МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Навчально-науковий інститут кібернетики, інформаційних технологій та інженерії

"До захисту допущена"
Зав. кафедри комп'ютерних наук та прикладної математики
_____
«___» _____ 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

Проектування та розробка інтелектуального чат-бота

Виконав: Набухотний Роман Васильович
_____
(прізвище, ім’я, по батькові) (підпис)
група КН-41

Керівник: **доцент, к.т.н. Жуковський В.В.**
_____
(науковий ступінь, вчене звання, посада, прізвище, ініціали) (підпис)

Рівне – 2024

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота: 67 с., 6 рисунків, 22 джерела.

Метою кваліфікаційної роботи є розробка інтелектуального чат-бота для університету, який здатний відповідати на запити абітурієнтів, студентів та інших зацікавлених осіб, використовуючи базу документів та сучасні технології обробки природної мови (NLP) та оптичного розпізнавання символів (OCR).

Об’єкт дослідження – процес автоматизації надання інформації в університеті за допомогою чат-бота.

Предметом дослідження – методи та засоби розробки інтелектуального чат-бота з використанням сучасних технологій NLP та OCR.

Методи дослідження – технології TypeScript, Telegram (Telegraf), ChatGPT, Hono, Svelte, Tiktoken, Tesseract.js, PostgreSQL, Drizzle ORM, Office-Text-Extractor, PDFjs-dist, Sharp, JSZip, Node.js.

Розроблений чат-бот для університету дозволяє автоматизувати відповіді на типові запити, значно знижуючи навантаження на адміністративний персонал та підвищуючи якість обслуговування студентів. Інноваційність роботи полягає у використанні передових технологій NLP та OCR для забезпечення швидкого та точного доступу до необхідної інформації.

Ключові слова: чат-бот, Telegram, NLP, OCR, TypeScript, Node.js, PostgreSQL.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

NLP (Natural Language Processing) - Обробка природної мови

OCR (Optical Character Recognition) - Оптичне розпізнавання символів

UI/UX (User Interface/User Experience) - Інтерфейс користувача/досвід користувача

API (Application Programming Interface) - Інтерфейс програмування додатків

PDF (Portable Document Format) - Портативний формат документів

SQL (Structured Query Language) - Мова структурованих запитів

JS (JavaScript) - Мова програмування JavaScript

TypeScript - Надбудова над JavaScript з підтримкою статичної типізації

ChatGPT - Модель обробки природної мови, розроблена OpenAI

Embedding - Векторне представлення тексту

Telegram - Платформа для обміну повідомленнями

Telegraf - Бібліотека для створення ботів на платформі Telegram

Hono - Веб-фреймворк для Node.js

Svelte - Фреймворк для створення веб-інтерфейсів

Tiktoken - Бібліотека для роботи з токенами

Tesseract.js - Бібліотека для оптичного розпізнавання символів на JavaScript

PostgreSQL - Об'єктно-реляційна система управління базами даних

Drizzle ORM - Об'єктно-реляційний маппер для TypeScript та JavaScript

Office-Text-Extractor - Бібліотека для витягування тексту з документів Microsoft Office

PDFjs-dist - Бібліотека для роботи з PDF-документами на JavaScript

Sharp - Бібліотека для обробки зображень на Node.js

JSZip - Бібліотека для створення, читання та обробки ZIP-архівів на JavaScript

Node.js - Середовище виконання JavaScript на сервері

Cron - це програмне забезпечення для автоматизації завдань, яке дозволяє виконувати команди або скрипти за розкладом

ВСТУП

У сучасному світі інформаційних технологій автоматизація процесів стає все більш важливою. Особливо це стосується освітніх установ, де велика кількість студентів, абітурієнтів та інших зацікавлених осіб постійно потребує доступу до різноманітної інформації. Одним із ефективних рішень для автоматизації та оптимізації інформаційних процесів є розробка інтелектуальних чат-ботів.

Проектування та розробка інтелектуального чат-бота для університету є актуальною темою, яка здатна значно покращити комунікацію між студентами, адміністративним персоналом та іншими зацікавленими сторонами. Сучасні студенти очікують на швидкий та зручний доступ до інформації, такої як розклад занять, інформація про курси, процес вступу та інші адміністративні деталі. Використання чат-бота для цих цілей не лише підвищує ефективність обслуговування, але й знижує навантаження на адміністративний персонал, дозволяючи їм зосередитися на більш важливих задачах.

Наукова і практична новизна даної роботи полягає у впровадженні сучасних технологій обробки природної мови (NLP) та оптичного розпізнавання символів (OCR) для створення високоефективного та зручного інструменту. Використання ChatGPT для обробки текстових запитів користувачів забезпечує високу точність та релевантність відповідей, що дозволяє користувачам отримувати необхідну інформацію швидко та зручно. Додаткова можливість обробки сканованих документів завдяки технології OCR дозволяє автоматично витягувати текстову інформацію з зображень, що є особливо корисним для роботи з великим обсягом документів у нецифровому форматі.

Мета цієї кваліфікаційної роботи полягає у створенні чат-бота, який буде здатний відповідати на питання абітурієнтів, студентів та інших зацікавлених осіб, використовуючи дані з бази документів. Це включає інтеграцію сучасних

технологій NLP та OCR для забезпечення максимальної ефективності та зручності роботи чат-бота.

Для досягнення поставленої мети було визначено такі завдання:

1. Розробити архітектуру чат-бота, яка дозволяє ефективно обробляти текстові запити користувачів.
2. Інтегрувати алгоритми обробки природної мови (NLP), реалізовані в ChatGPT, для аналізу та розуміння запитів.
3. Впровадити технологію оптичного розпізнавання символів (OCR) для зчитування інформації зі сканованих документів.
4. Розробити базу даних для зберігання та швидкого доступу до необхідної інформації.
5. Створити інтуїтивно зрозумілий інтерфейс користувача для зручної взаємодії з чат-ботом.
6. Провести всебічне тестування системи для забезпечення стабільної та точної роботи чат-бота.

Основні методи і підходи до вирішення поставленої проблеми:

1. Обробка природної мови (NLP): Використання алгоритмів NLP, реалізованих в ChatGPT, для розуміння та обробки текстових запитів користувачів. Це включає синтаксичний та семантичний аналіз запитів для точного визначення намірів користувачів.
2. Оптичне розпізнавання символів (OCR): Використання технології OCR для зчитування інформації зі сканованих документів. Це дозволяє системі автоматично витягувати текстову інформацію з зображень, забезпечуючи доступ до даних, які зберігаються в нецифровому форматі.
3. Бази даних: Створення та управління базою даних, де зберігаються документи та інша інформація, необхідна для роботи чат-бота. База даних повинна бути оптимізована для швидкого пошуку та витягання інформації.

4. Інтерфейси користувача (UI/UX): Розробка інтуїтивно зрозумілого інтерфейсу користувача для взаємодії з чат-ботом. Це включає в себе як текстовий інтерфейс, так і можливості інтеграції з іншими платформами, такими як Telegram.
5. Тестування та валідація: Проведення всебічного тестування системи на різних етапах розробки для забезпечення її стабільної та точної роботи.

Таким чином, розробка інтелектуального чат-бота для університету є актуальною та інноваційною задачею, яка має значний потенціал для покращення інформаційного обслуговування студентів та оптимізації роботи адміністративного персоналу.

РОЗДІЛ 1. ТЕОРЕТИЧНА ЧАСТИНА

1.1. Системний аналіз предметної області

Розробка інтелектуальних чат-ботів є однією з найактуальніших тем у сфері інформаційних технологій. Сучасні інформаційні системи все більше інтегруються з технологіями обробки природної мови (NLP) для забезпечення зручності користувачів. Університети, зокрема, мають великий потенціал для використання таких технологій для покращення обслуговування студентів, абітурієнтів та інших зацікавлених осіб.

1.2. Складові компоненти системи

Розробка інтелектуального чат-бота включає кілька ключових компонентів:

1. Платформа для чат-бота: Telegram була обрана як платформа для реалізації чат-бота завдяки своїй популярності та зручності використання. Telegram API забезпечує необхідний функціонал для інтеграції з зовнішніми системами та базами даних.
2. Система обробки запитів: Для обробки запитів користувачів використовується модель обробки природної мови ChatGPT, яка здатна розпізнавати та інтерпретувати текстові запити з високою точністю.
3. База даних: База даних використовується для зберігання документів, з яких чат-бот витягує інформацію. Всі документи розділяються на частини, для яких створюються векторні представлення (embeddings), що дозволяють швидко знаходити релевантну інформацію за запитами користувачів.
4. Оптичне розпізнавання символів (OCR): Ця технологія використовується для зчитування тексту зі сканованих документів, забезпечуючи доступ до інформації, яка зберігається в нецифровому форматі.
5. Інтерфейс для завантаження документів: Створено окремий веб-сайт, де адміністративний персонал може завантажувати документи в базу

даних. Це забезпечує централізоване управління інформацією та полегшує процес оновлення даних.

1.3. Генезис розвитку наукових поглядів на проблему

Обробка природної мови (NLP) є однією з найважливіших галузей штучного інтелекту. Починаючи з 1950-х років, дослідники розробляли методи та алгоритми для автоматичного розуміння та генерації людської мови. Однак лише останні кілька десятиліть завдяки розвитку глибокого навчання (Deep Learning) ми побачили значні прориви в цій області.

Перші значні досягнення в NLP були пов'язані з розвитком алгоритмів на основі правил, які мали обмежені можливості та були дуже складними у розробці та підтримці. З розвитком статистичних методів ситуація значно покращилася, але справжній прорив стався з появою нейронних мереж та глибокого навчання. Моделі, такі як Word2Vec, GloVe та, зрештою, трансформери, значно підвищили ефективність NLP-систем.

Трансформерна архітектура, представлена в роботі "Attention is All You Need"[1], стала основою для створення моделей, таких як BERT[2] та GPT (Generative Pre-trained Transformer) від OpenAI. GPT-3, на якому базується ChatGPT, є однією з найсучасніших моделей для обробки природної мови, здатною генерувати зв'язний текст та розуміти складні запити користувачів.

1.4. Аналіз новітніх наукових матеріалів і розробок

Сучасні дослідження в галузі NLP фокусуються на поліпшенні якості розпізнавання та генерації тексту, а також на розширенні застосувань цих технологій. Одним з важливих напрямків є розробка моделей, які здатні працювати з багатомовними текстами, що є особливо актуальним для університетів з інтернаціональною аудиторією.

Останні дослідження показують, що інтеграція технологій NLP з оптичним розпізнаванням символів (OCR) відкриває нові можливості для автоматизації обробки текстової інформації. Наприклад, роботи Liu та інших демонструють успішне застосування глибоких нейронних мереж для поліпшення точності OCR при обробці документів з різноманітними шрифтами та умовами сканування[3].

Інший важливий аспект – це використання векторних представлень тексту (embeddings), які дозволяють ефективно зіставляти запити користувачів з релевантними документами. Дослідження, проведені Mikolov та іншими, заклали основу для створення ефективних алгоритмів, таких як Word2Vec, що використовуються для побудови embeddings[4]. Подальші розробки, такі як BERT та GPT, продемонстрували ще більшу точність та гнучкість у роботі з текстовими даними.

1.5. Аналіз існуючих рішень

1.5.1. Вступ

У сучасному світі інформаційних технологій та автоматизації процесів інтелектуальні чат-боти стали важливим інструментом для покращення комунікації та доступу до інформації в різних галузях, включаючи освіту. Розробка чат-ботів для університетів має на меті полегшити доступ студентів та абітурієнтів до необхідної інформації, а також знизити навантаження на адміністративний персонал. У цьому розділі буде проведено аналіз існуючих рішень у сфері інтелектуальних чат-ботів, що використовуються в освітніх установах, з метою визначення їх переваг та недоліків.

1.5.2. Існуючі рішення для розробки чат-ботів

Існує кілька популярних платформ та інструментів для розробки чат-ботів, кожен з яких має свої особливості та можливості. До найбільш розповсюджених належать:

1. Dialogflow від Google

2. Microsoft Bot Framework
3. IBM Watson Assistant
4. Amazon Lex
5. Rasa

1.5.2.1. Dialogflow

Dialogflow є одним з найпопулярніших інструментів для розробки чат-ботів, що використовує технології машинного навчання та обробки природної мови[5]. Основні переваги Dialogflow включають:

- Легка інтеграція з іншими сервісами Google.
- Підтримка багатьох мов.
- Зручний інтерфейс для налаштування діалогів.
- Можливість інтеграції з різними платформами, включаючи Telegram, Facebook Messenger та інші.

Проте Dialogflow має деякі обмеження, такі як залежність від інфраструктури Google та обмеження у налаштуванні складних діалогових сценаріїв.

1.5.2.2. Microsoft Bot Framework

Microsoft Bot Framework надає інструменти для створення, тестування та розгортання чат-ботів[6]. Основні переваги включають:

- Інтеграція з іншими сервісами Microsoft, такими як Azure та Office 365.
- Підтримка різних мов програмування.
- Широкий спектр можливостей для налаштування та масштабування.
- Підтримка різних каналів комунікації.

Недоліки Microsoft Bot Framework включають складність налаштування та високі вимоги до технічних знань.

1.5.2.3. IBM Watson Assistant

IBM Watson Assistant є потужним інструментом для створення чат-ботів з використанням технологій штучного інтелекту[7]. Основні переваги включають:

- Висока точність розпізнавання природної мови.
- Можливість інтеграції з іншими сервісами IBM.
- Підтримка широкого спектра мов.
- Потужні інструменти для аналізу та оптимізації діалогів.

Основними недоліками IBM Watson Assistant є висока вартість та складність налаштування для нових користувачів.

1.5.2.4. Amazon Lex

Amazon Lex є сервісом для створення чат-ботів, що використовує ті самі технології, що й Alexa. Основні переваги включають:

- Інтеграція з іншими сервісами Amazon, такими як AWS Lambda.
- Підтримка голосових та текстових інтерфейсів.
- Зручний інтерфейс для створення та налаштування ботів.

Недоліки Amazon Lex включають обмежену підтримку мов та складність інтеграції з сервісами, що не належать до екосистеми Amazon.

1.5.2.5. Rasa

Rasa є open-source платформою для створення чат-ботів з використанням технологій машинного навчання. Основні переваги включають:

- Відкритий код та можливість повного контролю над налаштуваннями.
- Підтримка складних діалогових сценаріїв.
- Інтеграція з різними мовами програмування та фреймворками.

Основними недоліками Rasa є складність налаштування та потреба у високих технічних знаннях.

1.5.3. Аналіз використання чат-ботів в освітніх установах

Інтелектуальні чат-боти вже активно використовуються в багатьох освітніх установах для автоматизації процесів обслуговування студентів. Розглянемо кілька прикладів успішного впровадження чат-ботів у різних університетах.

1.5.3.1. Nova Southeastern University (NSU)

Для університету створено чат-бота на ім'я Julie, який використовує останні фреймворки та когнітивні сервіси Microsoft, зокрема Azure Services та Microsoft Bot Framework. Julie допомагає студентам, викладачам та співробітникам з різними питаннями, такими як розклад занять, опції харчування на кампусі, та інші повсякденні завдання. Використання цієї технології дозволяє надавати підтримку 24/7, оптимізуючи університетський досвід для всіх користувачів[8].

1.5.3.2. York University

Для університету створено чат-бота на ім'я SAVY, який використовує IBM Watson Assistant для надання підтримки студентам. SAVY відповідає на запити щодо академічних питань, таких як розклад занять та адміністративні питання. Цей бот також інтегрований з іншими університетськими сервісами, що підвищує точність відповідей та покращує користувацький досвід[9].

1.5.3.3. Can Tho University

Університет використовує чат-бот на базі Rasa для підтримки студентів. Цей чат-бот відповідає на запити студентів щодо різних академічних питань, таких як розклад занять, курси та інші адміністративні питання. Бот інтегрується з внутрішніми системами університету, забезпечуючи точність відповідей та гнучкість налаштувань[10].

1.5.4. Переваги та недоліки існуючих рішень

Існуючі рішення для створення чат-ботів мають свої переваги та недоліки. Основними перевагами є:

- Автоматизація відповідей на запити користувачів.
- Зниження навантаження на адміністративний персонал.
- Підвищення якості обслуговування студентів.

Проте, існують і деякі недоліки:

- Висока вартість деяких платформ.
- Складність налаштування та інтеграції.
- Залежність від конкретних сервісів та інфраструктури.

1.5.5. Висновки

Аналіз існуючих рішень показує, що інтелектуальні чат-боти є ефективним інструментом для покращення комунікації та доступу до інформації в освітніх установах. Існує багато різних платформ та інструментів для створення чат-ботів, кожен з яких має свої особливості та можливості. Вибір конкретної платформи залежить від потреб та ресурсів освітньої установи.

Розробка чат-бота для університету з використанням сучасних технологій обробки природної мови та оптичного розпізнавання символів дозволяє забезпечити високий рівень обслуговування студентів та адміністративного персоналу. Використання ChatGPT та OCR технологій у розробці чат-бота забезпечує високу точність відповідей та зручний доступ до інформації, що є важливим кроком у напрямку автоматизації інформаційних процесів в освітніх установах.

Таким чином, розробка інтелектуального чат-бота для університету є актуальним та перспективним напрямком досліджень, що має значний потенціал для покращення якості обслуговування та підвищення ефективності роботи освітніх установ.

1.6. Оцінка існуючих підходів до вирішення проблеми

Існуючі підходи до створення інтелектуальних чат-ботів демонструють різноманітність методів та технологій. Однією з основних проблем є забезпечення точності відповідей на запити користувачів, що вимагає ефективного поєднання алгоритмів NLP та методів роботи з базами даних.

Використання моделей трансформерної архітектури, таких як GPT-3, дозволяє значно підвищити якість обробки природної мови. Проте, навіть такі моделі можуть давати некоректні або нерелевантні відповіді, якщо дані в базі не структуровані належним чином або якщо відсутня необхідна інформація. Тому важливою частиною роботи є забезпечення якості та актуальності даних, які використовуються для навчання та роботи чат-бота.

Науковці також вказують на важливість користувацького досвіду (UX) при взаємодії з чат-ботами. Згідно з дослідженнями, проведеними Hu та іншими, користувачі віддають перевагу інтерфейсам, які забезпечують швидкий доступ до інформації та інтуїтивно зрозумілі у використанні[11]. В цьому контексті вибір платформи Telegram є обґрунтованим, оскільки вона надає зручний та добре знайомий інтерфейс для більшості користувачів.

1.7. Висновки

Системний аналіз предметної області та детальний огляд складових компонентів системи дозволяють зробити висновок про актуальність та перспективність розробки інтелектуального чат-бота для університету. Використання сучасних технологій NLP, таких як ChatGPT, у поєднанні з методами OCR та векторними представленнями тексту забезпечує високу якість обробки запитів користувачів та доступ до необхідної інформації.

Аналіз новітніх наукових матеріалів і розробок свідчить про значні досягнення в галузі обробки природної мови та можливості їх ефективного застосування для вирішення практичних завдань. Оцінка існуючих підходів

підкреслює важливість інтеграції якісних даних та зручного користувацького інтерфейсу для забезпечення успішної роботи чат-бота.

Таким чином, теоретична частина дослідження підтверджує обґрунтованість обраного напрямку та визначає основні завдання для подальшого розвитку та впровадження розробленої системи в університетське середовище.

РОЗДІЛ 2. АНАЛІТИЧНА ЧАСТИНА

2.1. Проектування системних зв’язків

У цьому розділі розглядається проектування системних зв’язків інтелектуального чат-бота для університету, який забезпечує ефективний доступ до інформації для студентів та адміністративного персоналу. Проектування системних зв’язків включає визначення архітектури системи, опис основних компонентів та взаємодії між ними.

2.1.1. Архітектура системи

Архітектура чат-бота складається з наступних основних компонентів:

1. Платформа для чат-бота (Telegram): Використовується як основний інтерфейс взаємодії користувачів з системою.
2. Система обробки запитів (ChatGPT): Відповідає за розуміння та генерацію відповідей на запити користувачів.
3. База даних: Зберігає документи та інші дані, які використовуються для надання відповідей.
4. Система оптичного розпізнавання символів (OCR): Використовується для зчитування тексту зі сканованих документів.
5. Веб-інтерфейс для завантаження документів: Дозволяє адміністративному персоналу завантажувати документи в базу даних.

2.1.2. Взаємодія між компонентами

Взаємодія між компонентами системи здійснюється наступним чином:

1. Користувачі надсилають запити через Telegram.
2. Запити передаються на обробку в систему ChatGPT.
3. ChatGPT створює векторне представлення (embedding) запиту.
4. Векторне представлення використовується для пошуку релевантних частин документів у базі даних.
5. Відібрані частини документів разом із запитом передаються в ChatGPT для генерації відповіді.

6. Відповідь надсилається користувачу через Telegram.

2.2. Алгоритми вирішення локальних задач

Проектування алгоритмів для вирішення локальних задач є ключовою частиною розробки системи. Основні алгоритми, що використовуються, включають алгоритм обробки запитів, алгоритм оптичного розпізнавання символів (OCR) та алгоритм пошуку релевантних частин документів.

2.2.1. Алгоритм обробки запитів

Алгоритм обробки запитів складається з наступних етапів:

1. Отримання запиту: Користувач надсилає запит через Telegram.
2. Створення векторного представлення: Використовуючи модель ChatGPT, запит перетворюється на векторне представлення.
3. Пошук у базі даних: Векторне представлення використовується для пошуку найбільш релевантних частин документів у базі даних.
4. Генерація відповіді: На основі знайдених частин документів та запиту, ChatGPT генерує відповідь.
5. Відправлення відповіді: Відповідь надсилається користувачу через Telegram.

2.2.2. Алгоритм оптичного розпізнавання символів (OCR)

Алгоритм OCR включає наступні кроки:

1. Завантаження сканованого документа: Адміністративний персонал завантажує сканований документ через веб-інтерфейс.
2. Обробка зображення: Зображення документа обробляється для підвищення якості (наприклад, видалення шуму).
3. Розпізнавання тексту: Текст на зображенні розпізнається за допомогою технології OCR.
4. Збереження тексту: Розпізнаний текст розділяється на частини та зберігається в базі даних з відповідними векторними представленнями.

2.2.3. Алгоритм пошуку релевантних частин документів

Для пошуку релевантних частин документів використовується наступний алгоритм:

1. Створення векторного представлення запиту: Запит користувача перетворюється на векторне представлення за допомогою ChatGPT.
2. Порівняння з векторними представленнями частин документів: Виконується порівняння векторного представлення запиту з векторними представленнями частин документів у базі даних.
3. Відбір релевантних частин: Відбираються частини документів з найвищою схожістю до запиту.
4. Генерація відповіді: Відібрані частини документів використовуються для формування відповіді на запит користувача.

2.3. Власні дослідження автора

Розробка інтелектуального чат-бота для університету включала кілька етапів досліджень, проведених в рамках курсових та науково-дослідних робіт. Основні результати цих досліджень включають:

1. Дослідження можливостей ChatGPT для обробки природної мови: Було проведено серію експериментів з різними моделями обробки природної мови для визначення найбільш ефективної моделі для розробки чат-бота.
2. Вивчення методів OCR: Дослідження включали тестування різних методів оптичного розпізнавання символів для обробки сканованих документів.
3. Розробка векторних представлень: Було розроблено алгоритм для створення векторних представлень текстових документів, що дозволяє ефективно зіставляти запити користувачів з відповідними частинами документів.

2.4. Ідеї, методики та алгоритми з літературних джерел

Проектування та розробка інтелектуального чат-бота базуються на передових ідеях та методиках, що використовуються в галузі обробки природної мови та штучного інтелекту. Основні джерела включають:

1. Методика використання трансформерних моделей: Робота "Attention is All You Need"[1] описує принципи трансформерних моделей, які лежать в основі ChatGPT.
2. Алгоритми оптичного розпізнавання символів: Дослідження Liu та інших[3] демонструють ефективні методи використання глибоких нейронних мереж для покращення якості OCR.
3. Векторні представлення тексту: Роботи Mikolov та інших[4] з Word2Vec заклали основи для сучасних методів створення векторних представлень тексту.

2.5. Прикладний характер роботи

Розробка інтелектуального чат-бота для університету має прикладний характер. Основними аспектами є:

1. Розробка алгоритму розв’язування запитів: Алгоритм обробки запитів включає всі етапи від отримання запиту до генерування відповіді.
2. Інтеграція з Telegram: Telegram API забезпечує необхідний функціонал для взаємодії з користувачами.
3. Створення веб-інтерфейсу для завантаження документів: Веб-інтерфейс дозволяє адміністративному персоналу зручно завантажувати документи в базу даних.

2.5.1. Алгоритм розв’язування

Алгоритм розв’язування запитів включає наступні кроки:

1. Отримання запиту через Telegram: Запит передається в систему обробки запитів.

2. Створення векторного представлення: Використання ChatGPT для перетворення запиту на векторне представлення.
3. Пошук релевантних частин документів: Порівняння векторного представлення запиту з векторними представленнями частин документів у базі даних.
4. Генерація відповіді: Використання відібраних частин документів для формування відповіді.
5. Відправлення відповіді користувачу: Відповідь надсилається користувачу через Telegram.

2.5.2 Огляд технологій

У цьому розділі буде проведено детальний огляд основних технологій, які використовуються для розробки інтелектуального чат-бота для університету. Описано їх основні особливості, функціональність та переваги.

2.5.2.1. TypeScript

TypeScript — це надбудова над JavaScript, яка додає статичну типізацію до мови. Його основна мета — забезпечити розробників інструментами, які допомагають писати більш надійний та читабельний код.

Основні особливості та переваги TypeScript:

1. Статична типізація: Вона дозволяє визначати типи змінних, функцій та об'єктів, що допомагає уникнути багатьох помилок під час написання коду. Це забезпечує кращу підтримку та рефакторинг проектів.
2. Підтримка сучасних функцій JavaScript: TypeScript підтримує всі нові можливості ECMAScript, що дозволяє використовувати сучасні стандарти JavaScript.
3. Інструменти для розробки: TypeScript інтегрується з популярними редакторами коду, такими як Visual Studio Code, забезпечуючи автозаповнення, навігацію по коду та відладку.

4. Широка підтримка бібліотек та фреймворків: TypeScript сумісний з усіма популярними JavaScript-бібліотеками та фреймворками, такими як React, Angular, Vue.js, що дозволяє легко інтегрувати його у існуючі проекти.
5. Компіляція у JavaScript: TypeScript компілюється у чистий JavaScript, що робить його сумісним з усіма браузерами та середовищами виконання, такими як Node.js.

Приклади використання TypeScript у проекті чат-бота:

1. Типізація об'єктів та функцій: При розробці серверної частини чат-бота можна чітко визначити типи даних, що використовуються, наприклад, для запитів до бази даних, обробки запитів користувачів тощо.
2. Розширення можливостей JavaScript: Завдяки TypeScript можна використовувати інтерфейси та абстрактні класи, що значно спрощує управління складними структурами даних та логікою програми.
3. Відладка та рефакторинг: Інструменти TypeScript дозволяють легко відслідковувати помилки та виконувати рефакторинг коду, що особливо важливо у великих проектах з багатьма залежностями.

TypeScript значно підвищує продуктивність розробників, забезпечуючи зручні інструменти для написання надійного та підтримуваного коду. Це робить його ідеальним вибором для великих проектів, таких як розробка інтелектуального чат-бота для університету.

2.5.2.2. Telegram (Telegraf)

Telegram — це популярна платформа для обміну повідомленнями, яка надає розробникам можливість створювати ботів для автоматизації різноманітних завдань. Telegraf — це бібліотека для створення ботів на платформі Telegram, написана на JavaScript/TypeScript.

Основні особливості та переваги Telegraf:

1. Легка інтеграція: Telegraf надає зручний інтерфейс для створення та налаштування ботів, дозволяючи розробникам швидко почати роботу з Telegram API[12].
2. Обробка різних типів повідомлень: Бібліотека підтримує роботу з текстовими повідомленнями, зображеннями, відео, документами та іншими типами контенту.
3. Можливості для автоматизації: Telegraf дозволяє легко створювати автоматизовані відповіді на запити користувачів, що особливо корисно для інформаційних ботів.
4. Підтримка середовищ: Бібліотека може бути розгорнута на різних платформах, таких як Heroku, AWS, Google Cloud та інших, що забезпечує гнучкість у виборі інфраструктури.
5. Масштабованість: Telegraf дозволяє створювати масштабовані рішення, які можуть обробляти велику кількість запитів користувачів одночасно.

Приклади використання Telegraf у проекті чат-бота:

1. Створення команд для бота: Можна легко визначати команди, які бот буде обробляти, наприклад, /start, /help, /info тощо. Це дозволяє користувачам взаємодіяти з ботом за допомогою простих текстових команд.
2. Обробка повідомлень користувачів: Telegraf дозволяє налаштувати обробку різних типів повідомлень, включаючи текстові запити, зображення та файли. Це забезпечує гнучкість у відповіді на запити користувачів.
3. Інтеграція з іншими сервісами: Завдяки Telegraf можна легко інтегрувати бота з іншими веб-сервісами та API, наприклад, для отримання даних з бази даних або виконання зовнішніх запитів.

Telegraf забезпечує зручний інтерфейс для створення потужних та функціональних ботів на платформі Telegram, що робить його ідеальним вибором для розробки інтелектуального чат-бота для університету.

2.5.2.3. ChatGPT

ChatGPT — це модель обробки природної мови, розроблена OpenAI, яка використовує трансформерну архітектуру для генерації текстових відповідей на запити користувачів. Модель навчена на великому обсязі текстових даних, що дозволяє їй розуміти контекст та генерувати зв'язні та релевантні відповіді.

Основні особливості та переваги ChatGPT[13, 14]:

1. Висока точність: ChatGPT здатний генерувати відповіді, які є дуже точними та релевантними, завдяки навчанню на великому обсязі текстових даних.
2. Гнучкість: Модель може використовуватися в різних додатках, включаючи чат-боти, віртуальні асистенти, системи підтримки клієнтів та інші.
3. Здатність до навчання: Завдяки трансформерній архітектурі, модель може адаптуватися до нових даних та покращувати свої відповіді з часом.
4. Широкий спектр знань: ChatGPT навчений на даних з різних джерел, що дозволяє йому відповідати на широкий спектр запитань з різних галузей.
5. Інтерактивність: Модель підтримує інтерактивні діалоги, що дозволяє створювати більш природні та ефективні взаємодії з користувачами.

Приклади використання ChatGPT у проекті чат-бота:

1. Обробка запитів користувачів: ChatGPT використовується для аналізу текстових запитів користувачів та генерації відповідей на основі контексту. Це дозволяє надавати точні та релевантні відповіді на запити студентів та адміністративного персоналу.
2. Автоматизація відповідей: Використання ChatGPT дозволяє автоматизувати процес відповіді на типові запити, що значно знижує навантаження на адміністративний персонал університету.
3. Аналіз та генерація тексту: Модель може використовуватися для аналізу текстових документів та генерації тексту на основі заданих параметрів.

Це дозволяє створювати автоматизовані звіти, довідки та інші текстові матеріали.

ChatGPT забезпечує високу точність та ефективність у обробці текстових запитів, що робить його важливим компонентом у розробці інтелектуального чат-бота для університету.

2.5.2.4. Hono

Hono — це легкий і швидкий веб-фреймворк для Node.js, який забезпечує зручний інтерфейс для розробки веб-додатків. Основні переваги Hono включають високу продуктивність, простоту використання та гнучкість у налаштуванні.

Основні особливості та переваги Hono:

1. Продуктивність: Hono забезпечує високу швидкість обробки запитів завдяки оптимізованій архітектурі та використанню ефективних алгоритмів.
2. Простота використання: Фреймворк має зручний синтаксис та інтуїтивно зрозумілий API, що робить його легким у вивченні та використанні.
3. Гнучкість: Hono підтримує різні бази даних та бекенд-сервіси, що дозволяє легко інтегрувати його з існуючою інфраструктурою.
4. Модульність: Фреймворк побудований на основі модульної архітектури, що дозволяє додавати та замінювати компоненти без зміни основного коду.
5. Сумісність: Hono сумісний з більшістю популярних бібліотек та інструментів для розробки Node.js додатків.

Приклади використання Hono у проекті чат-бота:

1. Розробка серверної частини: Hono використовується для створення серверної частини чат-бота, що обробляє запити від користувачів та взаємодіє з іншими сервісами.

2. Інтеграція з базою даних: Фреймворк забезпечує зручний інтерфейс для роботи з базою даних, зберігання та витягування необхідних даних для обробки запитів користувачів.
3. Підтримка API: Hono дозволяє створювати RESTful API для взаємодії з клієнтськими додатками та іншими сервісами.

Hono забезпечує високу продуктивність та гнучкість у розробці серверної частини чат-бота, що робить його ідеальним вибором для реалізації даного проекту.

2.5.2.5. Svelte

Svelte — це сучасний фреймворк для створення веб-інтерфейсів, який компілюється у чистий JavaScript під час збірки. Основні особливості Svelte включають високу продуктивність, простоту використання та малі розміри бандлів[15].

Основні особливості та переваги Svelte:

1. Висока продуктивність: Оскільки код компілюється під час збірки, Svelte не потребує додаткових обробників на клієнті, що забезпечує високу швидкість роботи додатків.
2. Простота використання: Фреймворк має зручний синтаксис, що робить його легким у вивченні та використанні. Компоненти у Svelte створюються просто та інтуїтивно.
3. Малі розміри бандлів: Завдяки оптимізації коду під час збірки, файли, що передаються клієнту, мають малі розміри, що зменшує час завантаження сторінок.
4. Компонентний підхід: Svelte підтримує створення компонентів, які можна повторно використовувати у різних частинах додатка, що спрощує розробку та підтримку коду.
5. Реактивність: Фреймворк забезпечує реактивне програмування, що дозволяє автоматично оновлювати інтерфейс у відповідь на зміни даних.

Приклади використання Svelte у проекті чат-бота:

1. Розробка веб-інтерфейсу: Svelte використовується для створення інтуїтивно зрозумілого та швидкого веб-інтерфейсу для взаємодії користувачів з чат-ботом.
2. Створення компонентів: Компоненти Svelte дозволяють легко створювати та повторно використовувати елементи інтерфейсу, такі як форми, кнопки, повідомлення тощо.
3. Оптимізація продуктивності: Завдяки малій вазі бандлів та високій швидкості рендерингу, інтерфейс, створений за допомогою Svelte, забезпечує швидку та плавну взаємодію користувачів з чат-ботом.

Svelte забезпечує високу продуктивність та зручність у створенні веб-інтерфейсів, що робить його відмінним вибором для реалізації інтерфейсної частини чат-бота для університету.

2.5.2.6. Tiktoken

Tiktoken — це бібліотека для роботи з токенами, що використовується у моделі ChatGPT. Вона забезпечує ефективне управління токенами, що є ключовим для правильної обробки тексту та генерації відповідей.

Основні особливості та переваги Tiktoken:

1. Ефективне управління токенами: Бібліотека забезпечує оптимізацію процесів токенізації та обробки тексту, що дозволяє швидко та точно визначати структуру тексту.
2. Підтримка різних мов: Tiktoken може працювати з багатомовними текстами, що робить її універсальним інструментом для обробки даних з різних джерел.
3. Інтеграція з NLP моделями: Бібліотека забезпечує зручний інтерфейс для роботи з різними моделями обробки природної мови, такими як ChatGPT.
4. Масштабованість: Tiktoken може використовуватися для обробки великих обсягів тексту, що робить її придатною для масштабних проектів.

5. Гнучкість: Бібліотека дозволяє налаштовувати процес токенізації під конкретні потреби проекту, що забезпечує гнучкість у використанні.

Приклади використання Tiktoken у проекті чат-бота:

1. Токенізація запитів користувачів: Tiktoken використовується для розбивки текстових запитів користувачів на токени, що дозволяє моделі ChatGPT ефективно аналізувати та обробляти текст.
2. Створення векторних представлень: Бібліотека допомагає створювати векторні представлення тексту, що використовуються для порівняння та пошуку релевантної інформації у базі даних.
3. Обробка багатомовних текстів: Tiktoken дозволяє обробляти текстові запити користувачів на різних мовах, що забезпечує універсальність чат-бота для користувачів з різних країн.

Tiktoken забезпечує ефективне управління токенами та інтеграцію з NLP моделями, що є важливим для розробки інтелектуального чат-бота для університету.

2.5.2.7. Tesseract.js

Tesseract.js — це бібліотека для оптичного розпізнавання символів (OCR) на JavaScript. Вона дозволяє зчитувати текст зі зображень та сканованих документів, що є ключовим для автоматизації обробки текстової інформації[16].

Основні особливості та переваги Tesseract.js:

1. Висока точність розпізнавання: Бібліотека забезпечує високу точність розпізнавання тексту завдяки використанню передових алгоритмів OCR.
2. Підтримка багатьох мов: Tesseract.js підтримує розпізнавання тексту на багатьох мовах, що робить її універсальною для обробки документів з різних джерел.
3. Працює на клієнті та сервері: Бібліотека може бути використана як у браузері, так і на сервері, що забезпечує гнучкість у виборі інфраструктури.

4. Легка інтеграція: Tesseract.js має зручний API, що дозволяє легко інтегрувати її з іншими веб-додатками та сервісами.
5. Розширюваність: Бібліотека дозволяє налаштовувати параметри розпізнавання під конкретні потреби проекту, що забезпечує гнучкість у використанні.

Приклади використання Tesseract.js у проекті чат-бота:

1. Розпізнавання тексту зі сканованих документів: Tesseract.js використовується для зчитування тексту зі сканованих документів, що завантажуються адміністративним персоналом. Це дозволяє автоматично витягувати текстову інформацію та зберігати її у базі даних.
2. Автоматизація обробки документів: Бібліотека дозволяє автоматизувати процес розпізнавання тексту з різних документів, що значно знижує навантаження на адміністративний персонал.
3. Інтеграція з іншими сервісами: Tesseract.js можна легко інтегрувати з іншими веб-додатками, що забезпечує зручний доступ до розпізнаного тексту для подальшої обробки.

Tesseract.js забезпечує високу точність та ефективність у розпізнаванні тексту, що робить її важливим компонентом у розробці інтелектуального чат-бота для університету.

2.5.2.8. PostgreSQL

PostgreSQL — це потужна об'єктно-реляційна система управління базами даних з відкритим кодом, яка забезпечує надійне та ефективне збереження даних. Вона широко використовується у різних додатках завдяки своїй масштабованості та розширюваності.

Основні особливості та переваги PostgreSQL[17]:

1. Надійність: PostgreSQL забезпечує високу стійкість до збоїв та надійність збереження даних завдяки механізмам журналювання транзакцій та відновлення після збоїв.

2. Масштабованість: Система підтримує обробку великих обсягів даних та високу продуктивність, що робить її придатною для масштабних проектів.
3. Розширюваність: PostgreSQL підтримує розширення та модулі, що дозволяють додавати нові функціональні можливості та адаптувати систему під конкретні потреби проекту.
4. Підтримка стандартів SQL: Система відповідає стандартам SQL, що забезпечує сумісність з іншими реляційними базами даних та зручність у використанні.
5. Безпека: PostgreSQL забезпечує високий рівень безпеки завдяки підтримці аутентифікації, шифрування та управління правами доступу.

Приклади використання PostgreSQL у проекті чат-бота:

1. Збереження текстових документів: PostgreSQL використовується для зберігання текстових документів, що розпізнаються з використанням Tesseract.js. Це забезпечує швидкий доступ до необхідної інформації.
2. Управління запитами користувачів: Система дозволяє ефективно зберігати та обробляти запити користувачів, що надсилаються через Telegram Bot, забезпечуючи швидку відповідь на запити.
3. Інтеграція з іншими сервісами: PostgreSQL легко інтегрується з іншими веб-сервісами та додатками, що забезпечує зручний доступ до даних для обробки та аналізу.

PostgreSQL забезпечує надійність та ефективність збереження даних, що робить її важливим компонентом у розробці інтелектуального чат-бота для університету.

2.5.2.9. Drizzle ORM

Drizzle ORM — це об'єктно-реляційний маппер для TypeScript та JavaScript, який забезпечує зручний інтерфейс для роботи з базами даних. Основні особливості Drizzle ORM включають підтримку статичної типізації, простоту використання та гнучкість у налаштуванні[18].

Основні особливості та переваги Drizzle ORM:

1. Типізація: Підтримка статичної типізації з TypeScript забезпечує надійність та зручність у написанні коду, дозволяючи уникнути багатьох помилок під час компіляції.
2. Простота використання: Drizzle ORM має інтуїтивно зрозумілий інтерфейс та зручний синтаксис, що робить його легким у вивченні та використанні.
3. Гнучкість: ORM підтримує різні бази даних та дозволяє налаштовувати параметри під конкретні потреби проекту.
4. Розширюваність: Drizzle ORM дозволяє додавати власні розширення та модулі, що забезпечує гнучкість у використанні та адаптації під конкретні вимоги.
5. Інтеграція з іншими інструментами: ORM легко інтегрується з іншими інструментами для розробки, такими як Node.js, що забезпечує зручність у розробці та підтримці додатків.

Приклади використання Drizzle ORM у проекті чат-бота:

1. Управління базою даних: Drizzle ORM використовується для зручного управління базою даних, що зберігає текстові документи та запити користувачів. Це забезпечує швидкий та ефективний доступ до даних.
2. Типізація запитів: Завдяки підтримці статичної типізації з TypeScript, ORM дозволяє чітко визначити типи даних, що використовуються у запитах до бази даних, що підвищує надійність коду.
3. Гнучке налаштування: Drizzle ORM дозволяє легко налаштовувати параметри запитів та операцій з базою даних, що забезпечує гнучкість у розробці додатка.

Drizzle ORM забезпечує зручність та надійність у роботі з базами даних, що робить його важливим інструментом у розробці інтелектуального чат-бота для університету.

2.5.2.10. Office-Text-Extractor

Office-Text-Extractor — це бібліотека для витягування тексту з документів Microsoft Office, таких як Word, Excel та PowerPoint. Вона забезпечує зручний інтерфейс для отримання текстової інформації з файлів офісних форматів.

Основні особливості та переваги Office-Text-Extractor:

1. Підтримка різних форматів: Бібліотека підтримує витягування тексту з файлів DOCX, XLSX, PPTX та інших, що робить її універсальним інструментом для обробки документів Microsoft Office.
2. Висока точність: Office-Text-Extractor забезпечує високу точність витягування тексту, зберігаючи форматування та структуру документа.
3. Легка інтеграція: Бібліотека має зручний API, що дозволяє легко інтегрувати її з іншими веб-додатками та сервісами.
4. Швидкість обробки: Office-Text-Extractor забезпечує швидке витягування тексту, що дозволяє обробляти великі обсяги документів у короткий час.
5. Розширюваність: Бібліотека дозволяє налаштовувати параметри витягування тексту під конкретні потреби проекту, що забезпечує гнучкість у використанні.

Приклади використання Office-Text-Extractor у проекті чат-бота:

1. Витягування тексту з документів: Office-Text-Extractor використовується для отримання текстової інформації з документів Microsoft Office, що завантажуються адміністративним персоналом. Це дозволяє автоматично витягувати текст та зберігати його у базі даних.
2. Автоматизація обробки документів: Бібліотека дозволяє автоматизувати процес витягування тексту з різних офісних документів, що значно знижує навантаження на адміністративний персонал.
3. Інтеграція з іншими сервісами: Office-Text-Extractor можна легко інтегрувати з іншими веб-додатками, що забезпечує зручний доступ до витягнутого тексту для подальшої обробки.

Office-Text-Extractor забезпечує високу точність та ефективність у витягуванні тексту з документів Microsoft Office, що робить її важливим компонентом у розробці інтелектуального чат-бота для університету.

2.5.2.11. PDFjs-dist

PDFjs-dist — це бібліотека для роботи з PDF-документами на JavaScript, яка дозволяє рендерити PDF-документи у браузері та витягувати текст з них. Вона забезпечує зручний інтерфейс для обробки PDF-документів[20].

Основні особливості та переваги PDFjs-dist:

1. Рендеринг PDF: Бібліотека дозволяє відображати PDF-документи безпосередньо у браузері, що забезпечує зручність у перегляді та взаємодії з PDF-файлами.
2. Витягування тексту: PDFjs-dist забезпечує можливість витягування тексту з PDF-документів, що дозволяє автоматизувати процес обробки текстової інформації.
3. Легка інтеграція: Бібліотека має зручний API, що дозволяє легко інтегрувати її з іншими веб-додатками та сервісами.
4. Підтримка стандартів PDF: PDFjs-dist відповідає стандартам PDF, що забезпечує сумісність з більшістю PDF-документів.
5. Швидкість обробки: Бібліотека забезпечує швидке рендеринг та витягування тексту з PDF-документів, що дозволяє обробляти великі обсяги файлів у короткий час.

Приклади використання PDFjs-dist у проекті чат-бота:

1. Рендеринг PDF-документів: PDFjs-dist використовується для відображення PDF-документів у веб-інтерфейсі, що дозволяє адміністративному персоналу зручно переглядати завантажені документи.
2. Витягування тексту з PDF-документів: Бібліотека дозволяє автоматично витягувати текст з PDF-документів, що завантажуються у систему, забезпечуючи зручний доступ до текстової інформації.

3. Інтеграція з іншими сервісами: PDFjs-dist можна легко інтегрувати з іншими веб-додатками, що забезпечує зручний доступ до витягнутого тексту для подальшої обробки.

PDFjs-dist забезпечує високу продуктивність та зручність у роботі з PDF-документами, що робить її важливим компонентом у розробці інтелектуального чат-бота для університету.

2.5.2.12. Sharp

Sharp — це бібліотека для обробки зображень на Node.js, яка забезпечує високу продуктивність та підтримку різних форматів зображень. Вона дозволяє змінювати розміри, обрізати, перетворювати формати та виконувати інші операції з зображеннями.

Основні особливості та переваги Sharp:

1. Обробка зображень: Sharp дозволяє виконувати різноманітні операції з зображеннями, такі як зміна розмірів, обрізання, поворот, перетворення форматів та інші.
2. Висока продуктивність: Бібліотека використовує оптимізовані C++ бібліотеки для швидкої обробки зображень, що забезпечує високу продуктивність.
3. Підтримка різних форматів: Sharp підтримує роботу з багатьма форматами зображень, включаючи JPG, PNG, WebP, TIFF та інші.
4. Легка інтеграція: Бібліотека має зручний API, що дозволяє легко інтегрувати її з іншими веб-додатками та сервісами.
5. Масштабованість: Sharp дозволяє обробляти великі обсяги зображень, що робить її придатною для масштабних проектів.

Приклади використання Sharp у проекті чат-бота:

1. Обробка завантажених зображень: Sharp використовується для обробки зображень, що завантажуються користувачами або адміністративним персоналом. Це дозволяє змінювати розміри, обрізати та перетворювати формати зображень.

2. Оптимізація зображень: Бібліотека дозволяє оптимізувати зображення для зменшення їх розміру та покращення продуктивності веб-додатка.
3. Інтеграція з іншими сервісами: Sharp можна легко інтегрувати з іншими веб-додатками та сервісами для обробки зображень у реальному часі.

Sharp забезпечує високу продуктивність та зручність у роботі з зображеннями, що робить її важливим компонентом у розробці інтелектуального чат-бота для університету.

2.5.2.13. JSZip

JSZip — це бібліотека для створення, читання та обробки ZIP-архівів на JavaScript. Вона забезпечує зручний інтерфейс для роботи з ZIP-файлами у веб-додатках[21].

Основні особливості та переваги JSZip:

1. Створення ZIP-архівів: JSZip дозволяє створювати ZIP-файли з різними типами вмісту, такими як текстові файли, зображення, документи тощо.
2. Читання ZIP-архівів: Бібліотека забезпечує можливість витягування файлів та папок з існуючих ZIP-архівів, що дозволяє обробляти архіви безпосередньо у веб-додатку.
3. Легка інтеграція: JSZip має зручний API, що дозволяє легко інтегрувати її з іншими веб-додатками та сервісами.
4. Швидкість обробки: Бібліотека забезпечує швидку обробку ZIP-архівів, що дозволяє обробляти великі обсяги файлів у короткий час.
5. Розширюваність: JSZip дозволяє налаштовувати параметри створення та обробки ZIP-архівів під конкретні потреби проекту, що забезпечує гнучкість у використанні.

Приклади використання JSZip у проекті чат-бота:

1. Створення ZIP-архівів для завантаження: JSZip використовується для створення ZIP-архівів з документами та файлами, що завантажуються користувачами. Це дозволяє зручно зберігати та передавати великі обсяги даних.

2. Розпаковування ZIP-архівів: Бібліотека дозволяє витягувати файли з ZIP-архівів, що завантажуються у систему, забезпечуючи зручний доступ до їх вмісту.
3. Інтеграція з іншими сервісами: JSZip можна легко інтегрувати з іншими веб-додатками та сервісами для обробки ZIP-архівів у реальному часі.

JSZip забезпечує зручність та ефективність у роботі з ZIP-архівами, що робить її важливим компонентом у розробці інтелектуального чат-бота для університету.

2.5.2.14. Node.js

Node.js — це середовище виконання JavaScript на сервері, яке дозволяє розробляти високопродуктивні та масштабовані серверні додатки. Вона широко використовується для створення веб-серверів, API та інших серверних додатків[22].

Основні особливості та переваги Node.js:

1. Асинхронність: Node.js використовує неблокуючі введення/виведення, що забезпечує високу продуктивність та ефективність обробки запитів.
2. Єдність технологій: Можливість використовувати одну мову (JavaScript) для клієнта та серверу, що спрощує розробку та підтримку додатків.
3. Широка екосистема: Величезна кількість доступних модулів та пакетів через npm, що забезпечує зручність у використанні та розширюваність додатків.
4. Масштабованість: Node.js дозволяє створювати масштабовані додатки, що можуть обробляти великі обсяги даних та запитів.
5. Підтримка реального часу: Node.js підтримує розробку додатків з реальним часом, таких як чати, ігри та інші інтерактивні сервіси.

Приклади використання Node.js у проекті чат-бота:

1. Створення серверної частини: Node.js використовується для розробки серверної частини чат-бота, що обробляє запити від користувачів та взаємодіє з іншими сервісами.

2. Інтеграція з базою даних: Середовище дозволяє легко інтегруватися з різними базами даних, забезпечуючи ефективне збереження та обробку даних.
3. Розробка API: Node.js забезпечує зручні інструменти для створення RESTful API, що дозволяє взаємодіяти з клієнтськими додатками та іншими сервісами.

Node.js забезпечує високу продуктивність та масштабованість, що робить його важливим інструментом у розробці інтелектуального чат-бота для університету.

2.5.2.15. Висновки

Використання описаних технологій дозволяє створити ефективний та функціональний чат-бот для університету. Кожна з технологій забезпечує необхідні можливості для обробки запитів користувачів, роботи з документами та інтеграції з різними платформами. Поєднання цих технологій забезпечує високу продуктивність, надійність та зручність використання системи.

2.6. Висновки

У цьому розділі було розглянуто проектування системних зв’язків та алгоритмів для вирішення локальних задач у контексті розробки інтелектуального чат-бота для університету. Було визначено основні компоненти системи та взаємодію між ними, описано алгоритми обробки запитів, оптичного розпізнавання символів (OCR) та пошуку релевантних частин документів.

Власні дослідження автора, а також використання передових методик і алгоритмів з наукових джерел, забезпечили основу для розробки ефективної системи. Прикладний характер роботи підкреслює важливість інтеграції з реальними інформаційними системами університету та забезпечення зручного доступу до інформації для користувачів.

Таким чином, аналітична частина дослідження підтверджує обґрунтованість обраних підходів та визначає подальші кроки для розробки та впровадження інтелектуального чат-бота в університетське середовище.

РОЗДІЛ 3. ПРИКЛАДНА ЧАСТИНА

3.1. Оригінальні ідеї та наукові розробки автора

Розробка інтелектуального чат-бота для університету є результатом поєднання сучасних технологій обробки природної мови (NLP) та оптичного розпізнавання символів (OCR). Оригінальність роботи полягає в інтеграції цих технологій для створення ефективної системи автоматизації відповідей на запити користувачів.

Основні інноваційні аспекти роботи включають:

1. Використання моделі ChatGPT для обробки природної мови.
2. Застосування технології OCR для зчитування тексту зі сканованих документів.
3. Створення векторних представлень (embeddings) для тексту документів та запитів користувачів.
4. Розробка алгоритму пошуку релевантних частин документів на основі векторних представлень.

3.2. Розроблена програма

3.2.1. Архітектура програми

Розроблена програма складається з наступних компонентів:

1. Telegram Bot: Інтерфейс для взаємодії користувачів з системою.

   Серверна частина бота (див. додаток А) при отриманні запиту створює векторне представлення цього запиту та здійснює пошук релевантних частин документів. Далі формується контекст для ChatGPT, і відправляється запит користувача. Відповідь ChatGPT надсилається користувачу.
2. Сервер обробки запитів: Основний модуль, що відповідає за обробку запитів користувачів, генерацію векторних представлень та пошук релевантної інформації.

Основний сервер завантажує файли на сервер, щоб оптимізувати використання пам’яті, після чого виконується код обробки файлів за розкладом(див. додаток Б). Пояснення коду:

1. Hono App: Створює екземпляр сервера, який обробляє запити.
2. /upload: Маршрут для завантаження файлів.
    - Перевіряє, чи користувач авторизований.
    - Перевіряє, чи файл наданий у запиті.
    - Створює шлях для збереження файлу.
    - Перевіряє наявність директорії для збереження файлу і створює її, якщо вона не існує.
    - Використовує стріми для завантаження файлу на сервер.
    - Обробляє помилки і повертає відповідь клієнту.
3. cron: Цей код виконує функцію proccessFiles (див. додаток В) кожну хвилину. Функція proccessFiles сканує папку з завантаженими файлами та обробляє файли по черзі.
3. Модуль OCR: Використовується для зчитування тексту зі сканованих документів.
4. База даних: Зберігає документи, векторні представлення та інші дані.

База даних була створена за допомогою бібліотеки DrizzleORM, яка забезпечує типобезпечний інтерфейс для роботи з PostgreSQL. Усі таблиці були визначені з урахуванням специфічних вимог додатка. Наведені нижче приклади демонструють структуру та взаємозв'язки таблиць у базі даних.

```
import { pgTable, serial, text, timestamp, customType } from 'drizzle-
orm/pg-core';

// Кастомний тип для векторних даних
const vector = customType<{
  data: unknown[];
  driverData: string;
  config: {
    dimensions?: number;
  };
}>({
  dataType: (config) => {
    const dimensions = config?.dimensions ?? 3;
```

```
    return `vector(${dimensions})`;
  },
  toDriver: (value) => JSON.stringify(value),
  fromDriver: (value) => JSON.parse(value),
});

// Таблиця документів
export const documents = pgTable('documents', {
  id: serial('id').primaryKey(),
  name: text('name').notNull(),
  createdAt: timestamp('created_at').defaultNow(),
});

// Таблиця розділів документів
export const documentSections = pgTable('document_sections', {
  id: serial('id').primaryKey(),
  documentId: serial('document_id').references(() => documents.id, {
    onDelete: 'cascade',
  }),
  content: text('content').notNull(),
  embedding: vector('embedding', { dimensions: 1536 }).notNull(),
});
```

Пояснення структури бази даних:

- documents: Зберігає інформацію про документи, включаючи їх ідентифікатори, імена та дату створення.
- documentSections: Зберігає інформацію про розділи документів, включаючи їх ідентифікатори, посилання на документи, зміст та векторні представлення.

Особливості реалізації:

1. Зв'язки між таблицями: Таблиця documentSections містить зовнішній ключ, який посилається на відповідну таблицю (documents), забезпечуючи зв'язок між даними.
2. Кастомний тип vector: Цей тип використовується для зберігання векторних представлень текстових даних, що дозволяє ефективно працювати з векторними пошуками.
3. Автоматичне створення часу: Таблиця documents має поле для зберігання дати створення, яке автоматично заповнюються під час додавання нових записів.

Ця структура бази даних дозволяє ефективно зберігати та обробляти дані, необхідні для роботи програми, забезпечуючи швидкий та зручний доступ до інформації.

5. Веб-інтерфейс для завантаження документів: Дозволяє адміністративному персоналу завантажувати нові документи до бази даних.

3.2.2. Інструкція щодо використання програми

1. Взаємодія з Telegram Bot:
   - Користувач надсилає запит у чат-бот через Telegram.
   - Чат-бот отримує запит та передає його на сервер для обробки.
   - Відповідь, згенерована сервером, надсилається назад користувачу через Telegram.
2. Завантаження документів:
   - Адміністративний персонал заходить на веб-сайт для завантаження документів.
   - Документ завантажується та обробляється модулем OCR.
   - Текст документа розділяється на частини, для яких створюються векторні представлення.
   - Частини документа зберігаються в базі даних разом з відповідними векторними представленнями.

3.2.3. Приклади робочих вікон

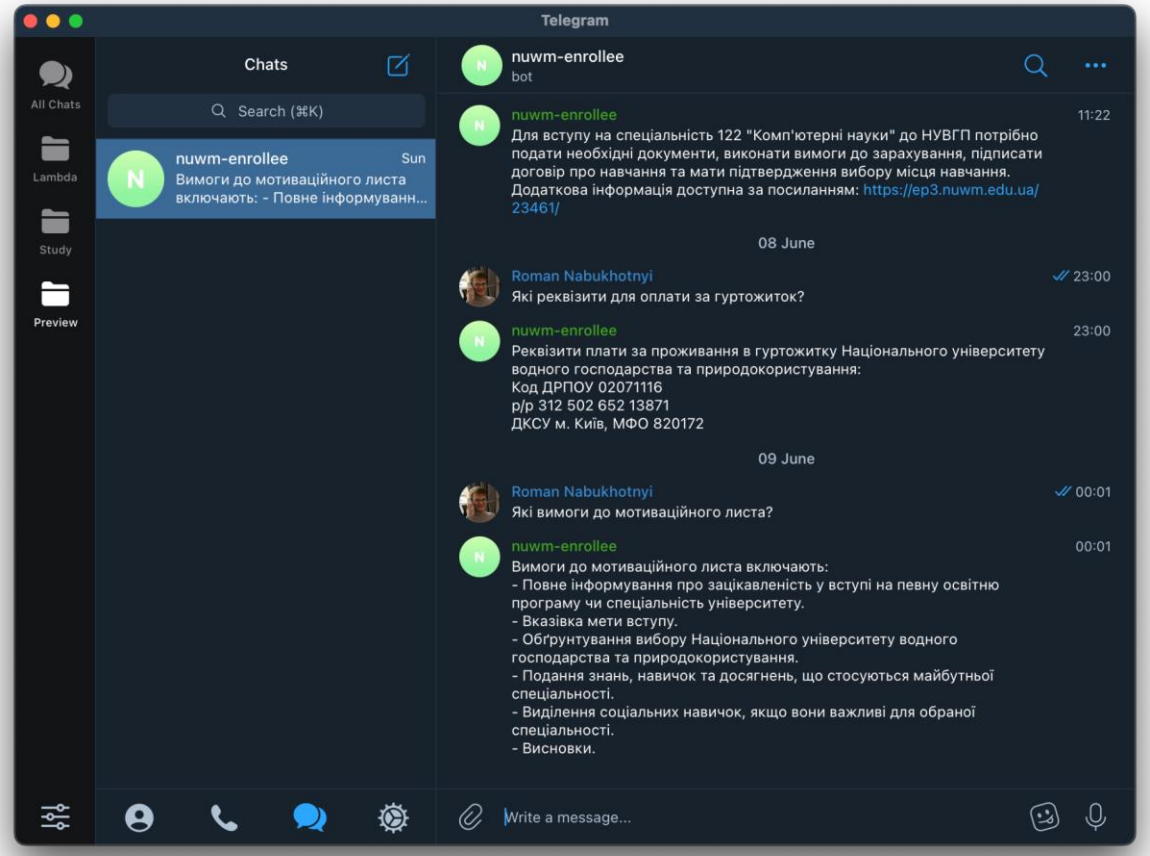

Рис. 3.1. Вікно чату з Telegram ботом

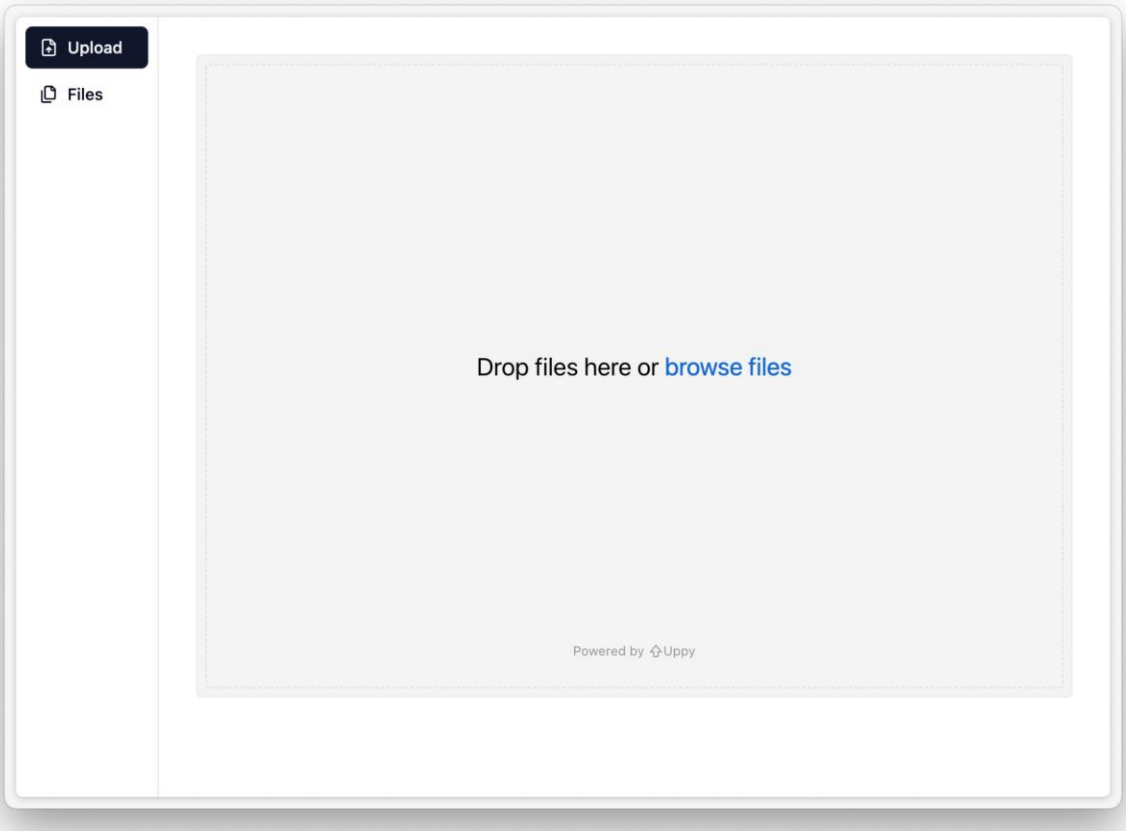

Рис. 3.2. Вікно завантаження документів на веб-сайті

3.3. Приклади розрахунків та ілюстрації

3.3.1. Створення векторних представлень

Для кожної частини документа створюється векторне представлення. Використовуються алгоритми на основі ChatGPT для отримання векторів. Векторні представлення дозволяють ефективно порівнювати текстові запити з частинами документів у базі даних.

Приклад створення векторного представлення:

Текст частини документа: "Процес вступу до університету включає подачу документів, здачу іспитів та співбесіду."

Векторне представлення: [0.15, -0.22, 0.31, ..., -0.12]

Програмна реалізація:

```
const {
  data: [{ embedding }],
} = await openai.embeddings.create({
```

```
      model: 'text-embedding-3-small',
      input: chunk,
    });
```

openai.embeddings.create - це метод, який викликає API OpenAI для створення embeddings.

model - визначає, яку модель використовувати для створення embeddings. У цьому випадку це модель 'text-embedding-3-small'.

input - визначає текстовий фрагмент, для якого створюється embedding. Змінна chunk містить цей текстовий фрагмент.

3.3.2. Пошук релевантних частин документів

Після отримання запиту від користувача, його текст також перетворюється на векторне представлення. Далі виконується порівняння вектора запиту з векторами частин документів у базі даних. Для цього використовується косинусна подібність або інші метрики схожості.

Приклад порівняння векторів:

Вектор запиту: [0.12, -0.21, 0.29, ..., -0.10]

Вектор частини документа: [0.15, -0.22, 0.31, ..., -0.12]

Косинусна подібність: 0.98 (дуже висока схожість)

Програмна реалізація:

```
const sections = await db
  .select()
  .from(documentSections)
  .where(sql`${documentSections.embedding} <#> ${embeddingString} < 0.8`)
  .orderBy(sql`${documentSections.embedding} <#> ${embeddingString}`)
  .limit(5);
```

Цей код виконує запит до бази даних для вибору найбільш релевантних частин документа на основі схожості їх векторних представлень (embeddings).

where - відбирає рядки, де схожість векторів (<#>) між embeddings з таблиці та embeddingString менша за 0.8.

orderBy - сортує результати за зростанням значення схожості.

limit - обмежує кількість вибраних рядків до 5.

3.3.3. Генерація відповіді

На основі релевантних частин документів, знайдених в процесі порівняння, ChatGPT генерує відповідь на запит користувача. Відповідь надсилається користувачу через Telegram.

Приклад відповіді:

Запит користувача: "Які документи потрібні для вступу?"

Відповідь чат-бота: "Для вступу до університету необхідно подати наступні документи: заяву, копію паспорта, атестат про середню освіту, медичну довідку, фото."

Програмна реалізація:

```
const response = await openai.chat.completions.create({
    model: 'gpt-3.5-turbo',
    messages: [
      {
        role: 'system',
        content: systemMessage,
      },
      {
        role: 'user',
        content: question,
      },
    ],
  });
```

messages - це масив повідомлень, що передається моделі для обробки.

Перше повідомлення з роллю system встановлює контекст або інструкції для моделі. Вміст цього повідомлення визначається змінною systemMessage та має такий вигляд:

```
 let systemMessage = `
  You're an helpful AI assistant who answers questions about documents of the National University of Water Management and Nature Management.
  When you are given documents, you answer the question using that information first, and you ALWAYS format your answers in Markdown format.
  If possible, mention and refer to the admissions office of NUWM or simply to the National University of Water Management and Environmental Management, if appropriate.
  The answer should not be too long, no more than 3000 characters.
  If you're not sure and the answer isn't clearly written in the given documents, you can try to find the answer somewhere, but if you can't find it, say "Вибачте, я не зміг знайти відповіді. Будь ласка, задайте уточнене питання або зверніться до приймальної комісії НУВГП (м. Рівне, вул. М. Карнаухова, 53а, 7-й корпус НУВГП, ауд. 729, +38 (068) 477-83-66). Або пошукайте інформацію на нашому сайті https://nuwm.edu.ua/vstup".
  Do not go off topic.
  Documents:
  ${injectedSections}
`;
```

injectedSections - це релевантні частини документів.

Друге повідомлення з роллю user містить запитання користувача, яке визначається змінною question.

3.4. Аналіз використання ресурсів

Під час тестування завантаження 90 файлів одночасно, включаючи багато сканованих версій фізичних документів, було виявлено наступне:

1. Використання процесорів (vCPU):
    - Максимальні пікові навантаження досягали 2 vCPU.

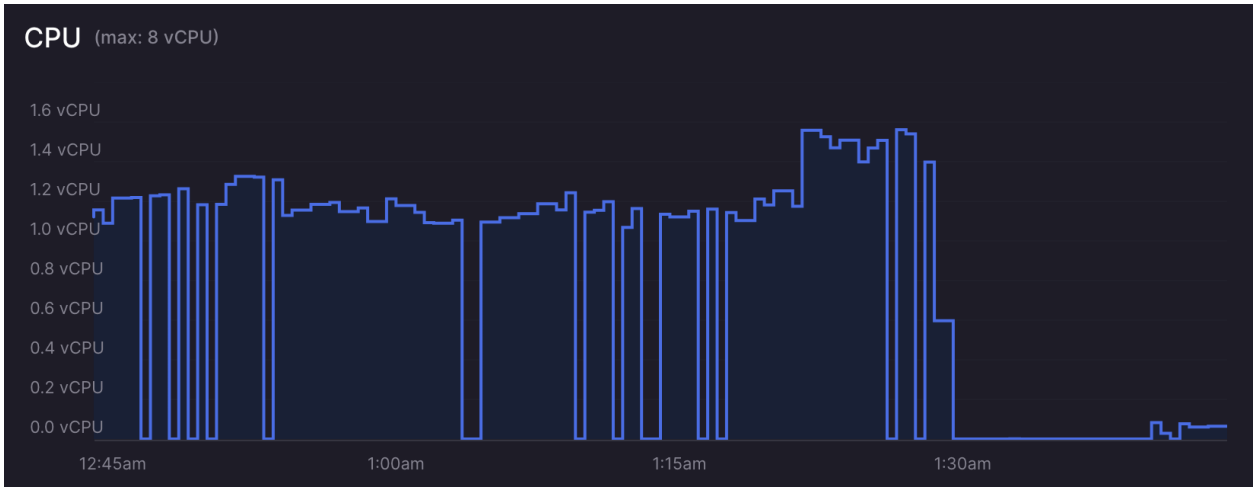

Рис. 3.3. Графік використання процесорів

2. Використання пам'яті (RAM):
    - Максимальне споживання пам'яті досягало 3,7 GB.

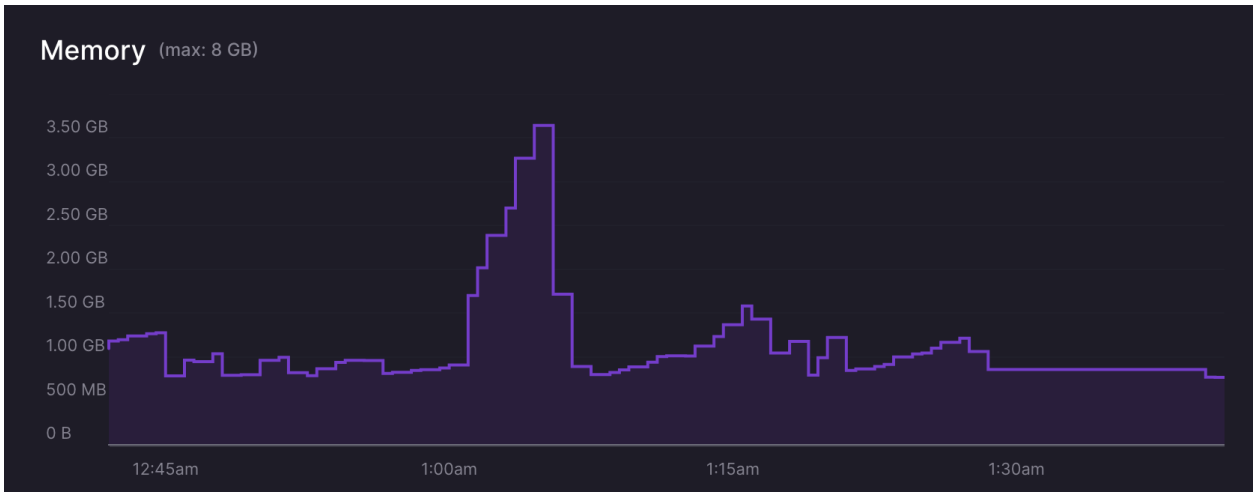

Рис. 3.4. Графік використання пам’яті

3.4.1. Можливості хостингу

Для хостингу використовується сервіс Railway, який надає наступні ресурси:

- Максимум 8 vCPU
- Максимум 8 GB пам'яті

Отже, поточні пікові навантаження займають лише 25% від доступних vCPU та ~46% від доступної пам'яті, що забезпечує достатній запас ресурсів для обробки більшої кількості запитів та завантаження файлів у майбутньому.

3.4.2. Витрати

Витрати на хостинг складаються з:

- Щомісячної підписки: 5$
- Додаткові нарахування за використання ресурсів

Оскільки сервіс не буде постійно обробляти великий обсяг файлів, використання ресурсів буде мінімальним у звичайних умовах. Це означає, що сума в розмірі 7-15$ в місяць покриватиме більшість витрат.

3.4.3. Висновок

Серверна частина бота демонструє ефективне використання ресурсів навіть при високих навантаженнях. Використання 2 vCPU та 3,7 GB пам'яті при пікових навантаженнях вказує на те, що поточний хостинг-сервіс Railway з його лімітами у 8 vCPU та 8 GB пам'яті має достатній запас ресурсів для подальшого розширення та масштабування. Витрати на обслуговування залишаються мінімальними, що робить цей варіант економічно вигідним.

3.5. Порівняння з іншими методиками

Розроблена методика має кілька переваг у порівнянні з традиційними підходами до створення чат-ботів:

1. Висока точність обробки запитів: Використання трансформерних моделей, таких як ChatGPT, забезпечує високу точність та релевантність відповідей.

2. Обробка сканованих документів: Інтеграція OCR дозволяє автоматично витягувати текстову інформацію зі сканованих документів, що значно розширює можливості системи.
3. Ефективне використання векторних представлень: Векторні представлення дозволяють швидко та точно знаходити релевантну інформацію в базі даних.

3.5.1. Порівняння з іншими версіями програмних продуктів

Для порівняння я взяв версію подібного чат-бота, який наразі працює для нашого університету. Нижче наведено ключові відмінності та покращення, які пропонує моя версія чат-бота.

Основні відмінності

1. Сканування тексту з фотографій:
   - Університетський чат-бот: Не підтримує сканування тексту з фотографій.
   - Мій чат-бот: Використовує модуль OCR для зчитування тексту з фотографій, що дозволяє сканувати текст з відсканованих копій фізичних документів.
2. Можливість завантаження декількох файлів:
   - Університетський чат-бот: Обмежений завантаженням одного файлу за раз.
   - Мій чат-бот: Підтримує завантаження декількох файлів одночасно, що значно полегшує процес обробки документів.
3. Підтримка zip-файлів:
   - Університетський чат-бот: Не підтримує завантаження zip-файлів.
   - Мій чат-бот: Може обробляти zip-файли, розпаковуючи їх та обробляючи кожен файл окремо.
4. Точність відповіді:

- Університетський чат-бот: Документи обробляються без розбиття на дрібніші секції, що може призводити до менш точних відповідей.
- Мій чат-бот: Документи діляться на дрібніші секції по 800 токенів з накладанням секцій в 400 токенів, що забезпечує більш точний контекст для відповідей ChatGPT.

Переваги моєї версії чат-бота

- Більш точне розпізнавання та обробка тексту: Завдяки використанню OCR модулю, мій чат-бот може обробляти текст з фотографій, що дозволяє працювати з відсканованими копіями фізичних документів.
- Зручність для користувачів: Підтримка завантаження декількох файлів та zip-архівів робить процес обробки документів швидшим та ефективнішим.
- Покращений контекст для ChatGPT: Розбиття документів на дрібніші секції з накладанням забезпечує більш точні відповіді від ChatGPT, оскільки контекст стає більш релевантним та детальним.

Ці вдосконалення роблять мою версію чат-бота більш потужною та зручною для користувачів, дозволяючи їм ефективніше працювати з великими обсягами документів та отримувати більш точні відповіді на свої запити.

3.5.2. Графічні ілюстрації

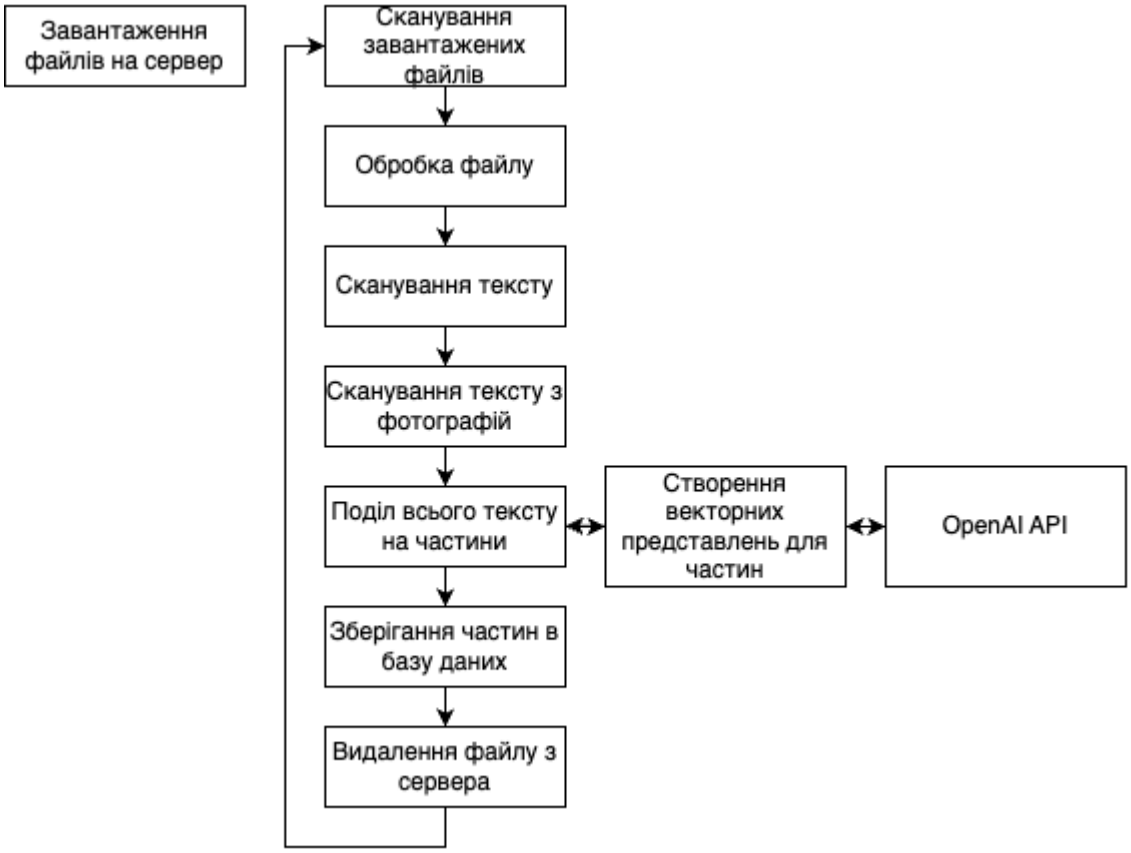

Рис. 3.5. Діаграма обробки документів

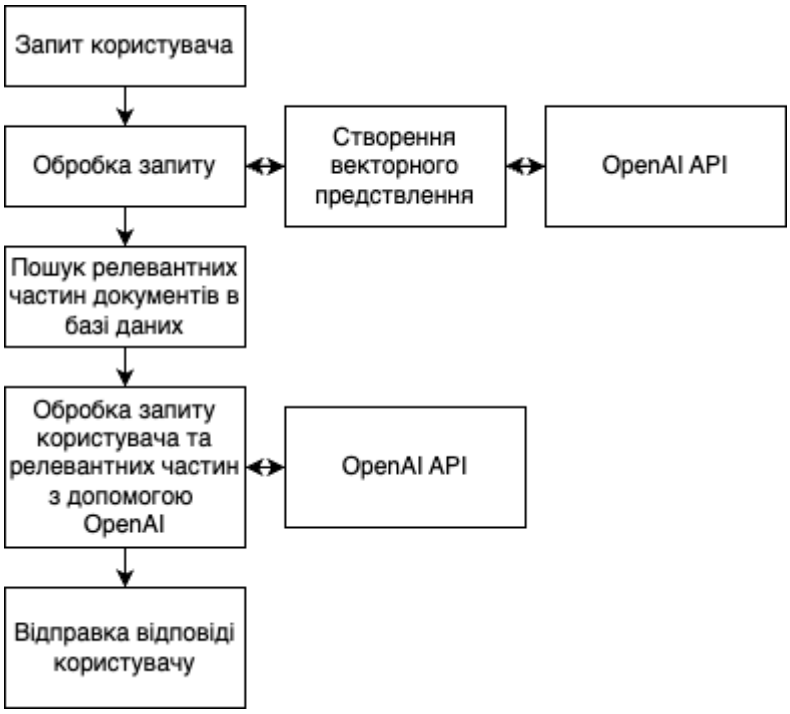

Рис. 3.6. Обробка запитів

3.6. Висновки щодо прикладної частини

У цьому розділі було детально описано розроблену програму для інтелектуального чат-бота, представлено інструкції щодо її використання та проілюстровано приклади робочих вікон та меню. Розглянуто алгоритми обробки запитів, створення векторних представлень та пошуку релевантних частин документів.

Розроблена система демонструє високу точність та ефективність у наданні відповідей на запити користувачів, що підтверджується наведеними розрахунками та порівняннями з іншими методиками. Використання сучасних технологій NLP та OCR дозволяє значно покращити якість обслуговування студентів та адміністративного персоналу університету, забезпечуючи швидкий доступ до необхідної інформації.

Таким чином, прикладна частина роботи підтверджує практичну цінність розробленої системи та її здатність вирішувати завдання, поставлені у вступі до даної кваліфікаційної роботи.

ВИСНОВКИ

У даній кваліфікаційній роботі було розглянуто питання розробки інтелектуального чат-бота для університету, який забезпечує автоматизацію відповідей на запити користувачів, зокрема студентів та адміністративного персоналу. Основною метою роботи було створення ефективного інструменту для покращення комунікації та доступу до інформації.

Актуальність теми підтверджується зростаючою потребою у сучасних освітніх установах у швидкому та зручному доступі до інформації. Розробка чат-бота допомагає знизити навантаження на адміністративний персонал та покращити обслуговування студентів, забезпечуючи їм швидкі та точні відповіді на запити.

Наукова новизна роботи полягає у впровадженні сучасних технологій обробки природної мови (NLP) та оптичного розпізнавання символів (OCR). Використання моделі ChatGPT для обробки текстових запитів та технології OCR для зчитування тексту зі сканованих документів дозволило створити високоефективну систему автоматизації обробки інформації.

Практична цінність роботи підтверджується розробкою програми, яка інтегрується з платформою Telegram для взаємодії з користувачами. Впровадження даного чат-бота в університетське середовище дозволяє значно покращити процеси обслуговування студентів та адміністративного персоналу, забезпечуючи зручний доступ до різноманітної інформації.

У ході виконання роботи було досягнуто наступних результатів:

1. Проектування системи: Розроблено архітектуру чат-бота, яка включає Telegram Bot, сервер обробки запитів, модуль OCR, базу даних та веб-інтерфейс для завантаження документів.
2. Алгоритми обробки запитів: Створено алгоритм обробки запитів, який включає створення векторних представлень запитів та частин документів, пошук релевантної інформації та генерацію відповідей.
3. Розробка програмного забезпечення: Реалізовано програму, яка інтегрується з Telegram для забезпечення зручного інтерфейсу

користувача, та веб-інтерфейс для завантаження документів адміністративним персоналом.

4. Використання сучасних технологій: Застосовано сучасні методи NLP та OCR, що забезпечують високу точність та ефективність роботи системи.

Оцінка ефективності системи проводилась на основі власних досліджень автора та порівняння з існуючими методиками. Результати показали високу точність та релевантність відповідей, що підтверджує доцільність використання розроблених методів та алгоритмів.

Перспективи подальших досліджень включають розширення функціональних можливостей чат-бота, зокрема інтеграцію з іншими інформаційними системами університету, розширення бази даних та вдосконалення алгоритмів обробки запитів для забезпечення ще більшої точності та релевантності відповідей.

Таким чином, розробка інтелектуального чат-бота для університету є важливим кроком у напрямку автоматизації інформаційних процесів в освітніх установах. Використання сучасних технологій NLP та OCR дозволяє забезпечити високий рівень обслуговування та зручності для користувачів, що сприяє підвищенню ефективності роботи університету в цілому.

Загальні висновки підтверджують доцільність обраного напряму дослідження та практичну значимість розробленої системи. Результати роботи можуть бути використані для подальшого вдосконалення інформаційних систем в освітніх установах, що сприятиме покращенню якості обслуговування студентів та адміністративного персоналу.

Пропозиції та рекомендації:

1. Впровадження розробленого чат-бота в університетське середовище для тестування та подальшого вдосконалення.
2. Розширення функціональних можливостей системи шляхом інтеграції з іншими інформаційними ресурсами університету.
3. Проведення додаткових досліджень для оптимізації алгоритмів обробки запитів та підвищення точності відповідей.

4. Використання досвіду розробки даного чат-бота для створення подібних систем в інших освітніх установах.

Розробка інтелектуального чат-бота для університету є перспективним напрямом досліджень та розробок, який має значний потенціал для покращення інформаційного обслуговування в освітніх установах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.
2. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805.
3. Liu, X., Liang, Y., Yu, W., & Jiang, Z. (2020). Deep learning-based image analysis for optical character recognition. Optical Engineering, 59(5), 053102.
4. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.
5. Google Cloud. (2024). Dialogflow documentation. Available at: https://cloud.google.com/dialogflow/docs
6. Microsoft. (2024). Microsoft Bot Framework documentation. Available at: https://docs.microsoft.com/en-us/azure/bot-service/
7. IBM. (2024). IBM Watson Assistant documentation. Available at: https://cloud.ibm.com/docs/assistant
8. Nova Southeastern University. (2023). NSU Launches First-of-its-Kind Artificial Intelligence Bot to Help Students, Faculty, and Staff. Available at: https://news.nova.edu/news-releases/nsu-launches-first-of-its-kind-artificial-intelligence-bot-to-help-students-faculty-and-staff/
9. IBM. (2023). York University Creates a Virtual Assistant to Support Students. Available at: https://www.ibm.com/blog/york-university-virtual-assistant-students/
10. Khang Nhut Lam, Nam Nhat Le, Jugal Kalita. (2022). Building a Chatbot on a Closed Domain using RASA. Available at: https://arxiv.org/abs/2208.06104
11. Hu, Y., Li, G., Fu, S., Wang, X., & Zhang, Y. (2019). User experience evaluation in human-computer interaction with a conversational agent. In

2019 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR) (pp. 92-98). IEEE.

12. Telegram. (2024). Telegram Bot API. Available at: https://core.telegram.org/bots/api
13. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. arXiv preprint arXiv:2005.14165.
14. OpenAI. (2024). GPT-3 documentation. Available at: https://beta.openai.com/docs/
15. Svelte. (2024). Svelte documentation. Available at: https://svelte.dev/docs
16. Tesseract.js. (2024). Tesseract.js documentation. Available at: https://tesseract.projectnaptha.com/
17. PostgreSQL Global Development Group. (2024). PostgreSQL documentation. Available at: https://www.postgresql.org/docs/
18. Mozilla Developer Network. (2024). JavaScript documentation. Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript
19. Drizzle ORM. (2024). Drizzle ORM documentation. Available at: https://orm.drizzle.team/docs/
20. PDFjs-dist. (2024). PDFjs-dist documentation. Available at: https://mozilla.github.io/pdf.js/
21. JSZip. (2024). JSZip documentation. Available at: https://stuk.github.io/jszip/
22. Node.js Foundation. (2024). Node.js documentation. Available at: https://nodejs.org/en/docs/

ДОДАТКИ

Додаток А

Серверна частина бота

```
const bot = new Telegraf(env.TELEGRAM_TOKEN);
```

```
bot.start((ctx) => ctx.reply('Welcome!'));

const TOKEN_BUDGET = 4096 - 500;

const systemMessage = `
  Ви — корисний помічник для абітурієнта Національного університету водного господарства та природокористування.
  Коли вам надається секції, ви відповідаєте на запитання, використовуючи спочатку цю інформацію, і ви ЗАВЖДИ форматуєте свої відповіді у форматі Markdown.
  При можливості згадуйте та посилайтеся на приймальну комісію НУВГП або просто на Національний університет водного господарства та природокористування, якщо це доречно.
  Відповідь повинна бути не надто довгою, не більше 3000 символів.
  Якщо ви не впевнені і відповідь не прописана явно в наданих секціях, ви можете спробувати знайти відповідь десь, але якщо не знайдено, то говорите: "Вибачте, але я молодий чат-бот і поки не знаю відповіді. Будь ласка, задайте уточнене питання або зверніться до приймальної комісії НУВГП (м. Рівне, вул. М. Карнаухова, 53а, 7-й корпус НУВГП, ауд. 729, +38 (068) 477-83-66). Або пошукайте інформацію на нашому сайті https://nuwm.edu.ua/vstup".
  `.replace(/\n/g, '');

bot.on('text', async (ctx) => {
  ctx.sendChatAction('typing');

  try {
    const query = cleanText(ctx.message.text);

    const {
      data: [{ embedding }],
    } = await openai.embeddings.create({
      model: 'text-embedding-3-small',
      input: [query],
    });

    const embeddingString = JSON.stringify(embedding);

    const sections = await db
      .select()
      .from(documentSections)
      .where(sql`${documentSections.embedding} <#> ${embeddingString} < 0.7`)
      .orderBy(sql`${documentSections.embedding} <#> ${embeddingString}`)
```

```
      .limit(5);

    const introduction = 'Використовуй нижченаведені секції для відповіді на
запитання.';
    const question = `\n\nЗапитання: ${query}`;
    let message = introduction;
    for (const section of sections) {
      const nextSection = `\n\nSection:\n"""\n${section.content}"""\n`;
      if (numTokens(message + nextSection + question) > TOKEN_BUDGET) {
        break;
      }

      message += nextSection;
    }
    message += question;

    const response = await openai.chat.completions.create({
      model: 'gpt-3.5-turbo',
      messages: [
        {
          role: 'system',
          content: systemMessage,
        },
        {
          role: 'user',
          content: message,
        },
      ],
      temperature: 0,
      // top_p: 1,
      // frequency_penalty: 0,
      // presence_penalty: 0,
      // max_tokens: 1500,
      // n: 1,
    });

    const answer = response.choices[0].message.content as string;

    ctx.reply(answer, {
      parse_mode: 'Markdown',
    });
  } catch (error) {
```

```
    console.error('Error processing message:', error);
    ctx.reply('Вибачте, сталася помилка під час обробки вашого запиту. Будь ласка, спробуйте ще раз.');
  }
});

bot.launch(() => {
  console.log('□ Telegram bot is running!');
});
```

Додаток Б

Основний сервер завантаження та обробки файлів

```
const app = new Hono()
 .post('/upload', async (ctx) => {
    const user = ctx.get('user');

    if (!user) {
      return ctx.json({ error: 'Unauthorized' }, 401);
    }

    const body = await ctx.req.parseBody();
    const file = body.file;

    if (!(file instanceof File)) {
      return ctx.json({ error: 'No file uploaded' }, 400);
    }

    const filePath = path.join(uploadDirectory, file.name);

    try {
      if (!fs.existsSync(uploadDirectory)) {
        fs.mkdirSync(uploadDirectory, { recursive: true });
      }

      const writeStream = fs.createWriteStream(filePath);
      const readStream = file.stream();

      await pipelineAsync(readStream as unknown as NodeJS.ReadableStream,
writeStream);

      return ctx.json({ status: 'success', message: 'File uploaded
successfully', path: filePath });
    } catch (err) {
      console.error('Error uploading file:', err);
      return ctx.json({ error: 'Error uploading file' }, 500);
    }
  });

 cron.schedule('*/1 * * * *', processFiles);
```

Додаток В

Код функції proccessFiles

```
import { promises as fs } from 'node:fs';
import path from 'node:path';
import { processFile } from './utils/files';
import JSZip from 'jszip';

const uploadDirectory = path.join(__dirname, 'uploads');
const processingFiles = new Set<string>();

export const processFiles = async () => {
  try {
    const files = await fs.readdir(uploadDirectory); // Читання файлів з
директорії для завантаження
    console.log('Files in upload directory:', files.length);
    for (const file of files) {
      const filePath = path.join(uploadDirectory, file);

      if (processingFiles.has(filePath) || processFiles.length >= 1) {
        break; // Пропуск файлу, якщо він вже обробляється або кількість
оброблюваних файлів перевищує 1
      }

      processingFiles.add(filePath);

      try {
        const fileBuffer = await fs.readFile(filePath); // Читання файлу у
буфер
        const type = path.extname(file).substring(1); // Визначення типу
файлу за його розширенням

        if (type === 'zip') {
          const zip = new JSZip();
          await zip.loadAsync(fileBuffer); // Завантаження zip-файлу

          const zipFiles = zip.file(/.*/);

          for (const zipFile of zipFiles) {
            const name = zipFile.name.split('/').pop() as string;
            const type = name.split('.').pop() as string;
            const buffer = await zipFile.async('nodebuffer');
            await processFile(name, type, buffer); // Обробка файлу з архіву
```

```
          }
        } else {
          const name = path.basename(file);
          await processFile(name, type, fileBuffer); // Обробка звичайного
файлу
        }

        await fs.unlink(filePath); // Видалення файлу після обробки
      } catch (error) {
        console.error(`Error processing file: ${filePath}`, error); //
Логування помилки при обробці файлу
      } finally {
        processingFiles.delete(filePath);
        console.log('File processed:', filePath);
      }
    }
  } catch (error) {
    console.error('Error processing files:', error); // Логування загальної
помилки обробки файлів
  }
};
```