереж в економіці // *Juvenisscientia*. 2017. № 6. С. 10-12.

31. Маркетинг в умовах розвитку цифрових технологій: матеріали I Міжнародної науково-практичної інтернет-конференції (24 травня 2024 р). – Луцьк: ЛНТУ, 2024. – 205 с.

32. Нейромережа – що це таке, як працює та навіщо потрібна [Електронний ресурс] – Режим доступу: <https://termin.in.ua/neyromerezha/>. 12.09.2024

33. Про охорону праці [Текст] зі змінами 2024 рік: [закон України: офіц. текст: від 14 жовтня 1992 року № 2694-XII]. Відомості Верховної Ради України. 1992. № 49. 668 ст. редакція від 14.08.2021.

34. 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*. 2013. № 3.

35. Субботін С. О. Нейронні мережі: теорія та практика: навч. посіб. Житомир: Вид. О. О. Євенок, 2020. 184 с.

36. Тимочко В. О., Городецький І. М., Березовецький А. П., Мазур І. Б. та ін. Безпека життєдіяльності та охорона праці. Навч. посібник. Львів: Сполом, 2022. 376 с.

37. Трансформер (архітектура глибокого навчання) [Електронний ресурс] – Режим доступу: [https://uk.wikipedia.org/wiki/Трансформер_\(архітектура_глибокого_навчання\)](https://uk.wikipedia.org/wiki/Трансформер_(архітектура_глибокого_навчання)) 30.09.2024

38. Чат-боти для малого бізнесу: як почати та що враховувати? [Електронний ресурс] – Режим доступу: <https://marketorium.ua/blog/chat-boti-dlya-malogo-biznesu-yak-pochati-ta-scho-vrahovuvati> . 11.09.2024

39. Що таке ШІ-агенти (AI Agents, Intelligent Agents, IA)? [Електронний ресурс] – Режим доступу: <https://thetransmitted.com/adlucem/shho-take-shi-agenty-ai-agents-intelligent-agents-ia/> . 09.10.2024

40. Як ChatGPT вплине на сферу менеджменту й управління персоналом [Електронний ресурс] – Режим доступу: <https://blog.liga.net/user/ovysotskyi/article/50605> . 01.10.2024

ДОДАТКИ

Додаток А

Глосарій основних термінів

Термін	Опис терміну
<i>Основні поняття та категорії предметної області та проекту</i>	
Чат-бот	Програмне забезпечення, яке використовує текстовий або голосовий інтерфейс для взаємодії з користувачами, автоматично відповідаючи на їхні запити.
Нейронна мережа	Процес організації та нагляду за робочою силою, задіяною в операціях на фермі, включаючи такі завдання, як підбір персоналу, складання графіків, навчання та управління продуктивністю.
Нейронна мережа	Математична модель, натхненна структурою та роботою людського мозку, яка використовується для виконання завдань штучного інтелекту, таких як розпізнавання зображень чи обробка текстів.
LLM (Large Language Model)	Велика мовна модель, тренувана на великих обсягах текстових даних для генерування тексту, розуміння мовлення, перекладу та інших завдань обробки природної мови.
Функціональні вимоги	Специфікації, які описують передбачувану поведінку і функціональність системи або програмного додатку.
Платформа	Базова технологія або фреймворк, на якому побудований і працює програмний додаток або система.
Мова програмування	Формальна мова, яка використовується для написання комп'ютерних програм, що складається з набору інструкцій і синтаксичних правил, які визначають поведінку програми.
База даних	Структурована колекція даних, що зберігається та організована в комп'ютерній системі, призначена для ефективного управління та пошуку інформації.
Донавчання (Fine-tuning)	Процес донавчання вже існуючої мовної моделі на додаткових даних для адаптації до конкретних вузькоспеціалізованих задач.
RAG (Retrieval-Augmented Generation)	Метод покращення роботи мовних моделей шляхом інтеграції механізму пошуку релевантної інформації в процес генерації.
Тестування	Процес оцінки функціональності, продуктивності та надійності програмної системи шляхом виконання тестових кейсів та сценаріїв.
Віртуальний помічник	Програмний агент, який допомагає користувачам у виконанні завдань, наприклад, пошуку інформації, управлінні календарем чи обробці запитів, використовуючи штучний інтелект.
API (Application Programming Interface)	Набір інструментів і протоколів для створення програмного забезпечення, що дозволяє взаємодію між різними програмами чи компонентами.
Генеративний штучний інтелект (Generative AI чи ГШІ)	Підрозділ штучного інтелекту, орієнтований на створення нового контенту, такого як текст, зображення, відео чи аудіо.

Додаток Б

Програмний код проекту

main.py

```

import os
import asyncio
import logging
import yaml
from aiohttp import web
from botbuilder.core import (
    BotFrameworkAdapter,
    BotFrameworkAdapterSettings,
    TurnContext,
    MemoryStorage,)
from botbuilder.schema import Activity, ActivityTypes
from dotenv import load_dotenv
from db_handler import DatabaseHandler
import ollama
import tiktoken

# Завантаження змінних середовища з файлу .env
load_dotenv()

def load_messages(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        return yaml.safe_load(file)

# Константи та параметри
TOKEN_LIMIT = 131072 # Ліміт токенів для збереження історії
DEBUG = eval(os.getenv("DEBUG")) # Режим налагодження
LOG_DIR = "debug_logs" # Назва папки для логів
LOG_FILE = os.path.join(LOG_DIR, "debug_log.txt") # Повний шлях до файлу
логу
LLM_MODEL = os.getenv("LLM_MODEL") # Використовувана модель
MESSAGE_LIMIT = int(os.getenv("MESSAGE_LIMIT"))
INACTIVITY_TIMEOUT=int(os.getenv("INACTIVITY_TIMEOUT"))

# Завантаження повідомлень з YAML
MESSAGES = load_messages("LLM_Messages.yaml")["messages"]

# Ініціалізація змінних з YAML
SYS_MESSAGE = MESSAGES["system_message"]
HELLO_MESSAGE = MESSAGES["hello_message"]
HELP_MESSAGE = MESSAGES["help_message"]

# Налаштування логування
logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)

# Створення директорії для логів
if not os.path.exists(LOG_DIR):
    os.makedirs(LOG_DIR)

# Логування повідомлень у файл
def log_to_file(message):
    with open(LOG_FILE, "a", encoding="utf-8") as log_file:
        log_file.write(message + "\n")

# Логування в режимі DEBUG
def debug_log(message):
    if DEBUG:
        print(message)
        log_to_file(message)

```

```

# Клас для обмеження історії повідомлень за кількістю токенів
class TokenLimitedMemory:
    def __init__(self, token_limit=TOKEN_LIMIT, system_message=None):
        self.token_limit = token_limit
        self.history = [{"role": "system", "content": system_message}] if
system_message else []
        self.tokenizer = tiktoken.get_encoding("cl100k_base")

    def count_tokens(self, messages):
        return sum(len(self.tokenizer.encode(msg["content"])) for msg in
messages)

    def trim_history(self):
        while self.count_tokens(self.history) > self.token_limit and
len(self.history) > 2:
            debug_log(f"Видалено повідомлення: {self.history[1]}")
            self.history.pop(1)

    def add_message(self, role, content):
        self.history.append({"role": role, "content": content})
        debug_log(f"Додано повідомлення: {role} -> {content}")
        self.trim_history()

# Клас бота Ollama
class OllamaBot:
    def __init__(self, model=LLM_MODEL, system_message=None):
        self.model = model
        self.memory = TokenLimitedMemory(system_message=system_message)

    def get_response(self, user_message: str) -> str:
        try:
            self.memory.add_message("user", user_message)
            response = ollama.chat(model=self.model,
messages=self.memory.history)
            bot_response = response["message"]["content"]
            self.memory.add_message("assistant", bot_response)
            return bot_response
        except ollama.ResponseError as e:
            logger.error(f"Помилка API Ollama: {e}")
            return "Сталася помилка під час спілкування з Ollama."
        except Exception as e:
            logger.exception("Неочікувана помилка")
            return "Сталася неочікувана помилка. Спробуйте знову пізніше."

    def get_conclusion(self) -> str:
        try:
            if len(self.memory.history) <= 2:
                return 'Користувач завершив розмову без спілкування.'
            self.memory.add_message("user", "ЦЕ СЛУЖБОВЕ ПОВІДОМЛЕННЯ,
КОРИСТУВАЧ ЙОГО НЕ ПОВАЧИТЬ! Підведи підсумок розмови!")
            response = ollama.chat(model=self.model,
messages=self.memory.history)
            return response["message"]["content"]
        except ollama.ResponseError as e:
            logger.error(f"Помилка API Ollama: {e}")
            return "Сталася помилка під час спілкування з Ollama."
        except Exception as e:
            logger.exception("Неочікувана помилка")
            return "Сталася неочікувана помилка. Спробуйте знову пізніше."

# Клас обробника Bot Framework
class BotFrameworkHandler:
    def __init__(self, app_id, app_password):

```



```

        self.adapter_settings = BotFrameworkAdapterSettings(app_id,
app_password)
        self.adapter = BotFrameworkAdapter(self.adapter_settings)
        self.memory = MemoryStorage()
        self.sessions = {}
        self.bot = OllamaBot(model=LLM_MODEL, system_message=SYS_MESSAGE)

    def reconnect_bot(self):
        self.bot = OllamaBot(model=LLM_MODEL, system_message=SYS_MESSAGE)

    async def inactivity_timer(self, user_id):
        try:
            await asyncio.sleep(INACTIVITY_TIMEOUT)
            if user_id in self.sessions:
                await self.sessions[user_id]["db_handler"].mark_ignored()
                del self.sessions[user_id]
                self.reconnect_bot()
                logger.debug(f"Сесія для користувача {user_id} завершена
через бездіяльність.")
            except asyncio.CancelledError:
                logger.debug(f"Таймер бездіяльності для {user_id} скасовано.")

    async def on_turn(self, turn_context: TurnContext):
        user_id = turn_context.activity.from_property.id
        user_name = turn_context.activity.from_property.name or "USER"

        session = self.sessions.setdefault(user_id, {"db_handler":
DatabaseHandler(user_id, user_name), "timer_task": None})
        db_handler = session["db_handler"]
        await db_handler.initialize()
        turn_context.service_url = turn_context.activity.service_url
        if session["timer_task"]:
            session["timer_task"].cancel()
        session["timer_task"] =
asyncio.create_task(self.inactivity_timer(user_id))
        if turn_context.activity.type == ActivityTypes.message:
            user_message = turn_context.activity.text
            await db_handler.save_message("user", user_message)

            if user_message.lower() == "/end" or user_message.lower() ==
"Завершити сесію":
                await self.handle_end_session(user_id, db_handler,
turn_context)
                return

            if session.get("awaiting_feedback"):
                await self.handle_feedback(user_message, user_id, db_handler,
turn_context)
                return

            bot_response = self.bot.get_response(user_message)
            await db_handler.save_message("assistant", bot_response)

            await turn_context.send_activity(bot_response)

            if len(self.bot.memory.history) / 2 - 1 > MESSAGE_LIMIT and not
session.get("completion_prompt_sent"):
                prompt = "Ви вже відправили більше 10 повідомлень. Введіть
'/end' для завершення."
                await turn_context.send_activity(prompt)
                session["completion_prompt_sent"] = True
            elif turn_context.activity.type == ActivityTypes.conversation_update:
                await turn_context.send_activity(HELLO_MESSAGE)
                await turn_context.send_activity(HELP_MESSAGE)

```

```

        self.bot.memory.add_message("assistant", HELLO_MESSAGE)
        await db_handler.save_message("assistant", HELLO_MESSAGE)

    async def handle_end_session(self, user_id, db_handler, turn_context):
        rate_prompt = "Оцініть спілкування від 1 до 10 або залишіть відгук."
        await turn_context.send_activity(rate_prompt)
        await db_handler.save_message("assistant", rate_prompt)
        self.sessions[user_id]["awaiting_feedback"] = True

    async def handle_feedback(self, user_message, user_id, db_handler,
turn_context):
        if user_message.isdigit() and 1 <= int(user_message) <= 10:
            await db_handler.save_feedback(int(user_message))
            await turn_context.send_activity("Дякуємо за вашу оцінку!")
        else:
            await db_handler.save_custom_feedback(user_message)
            await turn_context.send_activity("Дякуємо за ваш відгук!")
        conclusion = self.bot.get_conclusion()
        await db_handler.save_message("conclusion", conclusion)
        #await turn_context.send_activity(conclusion) # Виводить підсумок
розмови в чат.
        del self.sessions[user_id]
        self.reconnect_bot()

# Створення HTTP-сервера
async def messages(req: web.Request):
    body = await req.json()
    activity = Activity().deserialize(body)
    auth_header = req.headers.get("Authorization", "")

    await bot_handler.adapter.process_activity(activity, auth_header,
bot_handler.on_turn)
    return web.Response()

def run_bot_server():
    app = web.Application()
    app.router.add_post("/api/messages", messages)
    web.run_app(app, host="26.110.209.164", port=8080)

if __name__ == "__main__":
    bot_handler = BotFrameworkHandler(
        app_id=os.getenv("MICROSOFT_APP_ID"),
        app_password=os.getenv("MICROSOFT_APP_PASSWORD")
    )
    run_bot_server()

```

db_handler.py

```

import aiosqlite
from datetime import datetime
from pathlib import Path

CHAT_LOGS_FOLDER = Path("chat_logs")
CHAT_LOGS_FOLDER.mkdir(exist_ok=True)

class DatabaseHandler:
    def __init__(self, user_id, user_name=None):
        self.user_id = user_id
        self.user_name = user_name or "USER"
        self.start_time = datetime.now()
        self.original_name =
f"{self.user_name}_{self.start_time.strftime('%Y-%m-%d_%H-%M-%S')}.sqlite"
        self.db_path = CHAT_LOGS_FOLDER / self.original_name

    async def initialize(self):
        async with aiosqlite.connect(self.db_path) as conn:
            await conn.execute('''
                CREATE TABLE IF NOT EXISTS messages (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    role TEXT,
                    content TEXT,
                    timestamp TIME
                )
            ''')
            await conn.commit()

    async def save_message(self, role, content):
        async with aiosqlite.connect(self.db_path) as conn:
            await conn.execute('''
                INSERT INTO messages (role, content, timestamp)
                VALUES (?, ?, ?)
            ''', (role, content, datetime.now().strftime('%Y-%m-%d_%H-%M-%S')))
            await conn.commit()

    async def mark_ignored(self):
        async with aiosqlite.connect(self.db_path) as conn:
            await conn.execute('''
                INSERT INTO messages (role, content, timestamp)
                VALUES ("system", "[CONVERSATION IGNORED]", ?)
            ''', (datetime.now().strftime('%Y-%m-%d_%H-%M-%S'),))
            await conn.commit()
            self.rename_database("IGNORED")

    async def save_feedback(self, feedback):
        async with aiosqlite.connect(self.db_path) as conn:
            # Оновлення останнього запису
            await conn.execute('''
                UPDATE messages
                SET role = "rate",
                    content = ?,
                    timestamp = ?
                WHERE rowid = (SELECT MAX(rowid) FROM messages)
            ''', (feedback, datetime.now().strftime('%Y-%m-%d_%H-%M-%S')))
            await conn.commit()
            self.rename_database(f"RATED_{feedback}")

    async def save_custom_feedback(self, feedback_text):

```

```

    async with aisqlite.connect(self.db_path) as conn:
        # Оновлення останнього запису
        await conn.execute('''
            UPDATE messages
            SET role = "feedback",
                content = ?,
                timestamp = ?
            WHERE rowid = (SELECT MAX(rowid) FROM messages)
        ''', (feedback_text, datetime.now().strftime('%Y-%m-%d_%H-%M-%S')))

        await conn.commit()
        self.rename_database("FEEDBACK")

    def rename_database(self, prefix):
        new_name = f"{prefix}_{self.original_name}"
        new_path = CHAT_LOGS_FOLDER / new_name
        self.db_path.rename(new_path)
        self.db_path = new_path # Оновлення шляху

```

tg_bot.py

```

import logging
import os
import dotenv
import requests
from flask import Flask, request, jsonify
from telegram import Update
from telegram import ReplyKeyboardMarkup, KeyboardButton
from telegram.ext import Application, CommandHandler, MessageHandler,
    CallbackContext, filters
import uuid
from datetime import datetime
from threading import Thread

dotenv.load_dotenv()

class BotServer:
    def __init__(self):
        # Ініціалізація Flask
        self.app = Flask(__name__)
        # Ініціалізація Telegram
        self.telegram_token = os.getenv("TELEGRAM_TOKEN")
        self.application =
            Application.builder().token(self.telegram_token).build()
        # Глобальні змінні
        self.last_received_messages = []

        # Налаштування логування
        logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s -
            %(message)s',
                                level=logging.INFO)
        self.logger = logging.getLogger(__name__)

        # URL API для відправки повідомлень
        self.api_url = 'http://26.110.209.164:8080/api/messages'

        # Додати маршрути Flask
        self._add_flask_routes()

        # Додати хендлери Telegram
        self._add_telegram_handlers()

```

```

def _add_flask_routes(self):
    @self.app.route('/v3/conversations/<conversation_id>/activities',
methods=['POST'])
    def incoming_message(conversation_id):
        data = request.json
        self.last_received_messages.append(data.get('text', None))
        self.logger.info(f"Отримано повідомлення: {data}")
        return jsonify({"status": "received", "message": "Message
received"}), 200

def _add_telegram_handlers(self):
    # Команда /start
    async def start(update: Update, context: CallbackContext):
        user_message = update.message.text
        user_name = update.message.from_user.username # Отримуємо ім'я
користувача Telegram
        user_id = update.message.from_user.id
        self.send_message_to_server(user_message, user_id, user_name,
"conversationUpdate")
        # Відповідь із Flask

        if self.last_received_messages:
            for message in self.last_received_messages:
                await update.message.reply_text(message)
            self.last_received_messages.clear()
        else:
            await update.message.reply_text("Немає отриманих повідомлень
із Flask.")

    async def handle_message(update: Update, context: CallbackContext):
        user_message = update.message.text
        user_name = update.message.from_user.username # Отримуємо ім'я
користувача Telegram
        user_id = update.message.from_user.id
        # Відправка повідомлення на сервер
        self.send_message_to_server(user_message, user_id, user_name)
        self.logger.info(f"Повідомлення користувача {user_name}:
{user_message}")

        # await update.message.reply_text("Повідомлення надіслано на
сервер.")

        # Відповідь із Flask
        if self.last_received_messages:
            for message in self.last_received_messages:
                if message == "Ви вже відправили більше 10 повідомлень.
Введіть '/end' для завершення.":
                    keyboard = [[KeyboardButton("Завершити сесію")]]
                    reply_markup = ReplyKeyboardMarkup(keyboard,
resize_keyboard=True, one_time_keyboard=True)

                    prompt = "Ви вже спілкуєтесь досить довго, направити
вас до заявки у відділ КІТ і завершити сесію?"
                    await update.message.reply_text(prompt,
reply_markup=reply_markup)
                    break
                else:
                    await update.message.reply_text(message)

            # Очищаємо список повідомлень після їх обробки
            self.last_received_messages.clear()
        else:
            await update.message.reply_text("Немає отриманих повідомлень

```

```

из Flask.")

    # Додати хендлери
    self.application.add_handler(CommandHandler("start", start))
    self.application.add_handler(MessageHandler(filters.TEXT &
~filters.COMMAND & filters.ChatType.PRIVATE, handle_message))
    self.application.add_handler(MessageHandler(filters.REPLY &
~filters.COMMAND, handle_message))
    self.application.add_handler(MessageHandler(filters.TEXT &
~filters.COMMAND, handle_message))

    def send_message_to_server(self, message_text, user_id, user_name,
type_message='message'):
        client_activity_id = str(uuid.uuid4())
        sequence_id = int(datetime.now().timestamp() * 1000)
        data = {
            "channelData": {
                "clientActivityID": client_activity_id,
                "webchat:sequence-id": sequence_id,
                "state": "sent",
                "webchat:send-status": "sent"
            },
            "text": message_text,
            "textFormat": "plain",
            "type": type_message,
            "from": {
                "id": user_id,
                "name": user_name, # Використовуємо ім'я користувача
Telegram
                "role": "user"
            },
            "conversation": {"id": str(user_id) + "|livechat"},
            "timestamp": datetime.now().isoformat(),
            "serviceUrl": "http://localhost:60174"
        }

        response = requests.post(self.api_url, json=data)
        return response.status_code

    def run_flask(self):
        self.app.run(host="localhost", port=60174, threaded=True)

    def run_telegram(self):
        self.application.run_polling()

    def run(self):
        # Запуск Flask у окремому потоці
        flask_thread = Thread(target=self.run_flask)
        flask_thread.start()

        # Запуск Telegram
        self.run_telegram()
if __name__ == "__main__":
    bot_server = BotServer()
    bot_server.run()

```

LLM_Messages.yaml

```
messages:
  system_message: |
    (НІКОЛИ НЕ ПИШИ РОСІЙСЬКОЮ!) Пиши людяно, стисло і по ділу, відповідай лише на те.
    Ти електронний консультант відділу КІТ Львівського Національного Університету
    Природокористування (ЛНУП)
    і повинен займатись сервісною підтримкою користувачів, а саме допомагати їм, коли
    виникають проблеми з їх комп'ютерами.
    Голова відділу КІТ Богдан Шувар Іванович, його кабінет 132, навпроти кабінету КІТ.
    Університетський wifi називається Network. Для підключення до інтернету
    непотрібний пароль,
    але щоб отримати доступ, потрібно увійти в http://hotspot.lnup.edu.ua та увійти
    через корпоративну пошту або як гість.
    При провідному підключенні також потрібно проходити через hotspot.
    [(Згадуй про заявку в КІТ, тільки якщо людина попросила або ти не справляєшся з
    допомогою!)]
    Щоб подати заявку у відділ, потрібно перейти за посиланням https://e.surl.li/kit ,
    після чого відкриється форма, яку потрібно заповнити],
    щоб увійти потрібно залогінитись через корпоративну пошту @lnup.edu.ua. [УВАЖНО
    через форму можна лише повідомити відділ КІТ про технічну несправність,
    за надання нової техніки пишеть Друзюку! (не згадуй про це, поки не спитають за
    нову техніку напряму, Друзюк не займається ремонтом,
    йому потрібно писати заяву лише в випадку, коли потрібно виписати нове
    обладнання!)]. Університет ЛНУП заснований у 1856 як аграрна школа.
    Ректор: Лопушняк Василь Іванович – в.о. ректора. [Якщо вам необхідна нова техніка,
    пишеть заяву на ім'я Друзюка Василя Миколайовича (не згадуй про це, поки не
    спитають за нову техніку напряму)].
    Відділ КІТ знаходиться в 121-му кабінеті в головному корпусі. Якщо людина залишала
    заявку, але ніхто не прийшов, попроси,
    ще раз написати заявку у відділ КІТ.""
  hello_message: "Доброго дня, я електронний консультант відділу КІТ. \n\nНе соромтесь
  звертатись, якщо вам необхідна технічна підтримка!"
  help_message: "Щоб побачити список команд бота, напишіть /help \n\nЩоб завершити
  розмову, напишіть '/end'."
```

.env

```
# Налаштування LLM
DEBUG=True# або False
LLM_MODEL=gemma2:9b
# Ліміт токенів передаваних в LLM
TOKEN_LIMIT=131072
# Ліміт повідомлень перед тим, як бот запропонує звернутись у відділ КІТ
MESSAGE_LIMIT=10
# Час очікування відповіді користувача в секундах, перез завершенням сесії (1200 = 20
хв, стандартне значення)
INACTIVITY_TIMEOUT=1200
# telegram токен
TELEGRAM_TOKEN=6110479240:AAFvpnaR2Re0jPcPEl4T9hc5150iDKKQVM
# microsoft токен
MICROSOFT_APP_ID=""
MICROSOFT_APP_PASSWORD=""
```

requirements.txt

```
PyYAML~=6.0.2
ollama~=0.4.0
tiktoken~=0.8.0
aiohttp~=3.11.7
python-dotenv~=1.0.1
requests~=2.32.3
Flask~=3.1.0
python-telegram-bot~=21.7
aiosqlite~=0.20.0
```