НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
В.о. завідувача кафедри
_____ Артем ВОЛОКИТА .
(підпис)

“__” _____2026 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

на тему: Система голосової взаємодії на основі методів штучного інтелекту.

Виконав : студентка __4__ курсу, групи _ІМ-21
(шифр групи)

_____Бондар Антоніна Андріївна_____ _____ ___ _____ ___
(прізвище, ім’я, по батькові) (підпис)

Ке До захисту допущенорівник _____д.т.н., проф. Стіренко Сергій Григорович _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) ас., д-р. філос. Коренко Д.В._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2026 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Артем ВОЛОКИТА .
(підпис)
“__” _____2026 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Бондар Антоніни Андріївни

1. Тема проєкту *Система голосової взаємодії на основі методів штучного інтелекту.*
керівник проєкту д.т.н., проф. Стіренко Сергій Григорович ,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 3 червня 2026 року № 2055-с.

2. Термін здачі студентом закінченого проєкту 6 червня 2026 р.

3. Вихідні дані до проєкту *Технічна документація, науково-технічна література, документація бібліотек штучного інтелекту, програмні засоби обробки природної мови, технології голосової взаємодії..*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Огляд систем голосової взаємодії та методів штучного інтелекту.

Розділ 2. Огляд технологій та підходів у розробці системи голосової взаємодії.

Розділ 3. Опис процесу розробки системи голосової взаємодії.

Розділ 4. Огляд та тестування розробленої системи голосової взаємодії.

5. Перелік графічного матеріалу: принципова схема, функціональна схема, структурна схема системи.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Коренко Д.В.		

7. Дата видачі завдання «1» квітня 2026 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*12.02.2026 – 20.02.2026*	
2.	*Вивчення та аналіз завдання*	*20.02.2026 – 15.03.2026*	
3.	*Розробка архітектури та загальної структури системи*	*10.03.2026 – 20.05.2026*	
4.	*Розробка структур окремих частин системи*	09.05.2026-31.05.2026	
5.	*Програмна реалізація системи голосової взаємодії*	25.05.2026-31.05.2026	
6.	*Оформлення пояснювальної записки*	01.06.2026-02.06.2026	
7.	*Захист програмного продукту*	05.06.2026	
8.	*Передзахист*	09.06.2026	
9.	*Захист*	16.06.2026	

Студент-дипломник _____.
(підпис)

Керівник проєкту _____.
(підпис)

АНОТАЦІЯ

У даній роботі було проведено аналіз сучасних систем голосової взаємодії та методів штучного інтелекту з метою розробки системи для автоматизованого спілкування з користувачем у текстовому та голосовому режимах. У результаті виконання роботи розроблено систему голосової взаємодії мовою програмування Python, яка забезпечує обробку голосових і текстових запитів, пошук релевантної інформації у базі знань та генерацію відповідей за допомогою великої мовної моделі.

У розробленій системі використано підхід Retrieval-Augmented Generation (RAG), що дозволяє формувати відповіді на основі локальної бази документів. Система реалізована з використанням технологій Ollama та Streamlit і підтримує голосове введення та озвучення відповідей користувачу.

Ключові слова: штучний інтелект, голосова взаємодія, обробка природної мови, велика мовна модель, RAG, Python, Ollama, Streamlit.

ANNOTATION

This work presents an analysis of modern voice interaction systems and artificial intelligence methods aimed at developing a system for automated communication with users in both text and voice modes. As a result of the work, a voice interaction system was developed using the Python programming language, providing processing of voice and text queries, retrieval of relevant information from a knowledge base, and response generation using a large language model.

The developed system utilizes the Retrieval-Augmented Generation (RAG) approach, which enables generating responses based on a local document database. The system was implemented using Ollama and Streamlit technologies and supports voice input as well as voice playback of responses.

Keywords: artificial intelligence, voice interaction, natural language processing, large language model, RAG, Python, Ollama, Streamlit.

Справки	Формат	Значення	Найменування	Кіл. листів	№ екземпляра	Додаток
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Система голосової взаємодії	4		
			на основі методів штучного інтелекту.			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Система голосової взаємодії	78		
			на основі методів штучного інтелекту.			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Система голосової взаємодії	1		
			на основі методів штучного інтелекту.			
			Принципова схема (схема алгоритму)			
	А4	***ІАЛЦ.467200.005 Д2***	Система голосової взаємодії	1		
			на основі методів штучного інтелекту.			
			Функціональна схема (діаграма			
			компонентів)			
	А4	***ІАЛЦ.467200.006 Д3***	Система голосової взаємодії	1		
			на основі методів штучного інтелекту.			
			Структурна схема системи			
	А4	***ІАЛЦ.467200.007 Д4***	Система голосової взаємодії	16		
			на основі методів штучного інтелекту.			
			Текст програмного коду			

ІАЛЦ.467200.001 ОА

Зм	Лист	№ докум.	Підп	Дата
Розробив		Бондар А. А.		
Перевірив		Стіренко С. Г.		
Реценз.				
Н. Контр.		Коренко Д. В.		
Затв.		Волокита А. М.		

Система голосової взаємодії на основі методів штучного інтелекту
Опис альбому

Літ.	Аркуш	Аркушів
	1	1

НТУУ "КПІ" ФІОТ ІМ-21

ТЕХНІЧНЕ ЗАВДАННЯ

ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Система голосової взаємодії на основі методів штучного інтелекту»

Київ – 2026

ЗМІСТ

НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання визначає вимоги до розробки системи голосової взаємодії на основі методів штучного інтелекту. Система призначена для обробки текстових і голосових запитів користувача, пошуку інформації у базі знань та формування відповідей за допомогою великої мовної моделі.

Основною областю застосування системи є автоматизоване надання інформації користувачам у текстовому та голосовому форматах. Система може використовуватись як інформаційний помічник для взаємодії з користувачем за допомогою технологій обробки природної мови.

ПІДСТАВИ ДЛЯ РОЗРОБКИ

Розробка системи виконується в межах бакалаврського дипломного проєкту за спеціальністю «Інженерія програмного забезпечення» Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

Підставою для виконання роботи є затверджене завдання на дипломний проєкт та необхідність створення сучасної системи голосової взаємодії з використанням методів штучного інтелекту.

МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки є створення системи голосової взаємодії, яка забезпечує обробку текстових та голосових запитів користувача, пошук релевантної інформації та генерацію відповідей із використанням великої мовної моделі.

Система повинна забезпечувати зручну взаємодію користувача із базою знань, підтримувати голосове введення, текстовий режим роботи та озвучення відповідей користувачу.

Призначенням системи є автоматизація процесу надання інформації користувачам із використанням сучасних технологій штучного інтелекту та обробки природної мови.

ДЖЕРЕЛА РОЗРОБКИ

Під час розробки використовуються технічна документація, наукові та навчальні матеріали, офіційна документація програмних бібліотек і фреймворків, а також інформаційні ресурси мережі Інтернет.

Для реалізації системи використовуються сучасні технології та засоби розробки програмного забезпечення, пов’язані з обробкою природної мови та системами штучного інтелекту.

ТЕХНІЧНІ ВИМОГИ

Вимоги до розробленого продукту

Розроблена система повинна забезпечувати:

- обробку текстових та голосових запитів користувача;
- пошук інформації у базі знань;
- генерацію відповідей за допомогою великої мовної моделі;
- підтримку голосового відтворення відповідей;
- роботу через вебінтерфейс;
- збереження коректності та зрозумілості відповідей.

Інтерфейс системи повинен бути простим та зрозумілим для користувача.

Вимоги до програмного забезпечення

- операційна система Windows 10 або новіше;
- Python 3;
- Ollama.

Вимоги до апаратної частини

- процесор із частотою не менше 2 ГГц;
- оперативна пам’ять від 8 ГБ;
- наявність мікрофона та аудіопристроїв;
- стабільне підключення до мережі Інтернет.

ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Аналіз предметної області та існуючих рішень	27.01.2026-31.01.2026
Проєктування архітектури системи	31.01.2026-14.02.2026
Реалізація серверної частини системи	14.02.2026-14.03.2026
Реалізація модуля голосової взаємодії	14.03.2026-16.04.2026
Реалізація користувацького інтерфейсу	16.04.2026-07.05.2026
Тестування системи та виправлення помилок	07.05.2026-15.05.2026
Оформлення пояснювальної записки	15.05.2026-25.05.2026

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Система голосової взаємодії на основі методів штучного інтелекту»

Київ – 2026

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

AI (Artificial Intelligence) — штучний інтелект

API (Application Programming Interface) — інтерфейс програмування застосунків

HTTP (HyperText Transfer Protocol) — протокол передачі гіпертексту

JSON (JavaScript Object Notation) — формат обміну даними

LLM (Large Language Model) — велика мовна модель

NLP (Natural Language Processing) — обробка природної мови

RAG (Retrieval-Augmented Generation) — генерація відповідей із використанням пошуку інформації

REST (Representational State Transfer) — архітектурний стиль побудови вебсервісів

UI (User Interface) — інтерфейс користувача

STT (Speech-to-Text) — перетворення мовлення у текст

TTS (Text-to-Speech) — синтез мовлення з тексту

DB (Database) — база даних

GPU (Graphics Processing Unit) — графічний процесор

CPU (Central Processing Unit) — центральний процесор

ВСТУП

У сучасному світі інформаційні технології відіграють важливу роль у повсякденному житті людини. Одним із найбільш актуальних напрямків розвитку програмного забезпечення є використання методів штучного інтелекту для автоматизації взаємодії між користувачем та інформаційними системами. Особливу популярність останніми роками отримали системи голосової взаємодії, які дозволяють користувачам отримувати інформацію та взаємодіяти із системою за допомогою природної мови.

Розвиток технологій обробки природної мови та великих мовних моделей значно розширив можливості сучасних систем штучного інтелекту. Такі системи здатні аналізувати текстові та голосові запити користувача, виконувати пошук інформації, підтримувати діалог і формувати змістовні відповіді у природній формі. Завдяки цьому системи голосової взаємодії активно використовуються у віртуальних помічниках, інформаційних сервісах, чатботах та системах підтримки користувачів. [3] Значний внесок у розвиток методів штучного інтелекту, глибокого навчання та інтелектуальних інтерфейсів людино-машинної взаємодії зроблено науковою школою кафедри обчислювальної техніки ФІОТ КПІ ім. Ігоря Сікорського. Зокрема, у роботах цієї школи досліджено застосування глибокого навчання для аналізу та сегментації зображень [27], побудову інтелектуальних мультимодальних інтерфейсів на основі доповненої реальності та інтерфейсів “мозок–комп’ютер” для людино-машинної взаємодії [28], оцінювання стану користувача за даними мультимодальної взаємодії засобами глибокого навчання [29], а також масштабування спеціалізованих тензорних обчислювальних архітектур для моделей глибокого навчання [30]. Ці дослідження формують методологічне підґрунтя для побудови систем голосової взаємодії на основі методів штучного інтелекту.

Важливою перевагою сучасних AI-систем є можливість використання локальних баз знань і технологій Retrieval-Augmented Generation (RAG), що дозволяють поєднувати генерацію відповідей мовною моделлю з пошуком релевантної інформації у документах. Це підвищує точність відповідей та дозволяє адаптувати систему до конкретної предметної області.

Актуальність теми полягає у зростанні потреби в інтелектуальних системах, які забезпечують швидку та зручну взаємодію користувача із програмним забезпеченням у голосовому та текстовому форматах. Використання методів штучного інтелекту дозволяє автоматизувати процес надання інформації та зробити взаємодію із системою більш природною для користувача.

Метою дипломного проєкту є розробка системи голосової взаємодії на основі методів штучного інтелекту, яка забезпечує обробку текстових і голосових запитів, пошук інформації у базі знань та генерацію відповідей за допомогою великої мовної моделі.

Для досягнення поставленої мети в роботі виконано аналіз сучасних систем голосової взаємодії, досліджено методи обробки природної мови та великі мовні моделі, реалізовано систему пошуку інформації у базі знань, обробку голосових запитів користувача та клієнт-серверну архітектуру системи. Також проведено тестування розробленої системи та аналіз результатів її роботи.

Об’єктом дослідження є системи голосової взаємодії на основі методів штучного інтелекту.

Предметом дослідження є методи обробки природної мови, великі мовні моделі та технології голосової взаємодії в інформаційних системах.

РОЗДІЛ 1. ОГЛЯД СИСТЕМ ГОЛОСОВОЇ ВЗАЄМОДІЇ ТА МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ

1.1 Визначення понять

Теоретичне дослідження систем голосової взаємодії потребує визначення основних понять та технологій, які використовуються під час розробки сучасних інформаційних систем на основі штучного інтелекту. В основі таких систем лежить поєднання методів обробки природної мови, великих мовних моделей та технологій голосової взаємодії.

Людино-машинна взаємодія (Human-Computer Interaction, HCI) — це напрямок досліджень, пов’язаний із розробкою способів взаємодії користувача з комп’ютерними системами. Основною метою HCI є створення зручних, зрозумілих та ефективних інтерфейсів для користувача. [28]

Система голосової взаємодії (Voice User Interface, VUI) — це тип інтерфейсу, у якому взаємодія між користувачем і системою здійснюється за допомогою голосових команд та мовлення. Такі системи дозволяють виконувати обробку голосових запитів, формувати відповіді та озвучувати результати роботи системи.

Штучний інтелект (Artificial Intelligence, AI) — це галузь комп’ютерних наук, яка займається створенням систем, здатних виконувати задачі, що потребують аналізу інформації, навчання та прийняття рішень. Методи штучного інтелекту широко використовуються у сучасних інформаційних системах, чатботах та голосових помічниках. [24]

Обробка природної мови (Natural Language Processing, NLP) — це напрямок штучного інтелекту, який забезпечує аналіз, обробку та генерацію текстової інформації природною мовою. Методи NLP використовуються для розпізнавання мовлення, аналізу тексту та побудови діалогових систем. [1, 23]

Велика мовна модель (Large Language Model, LLM) — це нейромережева модель, навчена на великих обсягах текстових даних, яка здатна генерувати текст, відповідати на запити користувача та підтримувати діалог природною мовою. [2]

Для підвищення точності відповідей у сучасних AI-системах використовується підхід Retrieval-Augmented Generation (RAG). Даний підхід поєднує генерацію відповідей мовною моделлю із пошуком релевантної інформації у базі знань або документах. Це дозволяє системі формувати більш точні та контекстно-залежні відповіді.

Автоматичне розпізнавання мовлення (Speech-to-Text, STT) — це технологія перетворення голосового сигналу у текстовий формат. Дана технологія використовується для обробки голосових запитів користувача. [21]

Синтез мовлення (Text-to-Speech, TTS) — це технологія перетворення тексту у голосове повідомлення. Вона використовується для озвучення відповідей системи та реалізації голосового режиму взаємодії.

1.2 Огляд сучасних систем голосової взаємодії

1.2.1 Google Assistant

Google Assistant є однією з найбільш поширених систем голосової взаємодії, розробленою компанією Google. Система підтримує голосове керування, пошук інформації, виконання команд користувача та інтеграцію з різними сервісами й мобільними пристроями. [4]

Основою роботи Google Assistant є використання технологій обробки природної мови та машинного навчання для аналізу запитів користувача і формування відповідей. Система здатна розпізнавати голосові команди, підтримувати діалог та виконувати різні задачі в автоматичному режимі.

До переваг Google Assistant можна віднести швидкість роботи, підтримку великої кількості функцій та зручність голосової взаємодії. Недоліками системи є залежність від підключення до мережі Інтернет та обмежений доступ до локальних даних користувача.

1.2.2 Siri

Siri - це система голосового помічника, розроблена компанією Apple для пристроїв із операційною системою iOS. Система дозволяє користувачу взаємодіяти із мобільним пристроєм за допомогою голосових команд, виконувати пошук інформації, створювати нагадування та працювати із повідомленнями. [5]

Для обробки запитів Siri використовує технології штучного інтелекту та обробки природної мови. Система підтримує голосове керування та забезпечує інтеграцію із сервісами екосистеми Apple.

Перевагою Siri є зручність використання та інтеграція із пристроями Apple. До недоліків можна віднести обмежену підтримку окремих мов та меншу гнучкість при роботі із зовнішніми сервісами.

1.2.3 ChatGPT Voice

ChatGPT Voice є сучасною системою голосової взаємодії, побудованою на основі великих мовних моделей. Система забезпечує підтримку діалогу природною мовою, генерацію відповідей та обробку складних текстових і голосових запитів користувача. [6]

На відміну від класичних голосових помічників, ChatGPT Voice використовує великі мовні моделі для аналізу контексту діалогу та формування більш змістовних відповідей. Система може підтримувати тривалу взаємодію з користувачем і працювати із різними типами інформації.

Перевагами ChatGPT Voice є високий рівень розуміння природної мови, гнучкість у веденні діалогу та можливість використання сучасних AI-технологій. До недоліків можна віднести високі вимоги до обчислювальних ресурсів та залежність якості відповідей від використовуваної мовної моделі.

1.3 Методи обробки природної мови та великі мовні моделі

У сучасних інформаційних системах важливу роль відіграють технології обробки природної мови. Обробка природної мови (Natural Language Processing, NLP) є одним із напрямків штучного інтелекту, який забезпечує аналіз, інтерпретацію та генерацію текстової інформації природною мовою. Основною задачею NLP є забезпечення ефективної взаємодії між користувачем і комп'ютерною системою у зрозумілому для людини форматі.

Методи обробки природної мови широко використовуються у системах голосової взаємодії, чатботах, пошукових системах, сервісах автоматичного перекладу та інформаційних платформах. Основними задачами NLP є аналіз тексту, визначення контексту повідомлення, класифікація текстової інформації, розпізнавання мовлення та генерація відповідей.

Для реалізації сучасних систем обробки природної мови використовуються великі мовні моделі (Large Language Models, LLM). Великі мовні моделі являють собою нейромережеві системи, які навчаються на великих обсягах текстових даних та здатні виконувати широкий спектр задач, пов'язаних із генерацією та аналізом текстової інформації. [2, 24]

Сучасні мовні моделі будуються переважно на архітектурі Transformer, яка використовує механізм самоуваги (Self-Attention). Використання даної архітектури дозволяє моделі враховувати зв'язки між словами та реченнями у тексті, що забезпечує більш якісний аналіз контексту та генерацію відповідей. [7]

На відміну від класичних алгоритмів обробки тексту, великі мовні моделі здатні підтримувати природний діалог із користувачем, враховувати контекст повідомлень та формувати змістовні відповіді. Завдяки цьому LLM активно використовуються у сучасних AI-системах та системах голосової взаємодії.

Однією з особливостей великих мовних моделей є можливість генерації відповідей на основі природної мови без жорстко заданих сценаріїв взаємодії. Це дозволяє створювати більш гнучкі та адаптивні системи, здатні працювати із різними типами запитів користувача.

У сучасних системах голосової взаємодії великі мовні моделі поєднуються із технологіями автоматичного розпізнавання мовлення та синтезу голосу. Такий підхід дозволяє реалізувати повний цикл голосової взаємодії: від отримання голосового запиту користувача до генерації та озвучення відповіді.

У даному дипломному проєкті велика мовна модель використовується для генерації відповідей на текстові та голосові запити користувача. Для реалізації системи використовується локальна мовна модель, яка працює за допомогою платформи Ollama. Використання локальної моделі дозволяє забезпечити обробку даних у межах системи та інтеграцію AI-компонентів із базою знань.

Для реалізації взаємодії із мовною моделлю та організації логіки роботи системи використовуються сучасні інструменти обробки природної мови. Це дозволяє забезпечити підтримку текстового та голосового режимів роботи системи, а також генерацію відповідей у природній для користувача формі.

1.4 Використання підходу Retrieval-Augmented Generation (RAG)

У сучасних системах штучного інтелекту важливою задачею є забезпечення точності та актуальності відповідей, які формуються великою мовною моделлю. Одним із підходів, що використовується для вирішення даної задачі, є Retrieval-Augmented Generation (RAG). [8, 9]

Підхід RAG поєднує генерацію тексту великою мовною моделлю із пошуком релевантної інформації у зовнішніх джерелах даних або базі знань. На відміну від класичних мовних моделей, які формують відповіді лише на основі власних навчальних даних, RAG-системи додатково використовують інформацію із документів або локальних сховищ даних.

Основною перевагою підходу Retrieval-Augmented Generation є можливість підвищення точності та актуальності відповідей. Перед генерацією відповіді система виконує пошук релевантної інформації у базі знань, після чого знайдені фрагменти передаються мовній моделі як додатковий контекст для формування відповіді.

У більшості сучасних RAG-систем для пошуку інформації використовуються векторні бази даних та embeddings-моделі. Текстові документи попередньо розбиваються на окремі фрагменти, після чого кожен фрагмент перетворюється у векторне представлення. Під час обробки запиту користувача система виконує пошук найбільш релевантних фрагментів за допомогою порівняння векторних представлень. [10, 11, 12]

Використання RAG дозволяє зменшити кількість неточних або некоректних відповідей, які можуть виникати під час роботи великих мовних моделей. Також даний підхід забезпечує можливість роботи із локальними документами та спеціалізованими базами знань без необхідності повторного навчання моделі.

У системах голосової взаємодії підхід Retrieval-Augmented Generation дозволяє реалізувати більш точну обробку запитів користувача та формування відповідей на основі актуальної інформації. Це особливо важливо для інформаційних систем, які працюють із документацією, локальними даними або спеціалізованими знаннями.

У даному дипломному проєкті підхід RAG використовується для пошуку інформації у локальній базі знань та генерації відповідей на текстові й голосові запити користувача. Для реалізації системи використовується векторна база

даних, у якій зберігаються embeddings текстових документів. Після отримання запиту система виконує пошук релевантної інформації та передає знайдені дані великій мовній моделі для формування відповіді.

Використання підходу Retrieval-Augmented Generation дозволяє підвищити якість взаємодії користувача із системою та забезпечити більш точні й контекстно-залежні відповіді.

1.5 Порівняння існуючих рішень

На сьогодні існує велика кількість систем голосової взаємодії та AI-помічників, які використовують технології обробки природної мови та великі мовні моделі. Найбільш поширеними серед них є Google Assistant, Siri та ChatGPT Voice. Дані системи забезпечують підтримку голосових команд, пошук інформації та взаємодію користувача із програмним забезпеченням у природній формі.

Google Assistant та Siri орієнтовані переважно на інтеграцію із мобільними пристроями та екосистемами відповідних компаній. Дані системи підтримують голосове керування пристроєм, виконання команд користувача та доступ до різних онлайн-сервісів. Основною перевагою таких рішень є стабільність роботи та інтеграція із програмним забезпеченням мобільних платформ.

Системи, побудовані на основі великих мовних моделей, зокрема ChatGPT Voice, забезпечують більш гнучку взаємодію із користувачем та підтримку природного діалогу. Використання LLM дозволяє формувати змістовні відповіді та аналізувати контекст запитів користувача.

Разом із перевагами існуючі рішення мають певні обмеження. Більшість систем працюють через хмарні сервіси та потребують постійного підключення до мережі Інтернет. Також у багатьох випадках відсутня можливість роботи із локальними документами та власними базами знань користувача.

У даному дипломному проєкті розробляється система голосової взаємодії, яка використовує локальну велику мовну модель та підхід Retrieval-Augmented Generation (RAG) для роботи із локальною базою знань. Це дозволяє забезпечити більш гнучку інтеграцію із документами користувача та локальну обробку даних.

Таблиця 1.1 – Порівняння існуючих систем голосової взаємодії

Характеристика	**Google Assistant**	**Siri**	**ChatGPT Voice**	**Розроблена система**
Голосова взаємодія	Так	Так	Так	Так
Підтримка діалогу	Частково	Частково	Так	Так
Використання LLM	Ні	Ні	Так	Так
Робота з локальною базою знань	Ні	Ні	Частково	Так
Використання RAG	Ні	Ні	Частково	Так
Локальна обробка даних	Ні	Ні	Ні	Так
Голосове озвучення відповідей	Так	Так	Так	Так

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі було проведено аналіз сучасних систем голосової взаємодії та методів штучного інтелекту, які використовуються у сучасних інформаційних системах. Розглянуто основні поняття, пов’язані із технологіями голосової взаємодії, обробкою природної мови, великими мовними моделями та підходом Retrieval-Augmented Generation (RAG).

Було досліджено особливості роботи сучасних систем голосової взаємодії, зокрема Google Assistant, Siri та ChatGPT Voice. Проведене порівняння показало, що сучасні системи забезпечують підтримку голосових команд та діалогової взаємодії, однак більшість із них мають обмеження при роботі із локальними базами знань та локальною обробкою даних.

У результаті аналізу встановлено, що використання великих мовних моделей та підходу RAG дозволяє підвищити якість взаємодії користувача із системою, забезпечити більш точний пошук інформації та формування відповідей із врахуванням контексту запиту.

Проведений аналіз підтвердив доцільність розробки системи голосової взаємодії на основі методів штучного інтелекту із використанням локальної мовної моделі та технології Retrieval-Augmented Generation для роботи із базою знань користувача.

РОЗДІЛ 2
ПРОЄКТУВАННЯ СИСТЕМИ ГОЛОСОВОЇ ВЗАЄМОДІЇ НА ОСНОВІ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ

2.1 Розробка архітектури системи

2.1.1 Загальна архітектура системи

Під час проєктування системи голосової взаємодії на основі методів штучного інтелекту одним із ключових етапів є розробка її архітектури. Від обраної архітектури залежить можливість масштабування системи, зручність інтеграції окремих модулів та ефективність обробки запитів користувача.

Для розроблюваної системи було обрано клієнт-серверну архітектуру, яка передбачає розділення програмного забезпечення на користувацьку та серверну частини. Такий підхід дозволяє відокремити інтерфейс взаємодії з користувачем від модулів обробки даних, що спрощує подальшу модернізацію та підтримку системи.

До складу розробленої системи входять такі основні компоненти:

– користувацький вебінтерфейс, реалізований за допомогою Streamlit;

– серверна частина, реалізована із використанням FastAPI;

– модуль пошуку інформації на основі підходу Retrieval-Augmented Generation;

– векторна база даних ChromaDB для зберігання векторних представлень документів;

– велика мовна модель Qwen3, що виконується локально за допомогою платформи Ollama;

– модулі розпізнавання мовлення та синтезу голосу для підтримки голосового режиму взаємодії.

Загальна архітектура розробленої системи наведена на рисунку 2.1.

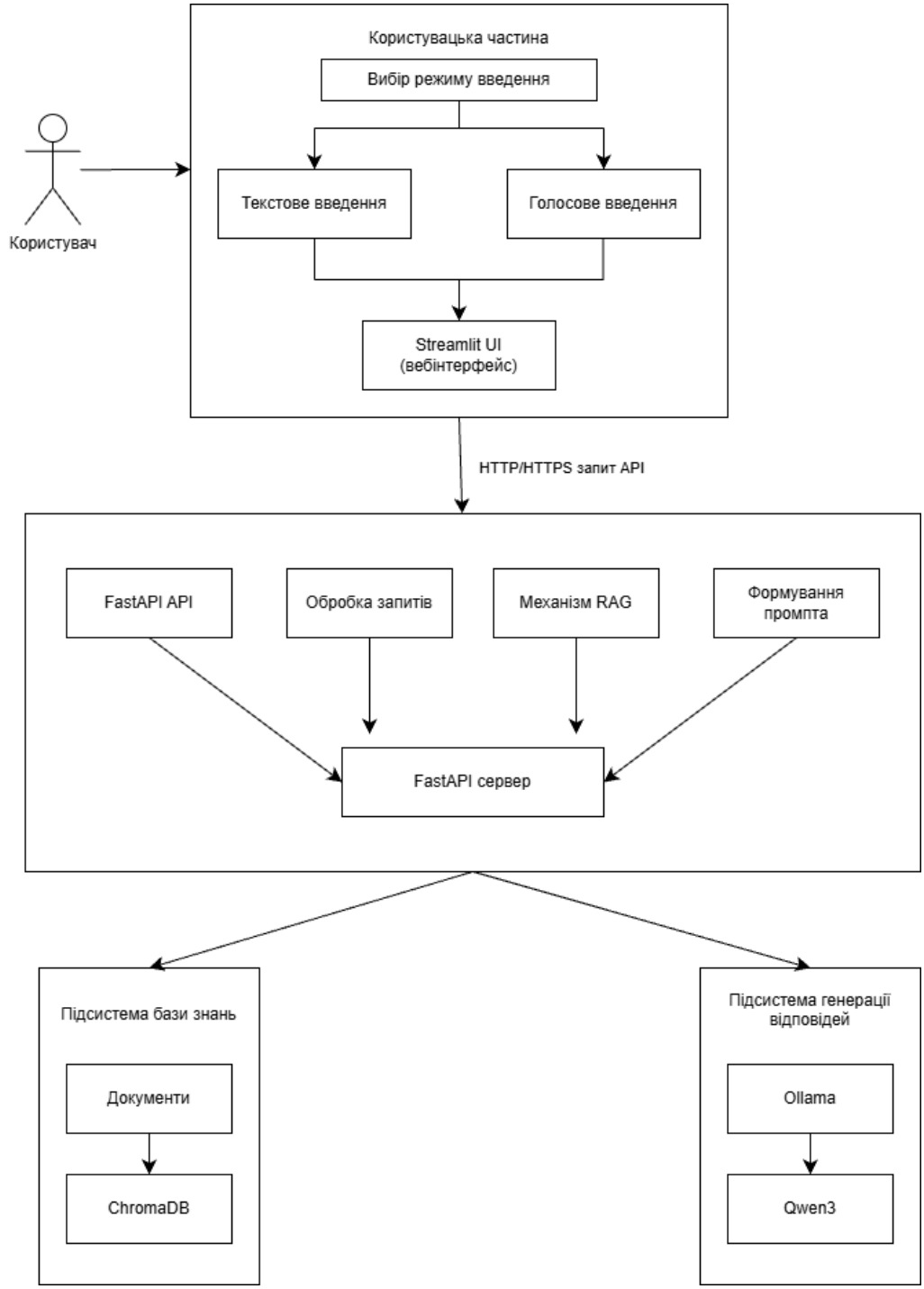


Рисунок 2.1 – Загальна архітектура системи голосової взаємодії

Як показано на рисунку 2.1, система має трирівневу структуру. На рівні користувацької частини користувач обирає режим введення та формує текстовий або голосовий запит через вебінтерфейс Streamlit. Сформований

запит передається до серверної частини за допомогою програмного інтерфейсу (HTTP/HTTPS запит API).

Серверна частина, реалізована засобами FastAPI, координує роботу основних функціональних блоків: обробки запитів, механізму Retrieval-Augmented Generation та формування промпта. У випадку голосового введення додатково виконується перетворення мовлення у текстове представлення.

Серверна частина взаємодіє з двома підсистемами. Підсистема бази знань містить документи, що зберігаються у вигляді векторних представлень у базі даних ChromaDB, та забезпечує пошук релевантної інформації. Підсистема генерації відповідей включає велику мовну модель Qwen3, що виконується локально за допомогою платформи Ollama. Знайдений контекст разом із запитом передається до мовної моделі для формування відповіді.

Згенерована відповідь повертається користувачеві через вебінтерфейс. За необхідності вона додатково перетворюється у голосове повідомлення за допомогою модуля синтезу мовлення.

2.1.2 Взаємодія програмних компонентів системи

Для забезпечення коректної роботи системи голосової взаємодії необхідно організувати ефективну взаємодію між окремими програмними компонентами. У розробленій системі обмін даними між модулями реалізовано за клієнт-серверним принципом із використанням програмного інтерфейсу прикладного програмування (API). Такий підхід забезпечує незалежність окремих компонентів та спрощує їх подальшу модернізацію.

Основними програмними компонентами системи є вебінтерфейс користувача, серверна частина, модуль Retrieval-Augmented Generation, векторна база даних ChromaDB та велика мовна модель Qwen3. Кожен із

компонентів виконує окремі функції та взаємодіє з іншими модулями для забезпечення повного циклу обробки запиту користувача.

Вебінтерфейс системи реалізовано за допомогою фреймворку Streamlit. Через нього користувач може вводити текстові запити або використовувати голосове введення. Після отримання запиту інформація передається до серверної частини системи для подальшої обробки.

Серверна частина реалізована засобами FastAPI та виконує функції координації роботи всіх компонентів системи. Після отримання запиту сервер обробляє його та передає до модуля пошуку інформації. Крім того, FastAPI забезпечує обмін даними між користувацьким інтерфейсом, векторною базою даних та мовною моделлю.

Для пошуку інформації використовується механізм Retrieval-Augmented Generation. На першому етапі запит користувача перетворюється у векторне представлення та використовується для пошуку найбільш релевантних документів у базі знань. Для зберігання векторних представлень документів використовується база даних ChromaDB, яка забезпечує швидкий семантичний пошук інформації.

Після завершення пошуку знайдені фрагменти документів формують додатковий контекст для генерації відповіді. Отриманий контекст разом із початковим запитом користувача передається до великої мовної моделі Qwen3, що виконується локально за допомогою платформи Ollama.

Модель Qwen3 аналізує отриманий контекст та формує відповідь на запит користувача. Згенерована відповідь повертається до серверної частини, після чого передається до вебінтерфейсу для відображення результату. У випадку використання голосового режиму взаємодії відповідь додатково може бути перетворена у звукове повідомлення засобами синтезу мовлення.

Взаємодію програмних компонентів системи зображено на рисунку 2.2.

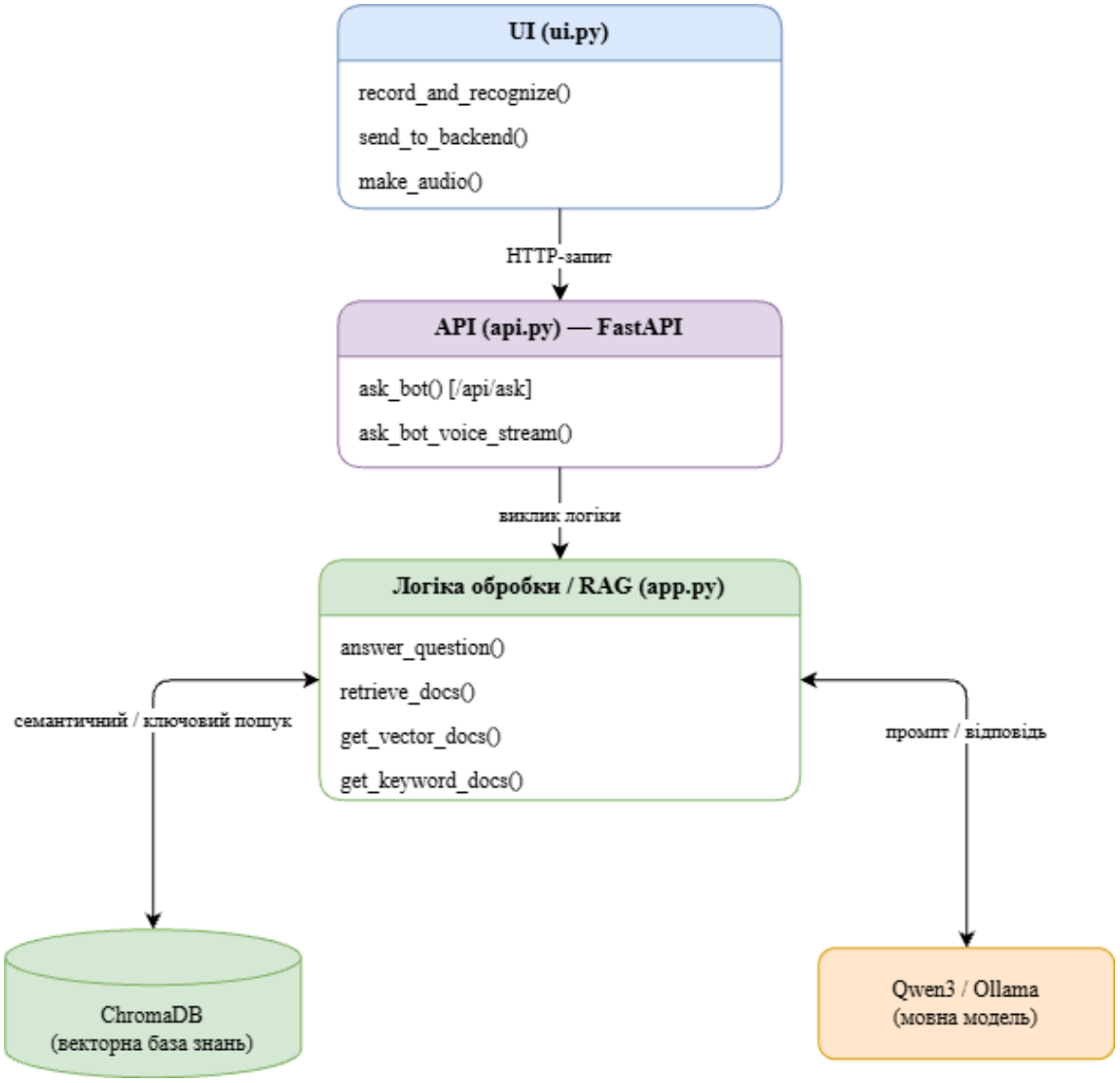


Рисунок 2.2 – Схема взаємодії програмних компонентів системи

Як показано на рисунку 2.2, користувацький інтерфейс взаємодіє із серверним модулем через HTTP-запит, а серверний модуль викликає модуль логіки обробки запитів. Останній взаємодіє з векторною базою даних ChromaDB для пошуку інформації та з мовною моделлю Qwen3 для генерації відповіді. Кожен компонент виконує окрему функцію, що забезпечує чіткий розподіл відповідальності між модулями системи.

Таким чином, запропонована організація взаємодії забезпечує послідовну обробку запитів користувача та дозволяє розподілити функціональність між окремими програмними модулями. Використання незалежних компонентів спрощує підтримку програмного забезпечення, забезпечує можливість його масштабування та створює умови для подальшого розширення функціональних можливостей системи.

2.1.3 Алгоритм функціонування системи

Після визначення архітектури та організації взаємодії між програмними компонентами необхідно описати алгоритм функціонування системи голосової взаємодії. Алгоритм визначає послідовність виконання основних операцій під час обробки запиту користувача та формування відповіді.

Робота системи починається з отримання запиту від користувача через вебінтерфейс. На першому етапі здійснюється вибір режиму введення запиту. Якщо обрано голосовий режим, виконується розпізнавання мовлення та перетворення голосового сигналу в текстове представлення. У разі текстового введення запит одразу передається на подальшу обробку.

Отриманий текстовий запит передається до серверної частини системи, реалізованої засобами FastAPI. Сервер виконує початкову обробку запиту та ініціює пошук релевантних документів у векторній базі даних ChromaDB на основі механізму Retrieval-Augmented Generation.

Залежно від результату пошуку алгоритм передбачає розгалуження. Якщо релевантні документи знайдено, на їх основі формується контекст, який доповнює запит користувача. Якщо документів не знайдено, формується промпт без додаткового контексту. У такий спосіб система здатна формувати відповідь навіть за відсутності релевантної інформації у базі знань.

Сформований промпт передається до великої мовної моделі Qwen3, що виконується локально за допомогою платформи Ollama. Модель аналізує отриману інформацію та генерує відповідь на запит користувача.

На завершальному етапі система перевіряє, у якому режимі працює користувач. У голосовому режимі згенерована відповідь додатково перетворюється у звукове повідомлення засобами синтезу мовлення. Після цього відповідь відображається користувачеві через вебінтерфейс.

Алгоритм функціонування системи голосової взаємодії зображено на рисунку 2.3.

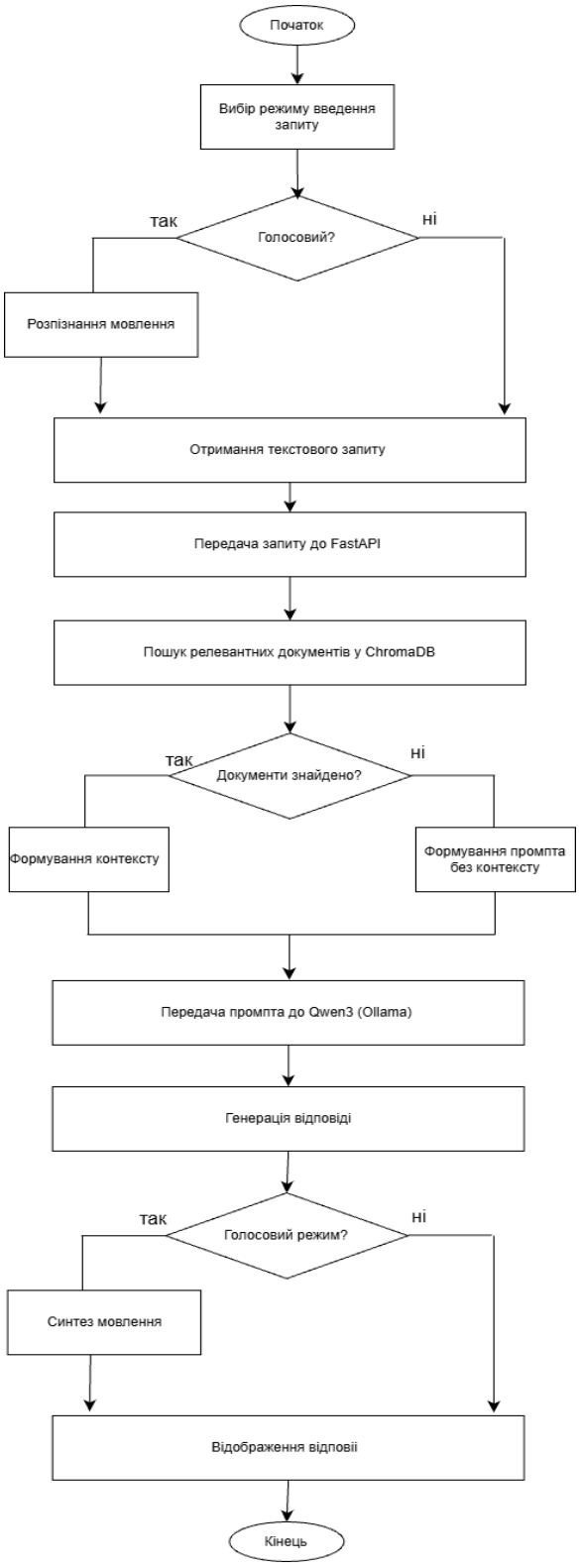


Рисунок 2.3 – Алгоритм функціонування системи голосової взаємодії

Таким чином, запропонований алгоритм забезпечує послідовну обробку запитів користувача, пошук релевантної інформації та генерацію відповідей із використанням технології Retrieval-Augmented Generation і великої мовної моделі Qwen3.

2.2 Обґрунтування вибору технологій розробки

Під час вибору технологій для реалізації системи голосової взаємодії враховувалися такі критерії, як продуктивність, масштабованість, простота інтеграції, підтримка сучасних методів штучного інтелекту, можливість локального розгортання та доступність необхідних програмних бібліотек. Особлива увага приділялася забезпеченню автономної роботи системи без використання зовнішніх хмарних сервісів, що дозволяє підвищити рівень конфіденційності даних користувачів.

Для реалізації системи було обрано мову програмування Python, вебфреймворк FastAPI для створення серверної частини, платформу Streamlit для розробки користувацького інтерфейсу, платформу Ollama для локального запуску великої мовної моделі Qwen3 та векторну базу даних ChromaDB для реалізації механізму семантичного пошуку інформації.

У наступних підрозділах наведено обґрунтування вибору кожної з використаних технологій та їх роль у розробленій системі голосової взаємодії.

2.2.1 Вибір програмної платформи розробки

Одним із важливих етапів проєктування програмного забезпечення є вибір мови програмування, яка забезпечуватиме реалізацію всіх функціональних можливостей системи. Від обраної технології залежить швидкість розробки, можливість інтеграції із зовнішніми програмними

компонентами, підтримка сучасних інструментів штучного інтелекту та подальший супровід програмного забезпечення.

Для реалізації системи голосової взаємодії було обрано мову програмування Python. Вибір цієї мови обумовлений її широким застосуванням у сфері штучного інтелекту, машинного навчання, обробки природної мови та розробки вебзастосунків. Python є однією з найбільш популярних мов програмування для створення інтелектуальних інформаційних систем завдяки великій кількості доступних бібліотек і програмних засобів. [22]

Важливою перевагою Python є наявність розвиненої екосистеми програмних інструментів для роботи з великими мовними моделями та технологіями обробки природної мови. Саме для Python реалізовано підтримку більшості сучасних фреймворків та бібліотек, які використовуються під час створення систем на основі штучного інтелекту.

У розроблюваній системі Python забезпечує інтеграцію всіх основних компонентів програмного забезпечення. Зокрема, серверна частина реалізується із використанням FastAPI [15], користувацький інтерфейс створюється за допомогою Streamlit [16], робота з великою мовною моделлю здійснюється через платформу Ollama [18], а для зберігання та пошуку векторних представлень документів використовується база даних ChromaDB [17]. Використання єдиної мови програмування для реалізації всіх компонентів системи дозволяє спростити процес розробки та забезпечити узгоджену взаємодію між окремими модулями.

Додатковою перевагою Python є його простий та зрозумілий синтаксис, що полегшує підтримку програмного коду та його подальшу модернізацію. Це особливо важливо для систем, які можуть розширюватися новими функціями або інтегрувати додаткові модулі штучного інтелекту.

Таким чином, використання Python дозволяє ефективно реалізувати всі функціональні компоненти системи голосової взаємодії, забезпечити інтеграцію сучасних технологій штучного інтелекту та створити основу для подальшого розвитку програмного забезпечення.

2.2.2 Вибір FastAPI для реалізації серверної частини

Серверна частина є одним із ключових компонентів системи голосової взаємодії, оскільки забезпечує обробку запитів користувачів, координацію роботи програмних модулів та обмін даними між окремими компонентами системи. Під час проєктування серверної частини особливу увагу було приділено продуктивності, масштабованості та можливості інтеграції із засобами штучного інтелекту.

Для реалізації серверної частини було обрано вебфреймворк FastAPI. Даний фреймворк дозволяє створювати високопродуктивні вебзастосунки та програмні інтерфейси прикладного програмування (API) мовою Python. Однією з його основних переваг є підтримка асинхронної обробки запитів, що дає змогу ефективно працювати з великою кількістю звернень користувачів.

У розроблюваній системі FastAPI використовується як центральний елемент взаємодії між користувацьким інтерфейсом, векторною базою даних ChromaDB та великою мовною моделлю Qwen3. Після отримання запиту від користувача сервер виконує його обробку, ініціює пошук релевантних документів, формує контекст для мовної моделі та повертає сформовану відповідь до інтерфейсу користувача.

Важливою перевагою FastAPI є його сумісність із сучасними бібліотеками машинного навчання та обробки природної мови. Це дозволяє інтегрувати в систему модулі Retrieval-Augmented Generation, засоби роботи з

векторними представленнями тексту та великі мовні моделі без необхідності використання додаткових програмних засобів.

Крім того, FastAPI автоматично формує документацію API, що спрощує тестування та подальшу підтримку програмного забезпечення. Такий підхід забезпечує зручність розробки та дозволяє швидко розширювати функціональні можливості системи.

Таким чином, використання FastAPI забезпечує ефективну взаємодію між усіма компонентами системи голосової взаємодії, підтримує сучасні підходи до розробки програмного забезпечення та створює основу для подальшого розвитку програмного продукту.

2.2.3 Вибір Streamlit для реалізації користувацького інтерфейсу

Одним із завдань під час розробки системи було створення інтерфейсу, який забезпечував би зручну взаємодію користувача з мовною моделлю та механізмом пошуку інформації. Крім підтримки текстового введення, інтерфейс повинен забезпечувати можливість використання голосових запитів та відображення результатів роботи системи в режимі реального часу.

Для реалізації користувацького інтерфейсу було розглянуто декілька підходів, серед яких використання класичних вебтехнологій (HTML, CSS та JavaScript) і спеціалізованих Python-фреймворків. З огляду на те, що всі інші компоненти системи реалізуються мовою Python, доцільним виявилося використання рішення, яке не потребує створення окремої фронтенд-частини.

У розроблюваній системі було прийнято рішення використовувати Streamlit. Основною причиною такого вибору стала можливість швидкої інтеграції інтерфейсу з компонентами штучного інтелекту та серверною частиною без використання додаткових засобів розробки. Це дозволило

зосередити основну увагу на реалізації функцій голосової взаємодії та механізму Retrieval-Augmented Generation.

Використання Streamlit забезпечує можливість реалізації єдиного середовища для введення текстових та голосових запитів, відображення відповідей мовної моделі та контролю роботи системи. Крім того, такий підхід спрощує подальшу модернізацію інтерфейсу та додавання нових функціональних можливостей.

Таким чином, вибір Streamlit дозволив скоротити складність розробки користувацького інтерфейсу, забезпечити його інтеграцію з іншими компонентами системи та створити зручний засіб взаємодії користувача із системою голосової взаємодії.

2.2.4 Вибір Ollama та мовної моделі Qwen3

Одним із ключових компонентів розроблюваної системи є підсистема генерації відповідей на запити користувача. Для реалізації цієї функціональності необхідно використовувати велику мовну модель, здатну аналізувати природну мову, враховувати контекст запиту та формувати змістовні відповіді.

На початковому етапі розглядалася можливість використання хмарних сервісів на базі великих мовних моделей. Проте такий підхід має низку недоліків, серед яких залежність від зовнішніх сервісів, необхідність постійного підключення до мережі Інтернет, обмеження на кількість запитів та ризики передачі даних користувачів стороннім системам. З огляду на це було прийнято рішення використовувати локальну мовну модель.

Для запуску великої мовної моделі було обрано платформу Ollama. Дане програмне забезпечення дозволяє виконувати сучасні мовні моделі безпосередньо на локальному комп'ютері, забезпечуючи простоту розгортання

та інтеграції з Python-застосунками. Використання Ollama дозволяє організувати автономну роботу системи без необхідності звернення до зовнішніх API.

Серед доступних мовних моделей було розглянуто декілька варіантів, зокрема Llama, Mistral та Qwen. Для реалізації системи було обрано модель Qwen3, яка демонструє високі показники якості генерації тексту та підтримує ефективну роботу з багатомовними запитами. Додатковою перевагою моделі є здатність працювати з контекстною інформацією, що є необхідною умовою для реалізації механізму Retrieval-Augmented Generation. [19]

У розроблюваній системі модель Qwen3 використовується для формування відповідей на основі запиту користувача та додаткового контексту, отриманого з бази знань. Такий підхід дозволяє поєднати можливості великої мовної моделі з актуальною інформацією, що зберігається у документах системи.

Використання Ollama та мовної моделі Qwen3 забезпечує локальну обробку даних, незалежність від зовнішніх сервісів та можливість подальшого масштабування системи. Крім того, обране рішення дозволяє інтегрувати мовну модель із механізмом Retrieval-Augmented Generation та забезпечити генерацію більш релевантних відповідей на запити користувачів.

Таким чином, використання платформи Ollama та мовної моделі Qwen3 є доцільним рішенням для реалізації підсистеми генерації відповідей у системі голосової взаємодії, оскільки забезпечує необхідний рівень функціональності, автономності та якості обробки природної мови.

2.2.5 Вибір ChromaDB для зберігання векторних представлень документів

Однією з основних особливостей розроблюваної системи є використання механізму Retrieval-Augmented Generation для пошуку інформації у базі знань та формування більш релевантних відповідей на запити користувачів. Реалізація такого підходу потребує використання засобу зберігання векторних представлень документів та виконання семантичного пошуку.

Під час проєктування системи було розглянуто декілька варіантів організації зберігання даних. Використання традиційних реляційних баз даних не є оптимальним рішенням для роботи з векторними представленнями тексту, оскільки вони орієнтовані переважно на пошук за структурованими даними та не забезпечують ефективного виконання операцій семантичного пошуку.

Для реалізації підсистеми пошуку знань було обрано векторну базу даних ChromaDB. Дане рішення дозволяє зберігати векторні представлення документів та швидко знаходити фрагменти тексту, зміст яких найбільш відповідає запиту користувача. Використання ChromaDB забезпечує ефективну роботу механізму Retrieval-Augmented Generation та спрощує інтеграцію з іншими компонентами системи.

У розроблюваній системі ChromaDB використовується для зберігання векторних представлень документів, що формуються на етапі підготовки бази знань. Під час обробки запиту користувача система створює його векторне представлення та виконує пошук найбільш схожих документів у базі даних. Знайдені фрагменти використовуються для формування контексту, який передається до великої мовної моделі Qwen3.

Важливою перевагою ChromaDB є підтримка інтеграції з Python-бібліотеками та інструментами розробки систем штучного інтелекту. Це дозволяє використовувати єдине програмне середовище для реалізації всіх

компонентів системи та спрощує подальшу модернізацію програмного забезпечення.

Крім того, використання ChromaDB забезпечує можливість масштабування бази знань шляхом додавання нових документів без необхідності зміни архітектури системи. Такий підхід дозволяє розширювати обсяг доступної інформації та підвищувати якість відповідей, що формуються мовною моделлю.

Таким чином, вибір ChromaDB обумовлений необхідністю реалізації ефективного семантичного пошуку, підтримкою роботи з векторними представленнями тексту та можливістю інтеграції з іншими компонентами системи голосової взаємодії. Використання даної бази даних забезпечує функціонування механізму Retrieval-Augmented Generation та підвищує релевантність відповідей, що генеруються мовною моделлю.

2.3 Проєктування механізму Retrieval-Augmented Generation

Однією з основних проблем сучасних великих мовних моделей є обмеженість знань, отриманих під час навчання, а також можливість формування неточних або недостовірних відповідей. Незважаючи на високий рівень розвитку мовних моделей, вони не мають прямого доступу до актуальних документів та не можуть використовувати інформацію, яка не була включена до навчальної вибірки.

Для підвищення якості відповідей у розроблюваній системі використовується підхід Retrieval-Augmented Generation (RAG). Даний підхід поєднує можливості семантичного пошуку інформації та генерації тексту великою мовною моделлю. Перед формуванням відповіді система виконує

пошук релевантних документів у базі знань та використовує знайдену інформацію як додатковий контекст під час генерації відповіді.

Використання механізму RAG дозволяє зменшити ймовірність виникнення помилкових відповідей, підвищити релевантність результатів та забезпечити використання актуальної інформації з документів, що зберігаються у базі знань системи. У наступних підрозділах розглянуто принцип роботи механізму Retrieval-Augmented Generation, процес формування векторних представлень документів та метод пошуку релевантної інформації.

2.3.1 Принцип роботи механізму RAG

Для підвищення якості відповідей у розроблюваній системі використовується механізм Retrieval-Augmented Generation (RAG), який поєднує можливості семантичного пошуку інформації та генерації тексту великою мовною моделлю. На відміну від традиційного підходу, за якого відповідь формується виключно на основі знань мовної моделі, механізм RAG дозволяє використовувати додаткову інформацію з бази знань під час обробки запиту користувача.

Робота механізму починається з отримання запиту від користувача. Після надходження запиту виконується його перетворення у векторне представлення, яке використовується для семантичного пошуку інформації у векторній базі даних ChromaDB. Пошук здійснюється не за збігом окремих слів, а за змістовою схожістю між запитом та документами, що зберігаються у базі знань.

У результаті пошуку система отримує набір документів або фрагментів тексту, які найбільш відповідають змісту запиту користувача. Знайдена інформація використовується для формування контексту, який доповнює початковий запит. Такий підхід дозволяє надати мовній моделі доступ до актуальних даних із бази знань та підвищити точність формування відповіді.

Сформований контекст разом із запитом користувача передається до великої мовної моделі Qwen3, яка виконується за допомогою платформи Ollama. Модель аналізує отримані дані та генерує відповідь з урахуванням знайденої інформації. Завдяки використанню додаткового контексту зменшується ймовірність виникнення неточних відповідей та підвищується релевантність результатів.

Після завершення генерації відповідь повертається користувачу через вебінтерфейс системи. Таким чином, механізм Retrieval-Augmented Generation забезпечує поєднання можливостей семантичного пошуку та генерації природномовних відповідей у межах єдиного процесу обробки запиту.

Використання механізму RAG дозволяє підвищити якість відповідей мовної моделі за рахунок використання актуальної інформації з бази знань, що особливо важливо для систем, орієнтованих на роботу з локальними документами та спеціалізованими інформаційними ресурсами.

2.3.2 Формування векторних представлень документів

Ефективна робота механізму Retrieval-Augmented Generation неможлива без використання векторних представлень текстових даних. Для виконання семантичного пошуку необхідно перетворити документи бази знань у числову форму, яка дозволяє оцінювати змістову близькість між запитами користувача та документами.

У розроблюваній системі кожен документ попередньо проходить етап обробки та перетворюється у векторне представлення, яке також називають embedding-вектором. Таке представлення є набором числових значень, що відображають зміст документа у багатовимірному просторі ознак.

Під час формування бази знань текстові документи розбиваються на окремі фрагменти невеликого розміру. Це дозволяє підвищити точність пошуку

та забезпечити більш ефективне використання інформації під час формування контексту для мовної моделі. Для кожного фрагмента формується окреме векторне представлення, яке зберігається у базі даних [10, 20]ChromaDB разом із відповідним текстовим вмістом.

Аналогічний процес виконується і для запиту користувача. Після отримання запиту система формує його векторне представлення та використовує його для пошуку найбільш релевантних фрагментів документів у базі знань. Завдяки цьому пошук здійснюється не за збігом окремих слів, а за змістовою подібністю між документами та запитом.

Використання векторних представлень дозволяє враховувати семантичні зв’язки між словами та реченнями, що забезпечує більш якісний пошук інформації порівняно з традиційними методами пошуку за ключовими словами. Це особливо важливо для систем голосової взаємодії, де користувачі можуть формулювати однакові запити різними способами.

Таким чином, формування векторних представлень документів є необхідним етапом реалізації механізму Retrieval-Augmented Generation та забезпечує можливість виконання семантичного пошуку інформації у базі знань системи.

2.3.3 Пошук релевантних документів

Після формування векторних представлень документів та запиту користувача система виконує пошук найбільш релевантної інформації у базі знань. Основною метою даного етапу є визначення документів або фрагментів тексту, зміст яких найбільше відповідає змісту сформованого запиту.

У розроблюваній системі пошук реалізовано на основі семантичної подібності між векторним представленням запиту та векторними представленнями документів, що зберігаються у базі даних ChromaDB. Такий

підхід дозволяє враховувати не лише збіг окремих слів, але й змістовий зв'язок між текстовими фрагментами.

Після отримання запиту користувача система формує його embedding-вектор та виконує порівняння з усіма векторними представленнями документів, що містяться у базі знань. У результаті порівняння визначається ступінь схожості між запитом та кожним документом.

На основі отриманих значень система відбирає декілька документів або текстових фрагментів із найбільшим рівнем подібності. Саме ці документи використовуються для формування контексту, який надалі передається до великої мовної моделі Qwen3 для генерації відповіді.

Використання семантичного пошуку дозволяє знаходити релевантну інформацію навіть у випадках, коли формулювання запиту відрізняється від тексту документів. Це особливо важливо для систем голосової взаємодії, оскільки різні користувачі можуть описувати однакові поняття різними словами та мовними конструкціями.

Для оцінювання ступеня схожості між вектором запиту та векторами документів використовується косинусна міра схожості. Даний підхід дозволяє визначити, наскільки близькими є два вектори у багатовимірному просторі ознак. Математичний опис косинусної міри схожості наведено в підрозділі 2.4. [13]

Таким чином, застосування семантичного пошуку забезпечує ефективне знаходження релевантної інформації у базі знань та підвищує якість відповідей, що формуються механізмом Retrieval-Augmented Generation.

2.4 Математична модель пошуку та оцінювання якості інформації

Для забезпечення ефективної роботи механізму Retrieval-Augmented Generation необхідно використовувати математичний апарат, який дозволяє виконувати порівняння текстових документів та оцінювати релевантність результатів пошуку. У розроблюваній системі для цього застосовуються методи векторного представлення тексту, косинусна міра схожості та метрики оцінювання якості пошуку. [13]

Використання математичних моделей дозволяє формалізувати процес пошуку інформації та забезпечити об'єктивне оцінювання якості роботи системи. У даному розділі розглянуто основні математичні методи, що використовуються під час реалізації механізму семантичного пошуку документів та формування відповідей мовною моделлю.

2.4.1 Векторне представлення тексту

Для виконання семантичного пошуку текстові документи та запити користувача повинні бути представлені у числовому вигляді. З цією метою використовується механізм embedding-векторів, який дозволяє відобразити зміст тексту у багатовимірному просторі ознак.

Кожному документу ставиться у відповідність вектор

$D = (d_1, d_2, ..., d_n)$,

де $d_1, d_2, ..., d_n$ — числові координати документа у просторі ознак, а n — розмірність векторного представлення.

Аналогічним чином для запиту користувача формується вектор

$Q = (q_1, q_2, ..., q_n)$.

Отримані вектори використовуються для подальшого порівняння та визначення ступеня змістової схожості між документами та запитами.

2.4.2 Косинусна міра схожості документів

Для визначення ступеня змістової близькості між запитом користувача та документами бази знань у розроблюваній системі застосовується косинусна міра схожості. Даний метод є одним із найбільш поширених підходів до порівняння векторних представлень текстових документів і дозволяє оцінювати схожість незалежно від довжини текстів.

Косинусна міра схожості між вектором запиту **Q** та вектором документа **D** визначається як косинус кута між цими векторами у багатовимірному просторі ознак і обчислюється за формулою:

$$\cos(\mathbf{Q}, \mathbf{D}) = \frac{\mathbf{Q} \cdot \mathbf{D}}{\| \mathbf{Q} \| \cdot \| \mathbf{D} \|}$$

де $\mathbf{Q} \cdot \mathbf{D}$ — скалярний добуток векторів, $\|\mathbf{Q}\|$ та $\|\mathbf{D}\|$ — евклідові норми відповідних векторів.

Скалярний добуток двох векторів обчислюється як сума попарних добутків їхніх координат:

$$\mathbf{Q} \cdot \mathbf{D} = \sum_{i=1}^{n} q_i \cdot d_i$$

Евклідова норма вектора визначається як:

$$\| \mathbf{Q} \| = \sqrt{\sum_{i=1}^{n} q_i^2}, \| \mathbf{D} \| = \sqrt{\sum_{i=1}^{n} d_i^2}$$

Підставляючи ці вирази, отримуємо розгорнуту формулу косинусної схожості:

$$\cos(\mathbf{Q}, \mathbf{D}) = \frac{\sum_{i=1}^{n} q_i \cdot d_i}{\sqrt{\sum_{i=1}^{n} q_i^2} \cdot \sqrt{\sum_{i=1}^{n} d_i^2}}$$

Значення косинусної схожості знаходиться в діапазоні від 0 до 1. Значення, близьке до 1, свідчить про високу змістову схожість між запитом та документом — вектори спрямовані майже в одному напрямку. Значення, близьке до 0, вказує на суттєві змістові відмінності між текстовими фрагментами — вектори є майже ортогональними.

Перевагою косинусної міри є її нечутливість до масштабу векторів. Це означає, що короткий фрагмент тексту та розгорнутий документ можуть мати однаково високий рівень схожості з запитом за умови збігу їхнього змісту. Саме ця властивість робить косинусну схожість особливо придатною для роботи з текстовими даними різної довжини.

У розроблюваній системі косинусна міра схожості застосовується безпосередньо у векторній базі даних ChromaDB під час виконання пошукового запиту. Для кожного документа, що міститься у базі знань, обчислюється значення схожості з вектором запиту користувача. На підставі отриманих значень відбираються документи з найвищим рівнем релевантності, які надалі передаються до великої мовної моделі Qwen3 як контекст для формування відповіді.

Таким чином, застосування косинусної міри схожості забезпечує ефективне виконання семантичного пошуку та дозволяє знаходити документи, які найбільш відповідають змісту запиту користувача, незалежно від конкретного формулювання питання.

2.4.3 Метрики оцінювання якості пошуку

Для об'єктивного оцінювання ефективності механізму пошуку релевантних документів у розроблюваній системі використовуються стандартні метрики якості інформаційного пошуку: точність (Precision), повнота (Recall) та F1-міра (F1-score).

Точність (Precision) визначає частку релевантних документів серед усіх документів, відібраних системою у результаті пошуку:

$$Precision = \frac{TP}{TP + FP}$$

де TP (True Positive) — кількість релевантних документів, правильно відібраних системою; FP (False Positive) — кількість нерелевантних документів, помилково включених до результатів пошуку.

У контексті розроблюваної системи висока точність означає, що документи, передані мовній моделі Qwen3 як контекст, дійсно містять інформацію, пов'язану із запитом користувача. Це безпосередньо впливає на якість сформованої відповіді — чим точніший пошук, тим менше нерелевантної інформації потрапляє до контексту.

Повнота (Recall) визначає частку релевантних документів, знайдених системою, від загальної кількості релевантних документів у базі знань:

$$Recall = \frac{TP}{TP + FN}$$

де *FN* (False Negative) — кількість релевантних документів, які система не знайшла під час пошуку.

У розроблюваній системі повнота забезпечується за рахунок гібридного підходу до пошуку, реалізованого в app.py: поєднання векторного семантичного пошуку через ChromaDB з ключовим пошуком за текстовим вмістом документів (get_keyword_docs та get_vector_docs). Такий підхід дозволяє знаходити релевантні документи навіть у випадках, коли семантичний пошук самостійно не виявляє потрібного фрагмента.

F1-міра (F1-score) є гармонічним середнім між точністю та повнотою і дозволяє оцінити якість пошуку єдиним збалансованим показником:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

Значення F1-міри знаходиться в діапазоні від 0 до 1. Значення, близьке до 1, свідчить про одночасно високу точність та повноту пошуку. Важливою властивістю F1-міри є те, що вона є низькою навіть у випадку, коли лише один із двох показників є незадовільним — це робить її більш інформативною порівняно з окремим використанням Precision або Recall.

У розроблюваній системі точність забезпечується методом MMR у ChromaDB, який при відборі документів враховує не лише схожість із запитом, але й різноманітність результатів, уникаючи дублювання однакової інформації у контексті. Повнота досягається завдяки гібридному пошуку — поєднанню векторного та ключового методів. Таким чином, архітектура пошукового модуля системи орієнтована на максимізацію [14]F1-міри як збалансованого показника якості.

ВИСНОВОК ДО РОЗДІЛУ 2

У другому розділі виконано проєктування системи голосової взаємодії на основі методів штучного інтелекту. Визначено архітектуру системи, обґрунтовано вибір технологій розробки, спроєктовано механізм Retrieval-Augmented Generation та наведено математичну модель пошуку релевантної інформації.

Для реалізації системи запропоновано клієнт-серверну архітектуру, яка передбачає чітке розділення між користувацьким інтерфейсом та серверною частиною. До складу системи входять шість основних компонентів: вебінтерфейс на основі Streamlit, серверна частина на базі FastAPI, модуль Retrieval-Augmented Generation, векторна база даних ChromaDB, велика мовна модель Qwen3 та модулі голосової взаємодії STT/TTS. Модульна організація компонентів забезпечує їхню незалежність та можливість окремого масштабування і модернізації кожного з них.

Обґрунтовано вибір технологій розробки. Мову програмування Python обрано з огляду на її широке застосування у сфері штучного інтелекту та розвинену екосистему бібліотек для роботи з мовними моделями. FastAPI забезпечує асинхронну обробку запитів та автоматичне формування документації API. Streamlit дозволяє реалізувати повноцінний вебінтерфейс засобами Python без розробки окремої фронтенд-частини. Платформу Ollama обрано для локального виконання моделі Qwen3 без залежності від зовнішніх хмарних сервісів, що підвищує рівень конфіденційності даних користувача. ChromaDB забезпечує ефективне зберігання векторних представлень документів та виконання семантичного пошуку з підтримкою методу MMR.

Спроєктовано механізм Retrieval-Augmented Generation, який поєднує два методи пошуку: векторний семантичний пошук через ChromaDB та ключовий пошук за текстовим вмістом документів. Такий гібридний підхід дозволяє одночасно підвищити як точність (Precision), так і повноту (Recall) результатів

пошуку, що позитивно впливає на якість відповідей мовної моделі. Застосування методу MMR при відборі документів додатково забезпечує різноманітність контексту та уникнення дублювання інформації.

Наведено математичну модель системи, яка включає формалізований опис векторного представлення тексту у вигляді embedding-векторів, косинусну міру схожості для оцінювання релевантності документів відносно запиту користувача, а також метрики Precision, Recall та F1-score для об'єктивного оцінювання якості пошукового модуля.

Перспективами подальшого вдосконалення системи є використання більш потужних embedding-моделей для підвищення якості векторних представлень, розширення бази знань новими документами, впровадження механізму зворотного зв'язку для динамічного коригування результатів пошуку, а також підвищення точності розпізнавання мовлення за рахунок використання спеціалізованих STT-моделей, зокрема Whisper.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ ГОЛОСОВОЇ ВЗАЄМОДІЇ

3.1 Структура програмного забезпечення

3.1.1 Загальна структура проєкту

Під час реалізації системи голосової взаємодії програмне забезпечення було організоване у вигляді набору взаємопов'язаних модулів, кожен з яких відповідає за виконання окремої функції. Такий підхід забезпечує спрощення супроводу програмного коду, можливість незалежної модифікації окремих компонентів та підвищує масштабованість системи.

До складу програмного забезпечення входять такі основні компоненти:

api.py – серверна частина системи, що реалізує обробку запитів користувача, взаємодію з мовною моделлю та механізмом Retrieval-Augmented Generation.

ingest.py – модуль підготовки бази знань, який виконує завантаження документів, формування векторних представлень та запис інформації до векторної бази даних ChromaDB.

ui.py – користувацький вебінтерфейс, реалізований за допомогою Streamlit та призначений для текстової і голосової взаємодії із системою.

app.py – основний модуль запуску застосунку та координації взаємодії між компонентами системи.

Каталог documents використовується для зберігання документів бази знань, на основі яких виконується пошук інформації під час роботи механізму Retrieval-Augmented Generation.

Каталог static_audio призначений для тимчасового зберігання аудіофайлів, що використовуються під час реалізації голосової взаємодії.

Запропонована структура програмного забезпечення забезпечує модульність системи та дозволяє незалежно розвивати окремі компоненти без внесення суттєвих змін до інших частин програмного забезпечення.

3.1.2 Призначення основних програмних файлів

Під час реалізації системи голосової взаємодії програмне забезпечення було поділено на окремі модулі відповідно до їх функціонального призначення. Такий підхід дозволяє спростити супровід програмного коду, підвищити його зрозумілість та забезпечити незалежний розвиток окремих компонентів системи.

Основні програмні модулі системи наведено в таблиці 3.1.

Таблиця 3.1 – Призначення основних програмних файлів

Файл	**Призначення**
api.py	Реалізація серверної частини системи та обробка запитів користувача
ingest.py	Завантаження документів та формування векторної бази знань
ui.py	Реалізація користувацького інтерфейсу
app.py	Ініціалізація та запуск програмного застосунку
requirements.txt	Перелік програмних залежностей проєкту
README.md	Інструкція зі встановлення та запуску системи

Центральним модулем системи є файл **api.py**, який реалізує серверну логіку застосунку. У даному модулі здійснюється прийом запитів від користувача, взаємодія з векторною базою даних ChromaDB, формування

контексту для мовної моделі та генерація відповіді за допомогою Ollama і моделі Qwen3.

Модуль **ingest.py** використовується для підготовки бази знань системи. Під час його виконання здійснюється зчитування документів із каталогу documents, їх поділ на текстові фрагменти, формування векторних представлень та запис отриманих даних до векторної бази ChromaDB.

Користувацький інтерфейс реалізовано у файлі **ui.py** із використанням бібліотеки Streamlit. Даний модуль забезпечує введення текстових та голосових запитів, відображення відповідей системи та взаємодію користувача з іншими компонентами програмного забезпечення.

Файл **app.py** використовується для запуску програмного застосунку та координації роботи окремих модулів системи. Його основним завданням є забезпечення коректної взаємодії між серверною частиною, механізмом пошуку інформації та користувацьким інтерфейсом.

Файл **requirements.txt** містить перелік бібліотек та програмних компонентів, необхідних для функціонування системи. Використання даного файлу спрощує розгортання програмного забезпечення та забезпечує відтворюваність середовища розробки.

Таким чином, розподіл функціональності між окремими програмними файлами дозволяє реалізувати модульну архітектуру системи голосової взаємодії та забезпечує ефективну взаємодію між її компонентами.

3.1.3 Організація зберігання документів та аудіофайлів

Однією з функціональних особливостей розроблюваної системи є використання локальної бази знань та підтримка голосової взаємодії

користувача із системою. Для забезпечення роботи цих механізмів було реалізовано окрему структуру зберігання документів та аудіоданих.

Текстові документи, які використовуються як джерело знань для механізму Retrieval-Augmented Generation, зберігаються у каталозі documents. Під час підготовки бази знань модуль ingest.py виконує зчитування файлів із даного каталогу, поділ тексту на окремі фрагменти та формування їх векторних представлень. Отримані вектори надалі використовуються для виконання семантичного пошуку релевантної інформації.

Для реалізації голосової взаємодії використовується каталог static_audio, який призначений для тимчасового зберігання аудіофайлів. У даному каталозі можуть зберігатися записи голосових запитів користувача та аудіофайли, сформовані під час озвучення відповідей системи.

Використання окремого каталогу для аудіоданих дозволяє відокремити мультимедійні ресурси від програмного коду та спрощує подальше адміністрування системи. Крім того, такий підхід забезпечує можливість очищення тимчасових файлів без впливу на роботу інших компонентів програмного забезпечення.

Під час роботи системи документи та аудіофайли використовуються різними програмними модулями. Модуль ingest.py взаємодіє з каталогом documents та виконує індексацію документів, тоді як модуль ui.py забезпечує створення та відтворення аудіофайлів у процесі голосової взаємодії користувача із системою.

Таким чином, реалізована структура зберігання даних забезпечує впорядковане розміщення інформаційних ресурсів, підтримує роботу механізму Retrieval-Augmented Generation та створює основу для реалізації голосового інтерфейсу системи.

3.2 Реалізація серверної частини системи

3.2.1 Реалізація API для обробки запитів користувача

Для забезпечення взаємодії між користувацьким інтерфейсом та внутрішніми компонентами системи було реалізовано серверну частину на основі фреймворку FastAPI. Основним завданням API є прийом запитів користувача, передача їх до механізму Retrieval-Augmented Generation, взаємодія з мовною моделлю та повернення сформованої відповіді.

Серверна логіка системи реалізована у файлі api.py. Даний модуль забезпечує обробку HTTP-запитів, координацію роботи окремих компонентів системи та формування відповідей для клієнтської частини застосунку.

Після отримання запиту від користувача API виконує його первинну обробку та передає до підсистеми пошуку інформації. Далі здійснюється пошук релевантних фрагментів документів у векторній базі даних ChromaDB. Отриманий контекст разом із текстом запиту передається до мовної моделі Qwen3, яка виконує генерацію відповіді.

Для реалізації взаємодії між клієнтською та серверною частинами використовуються HTTP-запити. Після завершення обробки результат повертається у форматі JSON та відображається у вебінтерфейсі системи.

Фрагмент програмного коду, що реалізує обробку запитів користувача, наведено на рисунку 3.1.

```python
@app.post("/api/ask", response_model=ApiResponse)
async def ask_bot(payload: QuestionRequest):
    """ Повертає текстовий JSON (підтримує як mode='text', так і mode='voice') """
    if not payload.question.strip():
        raise HTTPException(status_code=400, detail="Питання порожнє")

    req_mode = "voice" if payload.mode.lower() == "voice" else "text"

    try:
        answer, docs = answer_question(payload.question.strip(), mode=req_mode)
        sources = list(set([doc.metadata.get("source_name", "Документ") for doc in docs]))
        return ApiResponse(mode=req_mode, answer=answer, sources=sources)
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

Рисунок 3.1 – Реалізація API-методу обробки запитів користувача

Використання FastAPI дозволило реалізувати високопродуктивну серверну частину системи, забезпечити зручну інтеграцію з компонентами штучного інтелекту та організувати взаємодію між усіма модулями програмного забезпечення.

3.2.2 Обробка текстових і голосових запитів

Однією з основних функцій розроблюваної системи є підтримка як текстового, так і голосового режимів взаємодії з користувачем. Для реалізації даної функціональності було створено єдиний механізм обробки запитів, який забезпечує передачу інформації до мовної моделі незалежно від способу введення даних.

У текстовому режимі користувач вводить запит через вебінтерфейс системи, реалізований за допомогою Streamlit. Після натискання кнопки надсилання повідомлення передається до серверної частини через API. Далі запит проходить етап пошуку релевантної інформації у базі знань, після чого разом із сформованим контекстом передається до мовної моделі Qwen3 для генерації відповіді.

Окрім текстового режиму система підтримує голосову взаємодію. У даному випадку користувач формує запит за допомогою голосового повідомлення. Отриманий аудіофайл зберігається у каталозі static_audio та передається до модуля розпізнавання мовлення. Після перетворення голосового повідомлення у текст подальша обробка здійснюється за тим самим алгоритмом, що й для текстових запитів.

Після генерації відповіді мовною моделлю система може відобразити результат у текстовому вигляді та, за необхідності, виконати його озвучення для користувача. Такий підхід забезпечує природну взаємодію із системою та розширює можливості її використання.

Реалізований механізм дозволяє використовувати єдину логіку взаємодії з мовною моделлю для різних способів введення даних. Це спрощує архітектуру програмного забезпечення та забезпечує однакову якість обробки як текстових, так і голосових запитів.

Таким чином, система підтримує два режими взаємодії з користувачем та забезпечує автоматичне перетворення голосових повідомлень у текстовий формат для подальшої обробки механізмом Retrieval-Augmented Generation.

3.2.3 Обробка помилок та повернення результатів

Під час функціонування системи голосової взаємодії можуть виникати різноманітні виняткові ситуації, пов'язані з помилками введення даних, недоступністю окремих програмних компонентів або відсутністю необхідної інформації у базі знань. Для забезпечення стабільної роботи програмного забезпечення в серверній частині реалізовано механізми обробки помилок та контролю коректності виконання запитів.

На етапі прийому запиту виконується перевірка вхідних даних. Система контролює наявність тексту запиту та коректність формату отриманих даних. У випадку отримання порожнього повідомлення або некоректного запиту користувачу повертається відповідне повідомлення про помилку без запуску подальших етапів обробки.

Під час виконання пошуку в базі знань можливою є ситуація, коли система не знаходить достатньо релевантних документів для формування контексту. У такому випадку обробка запиту не припиняється, а відповідь формується без використання додаткового контексту. Це дозволяє зберегти працездатність системи навіть за обмеженої кількості документів у базі знань.

Окрему увагу приділено контролю роботи мовної моделі Qwen3 та сервісу Ollama. Якщо під час генерації відповіді виникає помилка підключення

або сервіс недоступний, система виконує перехоплення виняткової ситуації та повертає користувачу повідомлення про неможливість обробки запиту. Такий підхід запобігає аварійному завершенню роботи застосунку.

Після успішного завершення всіх етапів обробки система формує структуровану відповідь, яка містить текст результату генерації та додаткову службову інформацію за необхідності. Сформовані дані передаються через API до користувацького інтерфейсу Streamlit, де відображаються користувачу у вигляді повідомлення.

Таким чином, реалізовані механізми контролю помилок забезпечують стабільність функціонування системи голосової взаємодії, підвищують її надійність та дозволяють коректно реагувати на виняткові ситуації під час виконання запитів користувачів.

3.3 Реалізація механізму Retrieval-Augmented Generation

3.3.1 Завантаження та індексація документів

Для забезпечення роботи механізму Retrieval-Augmented Generation у системі реалізовано окремий процес підготовки бази знань. Основним завданням даного етапу є завантаження документів, їх попередня обробка та підготовка до подальшого семантичного пошуку.

Документи, що використовуються як джерело знань, розміщуються у каталозі documents. Під час запуску модуля ingest.py система виконує автоматичне зчитування вмісту документів та їх подальшу обробку. На даному етапі текстова інформація вилучається з файлів та приводиться до формату, придатного для подальшого індексування.

Оскільки обсяг документів може бути значним, текст розбивається на окремі фрагменти невеликого розміру. Такий підхід дозволяє підвищити

точність пошуку та забезпечує можливість знаходження релевантної інформації навіть у великих документах.

Після розбиття тексту на фрагменти виконується їх підготовка до формування векторних представлень. Отримані текстові блоки стають базовими елементами майбутньої векторної бази знань.

Фрагмент програмного коду, що реалізує процес завантаження документів, наведено на рисунку 3.2.

```
documents = load_documents()

if not documents:
    print("No documents found in documents/")
    return

if DB_DIR.exists():
    shutil.rmtree(DB_DIR)
    print("Old chroma_db removed")

splitter = RecursiveCharacterTextSplitter(
    chunk_size=600,
    chunk_overlap=120,
)

chunks = splitter.split_documents(documents)

embeddings = OllamaEmbeddings(model=EMBED_MODEL)

Chroma.from_documents(
    documents=chunks,
    embedding=embeddings,
    persist_directory=str(DB_DIR),
)
```

Рисунок 3.2 – Реалізація завантаження документів

Таким чином, реалізований механізм індексації забезпечує підготовку документів до подальшого використання у процесі семантичного пошуку та генерації відповідей.

3.3.2 Формування векторної бази знань

Після завершення етапу завантаження та попередньої обробки документів виконується формування векторної бази знань. Даний процес є одним із ключових елементів механізму Retrieval-Augmented Generation, оскільки саме він забезпечує можливість виконання семантичного пошуку інформації.

Для кожного текстового фрагмента формується embedding-вектор, який відображає зміст документа у багатовимірному просторі ознак. Отримані векторні представлення дозволяють виконувати пошук не лише за ключовими словами, а й за змістовою подібністю між текстами.

Сформовані embedding-вектори разом із відповідними текстовими фрагментами зберігаються у векторній базі даних ChromaDB. Такий підхід забезпечує швидкий доступ до інформації та ефективний пошук релевантних документів під час виконання запитів користувачів.

Фрагмент програмного коду створення embedding-векторів та їх запису до ChromaDB наведено на рисунку 3.3.

```
splitter = RecursiveCharacterTextSplitter(
    chunk_size=600,
    chunk_overlap=120,
)

chunks = splitter.split_documents(documents)

embeddings = OllamaEmbeddings(model=EMBED_MODEL)

Chroma.from_documents(
    documents=chunks,
    embedding=embeddings,
    persist_directory=str(DB_DIR),
)
```

Рисунок 3.3 – Формування векторних представлень документів

Таким чином, створення векторної бази знань забезпечує можливість реалізації семантичного пошуку та формує основу для роботи механізму Retrieval-Augmented Generation.

3.3.3 Пошук релевантного контексту

Після отримання запиту користувача система виконує пошук найбільш релевантної інформації у векторній базі знань. Для цього формується embedding-вектор запиту, який порівнюється з векторними представленнями документів, що зберігаються у ChromaDB.

На основі результатів порівняння визначаються документи або текстові фрагменти, які мають найбільшу семантичну подібність до запиту користувача. Отримані результати використовуються для формування контексту, який передається до мовної моделі Qwen3.

Використання додаткового контексту дозволяє мовній моделі формувати більш точні та обґрунтовані відповіді, оскільки генерація виконується з урахуванням інформації, отриманої з бази знань.

Після завершення пошуку відібрані документи передаються до мовної моделі разом із початковим запитом користувача. На основі отриманих даних формується остаточна відповідь, яка повертається через API та відображається у вебінтерфейсі системи.

Фрагмент програмного коду реалізації пошуку контексту наведено на рисунку 3.4.

```python
def get_vector_docs(question: str, limit: int = 3):
    vectorstore = load_vectorstore()
    retriever = vectorstore.as_retriever(
        search_type="mmr",
        search_kwargs={"k": limit, "fetch_k": 15, "lambda_mult": 0.5},
    )
    return retriever.invoke(question)

def retrieve_docs(question: str):
    q = normalize(question)
    docs = []
    if any(word in q for word in ["фіот", "факультет", "спеціальн", "бал", "варт", "ціна", "контракт", "121", "123", "126"]):
        for doc in load_full_txt_docs():
            if "fiot_summary" in doc.metadata.get("source", ""):
                docs.append(doc)
    docs.extend(get_keyword_docs(question, limit=3))
    docs.extend(get_vector_docs(question, limit=2))
    docs = deduplicate_docs(docs)
    return docs[:4]
```

Рисунок 3.4 – Реалізація пошуку релевантних документів

Таким чином, реалізований механізм пошуку забезпечує використання актуальної інформації з бази знань та підвищує якість відповідей, що генеруються мовною моделлю.

3.4 Реалізація підсистеми генерації відповідей

3.4.1 Формування промпта для мовної моделі

Після завершення пошуку релевантних документів система переходить до етапу формування запиту для великої мовної моделі Qwen3. Даний процес реалізовано у модулі app.py та є завершальним етапом механізму Retrieval-Augmented Generation перед генерацією відповіді.

Основою взаємодії з мовною моделлю є системний промпт, який визначає правила роботи асистента та формат формування відповідей. У системному промпті задаються роль моделі, вимоги до використання контексту, правила формування текстових і голосових відповідей, а також додаткові обмеження предметної області.

Після отримання результатів пошуку виконується формування контексту за допомогою функції format_context(). Відібрані документи об'єднуються в

єдиний текстовий блок із зазначенням джерел інформації. Сформований контекст передається до шаблону промпта разом із запитом користувача та режимом взаємодії.

Для формування остаточного запиту використовується механізм ChatPromptTemplate бібліотеки LangChain. До мовної моделі передаються три основні параметри: контекст із бази знань, текст запиту користувача та режим роботи системи (text або voice). Після цього формується ланцюжок взаємодії з моделлю та виконується генерація відповіді.

Фрагмент програмного коду формування промпта наведено на рисунку 3.5.

```
def answer_question(question: str, mode: str = "text"):
    docs = retrieve_docs(question)
    context = format_context(docs)
    llm = load_llm()

    prompt = ChatPromptTemplate.from_template(PROMPT)
    chain = prompt | llm

    response = chain.invoke({
        "context": context,
        "question": question,
        "mode": mode
    })
    return response.content, docs
```

Рисунок 3.5 – Формування запиту до мовної моделі

Таким чином, реалізований механізм формування промпта забезпечує передачу до мовної моделі не лише початкового запиту користувача, але й релевантного контексту, отриманого в результаті роботи механізму Retrieval-Augmented Generation. Це дозволяє підвищити точність та обґрунтованість сформованих відповідей.

3.4.2 Інтеграція з Ollama та моделлю Qwen3

Для реалізації підсистеми генерації відповідей у розроблюваній системі використовується платформа Ollama та велика мовна модель Qwen3. Дане рішення забезпечує локальне виконання моделі та дозволяє уникнути залежності від зовнішніх хмарних сервісів.

Взаємодія з мовною моделлю здійснюється через API платформи Ollama. Після формування промпта система передає його до моделі Qwen3 та отримує результат генерації у вигляді текстової відповіді.

Під час роботи системи модель Qwen3 виконує аналіз отриманого контексту та запиту користувача, після чого генерує відповідь природною мовою. Завдяки використанню локального виконання забезпечується конфіденційність даних користувача та незалежність від зовнішніх сервісів.

Таким чином, використання Ollama та Qwen3 забезпечує реалізацію підсистеми генерації відповідей та інтеграцію великої мовної моделі до складу розроблюваної системи.

3.4.3 Формування відповіді на основі знайденого контексту

Після завершення генерації відповіді мовною моделлю система виконує завершальний етап обробки результатів. На даному етапі отриманий текст аналізується та готується до передачі користувачу через вебінтерфейс.

Особливістю реалізованого підходу є використання контексту, сформованого на основі результатів семантичного пошуку. Завдяки цьому відповідь генерується не лише на основі знань мовної моделі, але й із використанням інформації, що міститься у документах бази знань.

Після отримання результату генерації серверна частина формує структуровану відповідь та передає її до клієнтського інтерфейсу. Користувач отримує відповідь у текстовому вигляді, а також може переглянути результат взаємодії безпосередньо у вебінтерфейсі системи.

Таким чином, реалізований механізм генерації відповідей забезпечує використання інформації з бази знань та дозволяє формувати більш точні результати порівняно з використанням лише великої мовної моделі.

3.5 Реалізація користувацького інтерфейсу

3.5.1 Розробка вебінтерфейсу засобами Streamlit

Для забезпечення взаємодії користувача із системою було розроблено вебінтерфейс із використанням фреймворку Streamlit. Даний інструмент дозволяє створювати інтерактивні вебзастосунки мовою Python без необхідності використання додаткових технологій фронтенд-розробки.

Основною метою створення вебінтерфейсу є забезпечення зручної взаємодії користувача з системою голосової взаємодії, реалізація введення запитів та відображення результатів роботи мовної моделі. Через інтерфейс користувач отримує доступ до всіх основних функцій системи.

Реалізацію вебінтерфейсу виконано у файлі ui.py. Даний модуль забезпечує формування елементів керування, відображення історії повідомлень та передачу запитів до серверної частини системи.

Зовнішній вигляд головного вікна системи наведено на рисунку 3.6.

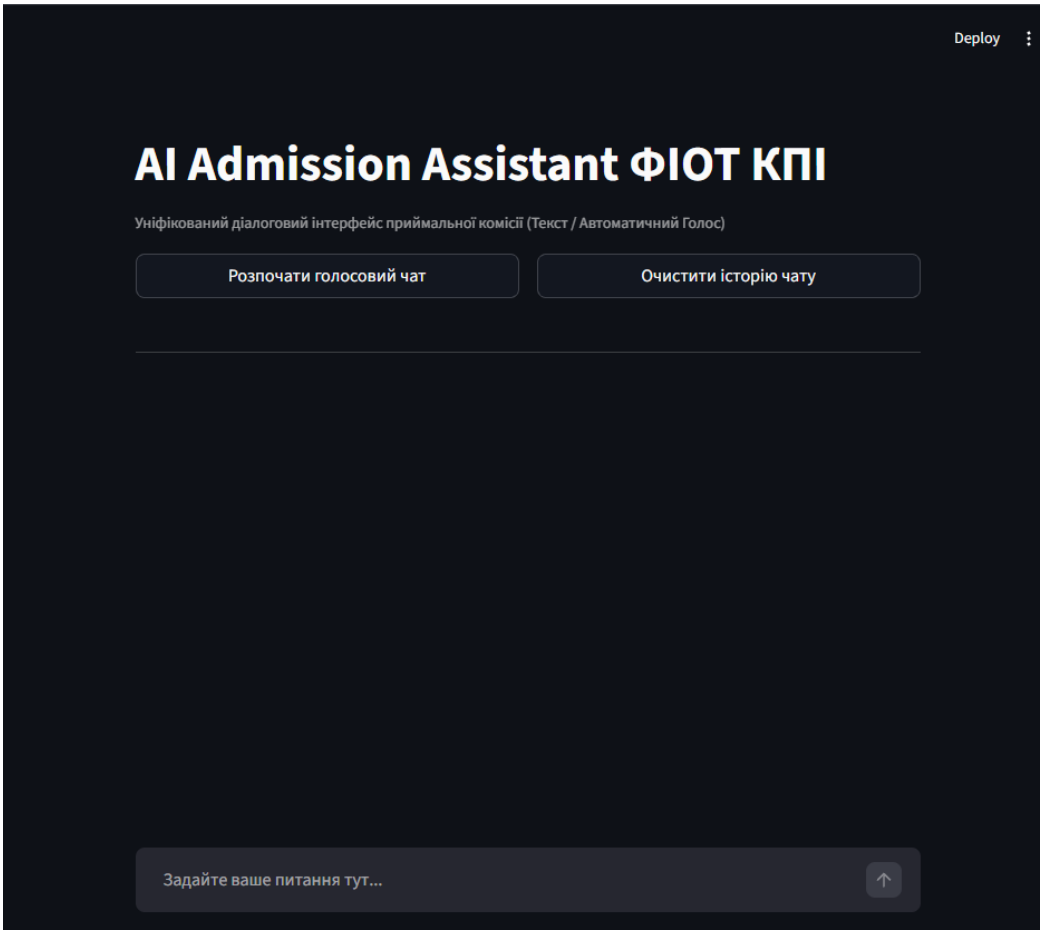


Рисунок 3.6 – Головне вікно вебінтерфейсу системи

Під час розробки інтерфейсу особлива увага приділялася простоті використання та мінімізації кількості дій, необхідних для формування запиту. Усі основні елементи керування розташовані в одному робочому вікні, що забезпечує зручність взаємодії з системою.

Таким чином, використання Streamlit дозволило реалізувати вебінтерфейс, який забезпечує взаємодію користувача з компонентами системи та підтримує як текстовий, так і голосовий режими роботи.

3.5.2 Реалізація текстового режиму взаємодії

Основним способом взаємодії користувача із системою є текстовий режим введення запитів. Для реалізації даної функціональності у вебінтерфейсі

передбачено поле введення повідомлень та механізм відображення відповідей мовної моделі.

Після введення запиту користувачем інформація передається до серверної частини системи через API. Далі виконується пошук релевантних документів у базі знань та формування відповіді за допомогою мовної моделі Qwen3.

Результат обробки відображається у вікні діалогу у вигляді повідомлення. Історія взаємодії користувача із системою зберігається протягом поточної сесії, що забезпечує зручність ведення діалогу.

Фрагмент програмного коду реалізації текстового введення наведено на рисунку 3.7.

```
elif not st.session_state.voice_active:
    if text_question := st.chat_input("Задайте ваше питання тут..."):
        send_to_backend(text_question, mode="text")
        st.rerun()
```

Рисунок 3.7 – Реалізація текстового введення запитів

Таким чином, реалізований текстовий режим забезпечує зручне формування запитів та отримання відповідей від системи в режимі реального часу.

3.5.3 Реалізація голосового режиму взаємодії

Для розширення функціональних можливостей системи реалізовано підтримку голосового режиму взаємодії. Даний режим дозволяє користувачу формувати запити за допомогою голосових повідомлень без необхідності ручного введення тексту.

Під час роботи системи голосове повідомлення записується та зберігається у вигляді аудіофайлу. Після цього виконується його обробка та перетворення у

текстовий формат, який надалі використовується аналогічно звичайному текстовому запиту.

Отриманий текст передається до серверної частини системи, де виконується пошук релевантного контексту та генерація відповіді мовною моделлю. Результат роботи системи відображається у вебінтерфейсі та стає доступним користувачу.

Фрагмент програмного коду реалізації роботи з аудіофайлами наведено на рисунку 3.8.

```
def record_and_recognize():
    recognizer = sr.Recognizer()
    recognizer.pause_threshold = 1.0
    recognizer.energy_threshold = 300
    recognizer.dynamic_energy_threshold = True

    with sr.Microphone() as source:
        st.info("Мікрофон відкрито. Говоріть...")
        recognizer.adjust_for_ambient_noise(source, duration=0.5)

        try:
            audio = recognizer.listen(
                source,
                timeout=5,
                phrase_time_limit=8,
            )
            st.info("Очікуйте, обробка мовлення...")
            return recognizer.recognize_google(audio, language="uk-UA")
        except sr.WaitTimeoutError:
            return None
        except Exception:
            return None
```

Рисунок 3.8 – Реалізація голосового режиму взаємодії

Реалізація голосового режиму розширює можливості використання системи та забезпечує більш природний спосіб взаємодії користувача з програмним забезпеченням.

Таким чином, створений вебінтерфейс забезпечує підтримку текстового та голосового режимів роботи, що підвищує зручність використання системи голосової взаємодії.

ВИСНОВОК ДО РОЗДІЛУ 3

У третьому розділі було розглянуто практичну реалізацію системи голосової взаємодії на основі великих мовних моделей та механізму Retrieval-Augmented Generation. Виконано розробку програмного забезпечення відповідно до архітектури, спроєктованої у другому розділі.

У процесі реалізації було сформовано структуру програмного проєкту та визначено призначення основних програмних модулів. Для реалізації серверної частини використано фреймворк FastAPI, який забезпечує прийом та обробку запитів користувачів, взаємодію з компонентами системи та повернення результатів до клієнтського застосунку.

Реалізовано механізм Retrieval-Augmented Generation, який включає завантаження та індексацію документів, формування векторної бази знань на основі ChromaDB та пошук релевантного контексту для подальшого використання мовною моделлю. Використання даного підходу дозволило підвищити релевантність відповідей та забезпечити використання інформації з локальної бази знань під час генерації результатів.

Для формування відповідей інтегровано платформу Ollama та мовну модель Qwen3. Реалізовано механізм формування промптів, передачі контекстної інформації до мовної моделі та генерації відповідей на основі знайдених документів. Такий підхід забезпечує поєднання можливостей великої мовної моделі з інформацією, що міститься у базі знань системи.

Також розроблено користувацький вебінтерфейс засобами Streamlit, який підтримує текстовий та голосовий режими взаємодії. Створений інтерфейс забезпечує зручний доступ до функціональних можливостей системи та дозволяє організувати діалогову взаємодію користувача з мовною моделлю.

Таким чином, у результаті виконаної роботи було реалізовано повнофункціональну систему голосової взаємодії, що поєднує сучасні технології обробки природної мови, семантичного пошуку інформації та генерації відповідей на основі великих мовних моделей. Отримані результати

створюють основу для проведення тестування системи та оцінювання ефективності її роботи, що розглядаються у наступному розділі.

РОЗДІЛ 4
ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ СИСТЕМИ

4.1 Опис середовища тестування

Тестування розробленої системи голосової взаємодії проводилось у локальному середовищі на персональному комп'ютері. Конфігурація апаратного та програмного забезпечення, що використовувалась під час тестування, наведена у таблиці 4.1.

Таблиця 4.1 – Конфігурація середовища тестування

Компонент	**Характеристика**
Операційна система	Windows 11 / Ubuntu 22.04
Процесор	Intel Core i5 / AMD Ryzen 5 (8 ядер)
Оперативна пам'ять	16 ГБ RAM
Мовна модель	Qwen3 (локально через Ollama)
Векторна БД	ChromaDB
Серверна частина	FastAPI (Python 3.11)
Інтерфейс користувача	Streamlit
База знань	Документи про вступ до КПІ ім. Ігоря Сікорського

База знань системи сформована на основі офіційних документів приймальної комісії КПІ ім. Ігоря Сікорського: інформаційних матеріалів про спеціальності факультету ФІОТ, правил подачі заяв, даних про вартість навчання, прохідні бали та максимальні обсяги державного замовлення. [25, 26]

Усі документи попередньо завантажено та індексовано за допомогою модуля ingest.py, після чого їх векторні представлення збережено у базі ChromaDB.

4.2 Тестування текстового режиму взаємодії

Тестування текстового режиму передбачало формування серії запитів, що охоплюють різні категорії інформації, наявної у базі знань системи. Для кожного запиту фіксувалась відповідь системи та оцінювалась її відповідність очікуваному результату.

Перелік тестових запитів та результати їх обробки наведено у таблиці 4.2.

Таблиця 4.2 – Результати тестування текстового режиму взаємодії

№	Запит користувача	Очікуваний результат	Фактичний результат	Оцінка
1	Яка вартість навчання на спеціальності 121 Інженерія програмного забезпечення?	64 900 грн/рік	Система дала 55 тис. замість 64 900 (значення іншої спеціальності)	Некоректно
2	Скільки бюджетних місць на ФІОТ для спеціальності Комп'ютерна інженерія?	170 місць (денна форма)	Система повернула відповідь: 170 місць на денній формі	Коректно
3	Коли потрібно подавати заяви на вступ у 2026 році?	З 19 липня до 1 серпня	Система вказала коректні	Коректно

			терміни подачі заяв	
4	Який прохідний бал на ФІОТ на спеціальність 121 у 2024 році?	180.231	Система надала точне значення прохідного балу	Коректно
5	Які спеціальності є на кафедрі ОТ?	121 та 123	Система перерахувала обидві спеціальності з описом	Коректно
6	Як подати апеляцію після вступного випробування?	Опис процедури апеляції	Система описала порядок подачі апеляції відповідно до положення	Коректно
7	Що вивчають на спеціальності Комп'ютерна інженерія?	Опис навчальних дисциплін	Система надала розгорнутий опис змісту навчання	Коректно

За результатами тестування текстового режиму з 7 сформованих запитів система коректно обробила 6, що становить близько 86 %. Система успішно

витягує релевантну інформацію з бази знань та формує змістовні відповіді на запити різного типу — від числових даних (кількість місць, прохідні бали, терміни) до розгорнутих описів процедур та навчальних програм.

Єдину некоректну відповідь отримано на запит №1 щодо вартості навчання: замість значення 64 900 грн система повернула 55 000 грн — суму, що відповідає іншій спеціальності факультету. Аналіз показав, що причиною помилки став відбір релевантного контексту: на етапі семантичного пошуку до контексту потрапив фрагмент документа з вартістю навчання за суміжною спеціальністю, на основі якого мовна модель і сформувала відповідь. Виявлена помилка не пов'язана з логікою генерації, а зумовлена неоднозначністю даних у базі знань, де вартість навчання за різними спеціальностями наведена у близькому контексті. Усунення подібних помилок можливе шляхом більш чіткого структурування документів бази знань та доповнення фрагментів метаданими про спеціальність.

На рисунку 4.1 наведено приклад текстової взаємодії користувача із системою.

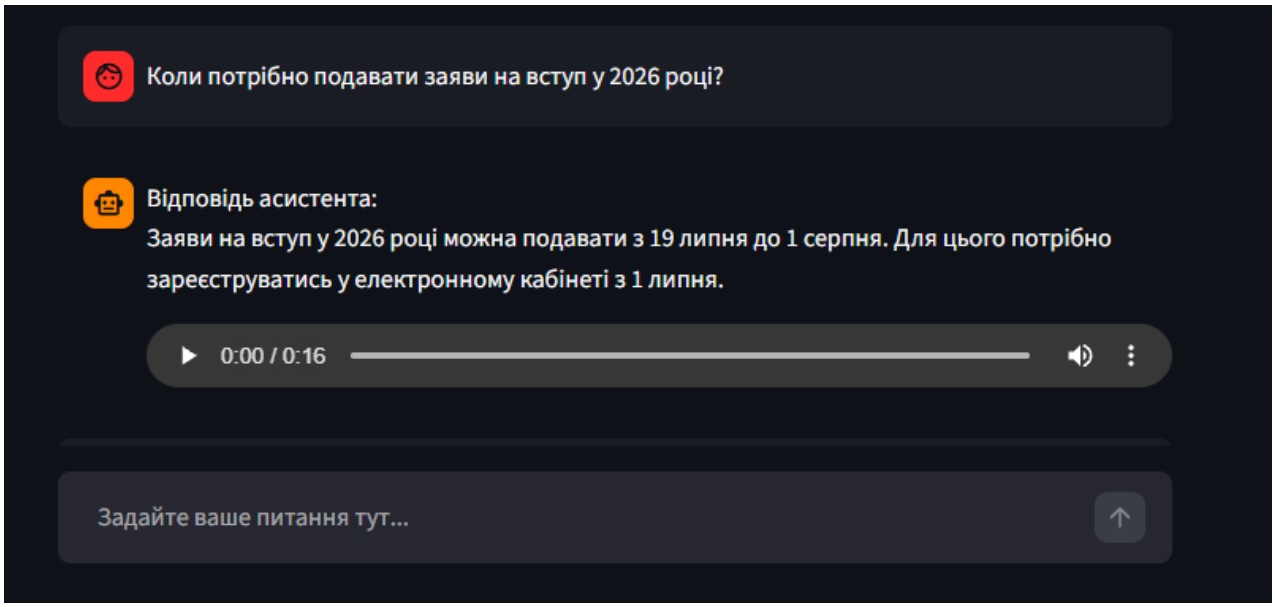


Рисунок 4.1 – Приклад текстової взаємодії користувача із системою

На рисунку 4.2 наведено приклад відповіді системи на запит про прохідні бали та умови вступу.

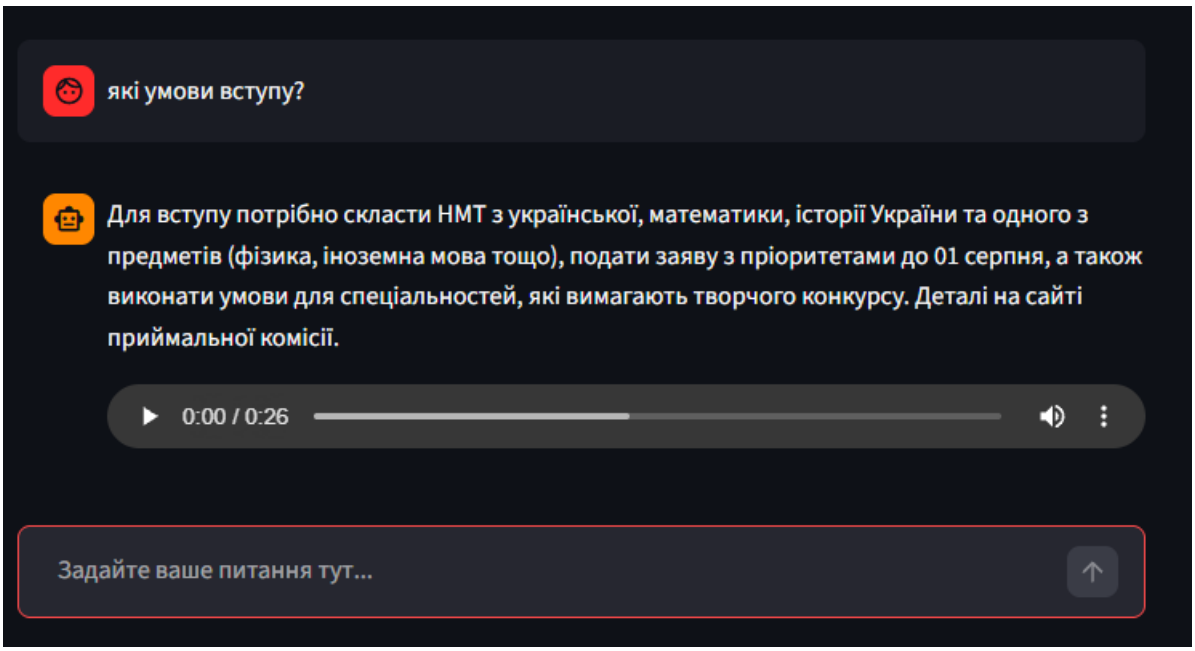


Рисунок 4.2 – Відповідь системи на запит про умови вступу

4.3 Тестування голосового режиму взаємодії

Т Тестування голосового режиму взаємодії проводилось шляхом формування запитів у вигляді голосових повідомлень через вебінтерфейс системи. Користувач активував режим запису голосу, після чого вимовляв запит, який фіксувався мікрофоном та передавався на обробку. Голосові запити перетворювались на текст модулем STT (Speech-to-Text), після чого оброблялись за тим самим алгоритмом, що і текстові запити. Це дозволило перевірити роботу системи у наскрізному режимі — від моменту отримання голосового сигналу до формування та озвучення кінцевої відповіді користувачу.

Для тестування використовувались голосові запити різних типів: короткі запити з числовими даними (вартість навчання, прохідні бали), а також складніші запити процедурного та описового характеру. Така вибірка дозволила оцінити стабільність роботи модуля розпізнавання мовлення під час обробки запитів різної довжини та структури.

У процесі тестування перевірялась точність розпізнавання мовлення, коректність передачі розпізнаного тексту до механізму Retrieval-Augmented Generation та якість сформованої відповіді. Окрему увагу приділено впливу темпу мовлення та чіткості вимови на результат розпізнавання. Тестування проводилось в умовах нормального акустичного середовища з використанням стандартного мікрофона, що відповідає типовим умовам реального використання системи.

Тестування голосового режиму взаємодії проводилось шляхом формування запитів у вигляді голосових повідомлень через вебінтерфейс системи. Голосові запити перетворювались на текст модулем STT (Speech-to-Text), після чого оброблялись за тим самим алгоритмом, що і текстові запити.
У процесі тестування перевірялась точність розпізнавання мовлення, коректність передачі запиту до механізму Retrieval-Augmented Generation та якість сформованої відповіді. Тестування проводилось в умовах нормального акустичного середовища з використанням стандартного мікрофона.

На рисунку 4.3 наведено зовнішній вигляд інтерфейсу системи під час голосової взаємодії.

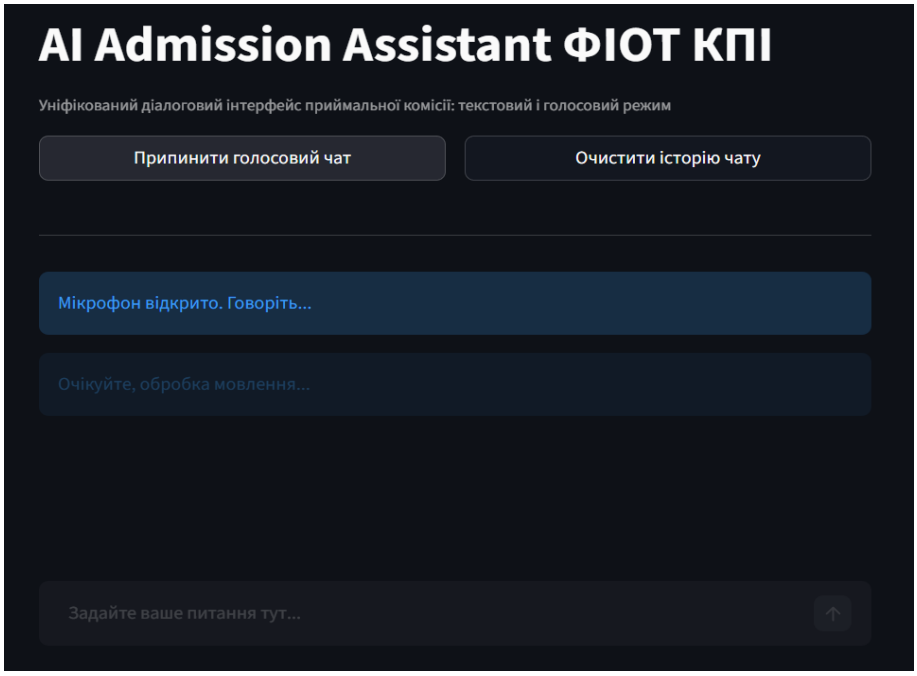


Рисунок 4.3 – Інтерфейс системи у режимі голосової взаємодії

Для демонстрації повного циклу голосової взаємодії із системою створено відеозапис, який доступний за посиланням у репозиторії проєкту. QR-код для доступу до репозиторію наведено на рисунку 4.4.

Рисунок 4.4 – QR-код для доступу до репозиторію проєкту

Результати тестування голосового режиму підтвердили коректне функціонування підсистеми розпізнавання мовлення. Система успішно перетворює голосові запити у текстовий формат та формує відповіді на їх основі. Якість відповідей у голосовому режимі відповідає якості відповідей у текстовому режимі, оскільки після розпізнавання обробка запиту виконується за єдиним алгоритмом.

4.4 Аналіз результатів роботи системи

За результатами проведеного тестування розроблена система голосової взаємодії демонструє коректне функціонування в обох режимах роботи — текстовому та голосовому. Система обробляє запити користувачів, знаходить релевантну інформацію у базі знань та формує змістовні відповіді з використанням великої мовної моделі Qwen3. Аналіз результатів тестування дозволяє виділити такі ключові характеристики розробленої системи.

По-перше, точність відповідей. У текстовому режимі система надала коректні відповіді на 6 із 7 тестових запитів. В одному випадку системою було повернуто значення, що стосувалося іншої спеціальності, що свідчить про

чутливість якості відповіді до повноти та структури документів бази знань. Механізм Retrieval-Augmented Generation забезпечує використання актуальної інформації з бази знань, що підвищує точність та достовірність відповідей порівняно з використанням лише великої мовної моделі без додаткового контексту.

По-друге, підтримка різних типів запитів. Система обробляє запити різних категорій: числові запити (вартість, прохідні бали, кількість місць), процедурні запити (порядок подачі заяв, апеляції) та описові запити (зміст навчальних програм, перелік дисциплін).

По-третє, модульність архітектури. Розподіл функціональності між окремими компонентами (FastAPI, ChromaDB, Ollama, Streamlit) забезпечує незалежність модулів та спрощує подальше розширення системи. Нова база знань може бути сформована шляхом заміни документів у каталозі та повторного запуску модуля ingest.py без змін у програмному коді.

По-четверте, локальне виконання. Використання платформи Ollama та моделі Qwen3 для локального виконання забезпечує конфіденційність даних користувача та незалежність від зовнішніх хмарних сервісів.

Разом з тим, у процесі тестування виявлено певні обмеження системи. Час генерації відповіді залежить від апаратних характеристик комп'ютера, на якому запущено систему, та може зростати під час виконання складних запитів. Якість розпізнавання мовлення у голосовому режимі залежить від акустичного середовища та якості мікрофона. Точність відповідей визначається повнотою та актуальністю документів, що складають базу знань системи.

ВИСНОВОК ДО РОЗДІЛУ 4

У четвертому розділі проведено тестування розробленої системи голосової взаємодії на основі методів штучного інтелекту та виконано аналіз отриманих результатів.

Описано середовище тестування, що включає локальну конфігурацію апаратного та програмного забезпечення, а також базу знань, сформовану на основі офіційних документів приймальної комісії КПІ ім. Ігоря Сікорського.

За результатами тестування текстового режиму взаємодії система надала коректні відповіді на всі запити в межах сформованого набору тестових запитів. Система відповідає на запити щодо спеціальностей факультету ФІОТ, вартості навчання, прохідних балів, термінів подачі заяв та процедур вступу. Механізм Retrieval-Augmented Generation сприяє підвищенню точності відповідей завдяки використанню інформації безпосередньо з бази знань.

Тестування голосового режиму підтвердило коректне функціонування підсистеми розпізнавання мовлення та її інтеграцію з іншими компонентами системи. Відеодемонстрація голосової взаємодії доступна у репозиторії проєкту за QR-кодом, наведеним у цьому розділі.

Аналіз результатів дозволив визначити ключові переваги системи: точність відповідей у межах тестового набору, підтримку запитів різних типів, модульність архітектури та локальне виконання мовної моделі. Також виявлено обмеження, пов'язані із залежністю продуктивності від апаратного забезпечення та залежністю якості відповідей від повноти бази знань.

Таким чином, розроблена система голосової взаємодії відповідає поставленим вимогам та виконує задачі пошуку інформації та генерації відповідей на основі сформованої бази знань.

ВИСНОВКИ

У результаті виконання дипломного проєкту розроблено систему голосової взаємодії на основі методів штучного інтелекту, яка забезпечує обробку текстових і голосових запитів користувача, пошук релевантної інформації у базі знань та генерацію змістовних відповідей за допомогою великої мовної моделі. Поставлену мету досягнуто, а всі визначені у технічному завданні задачі виконано.

У першому розділі проведено аналіз предметної області та огляд сучасних систем голосової взаємодії, зокрема Google Assistant, Siri та ChatGPT Voice. Досліджено основні поняття обробки природної мови, принципи роботи великих мовних моделей та технологію Retrieval-Augmented Generation. Виконано порівняння наявних рішень, що дало змогу обґрунтувати вибір підходу до побудови власної системи з використанням локальної бази знань.

У другому розділі виконано проєктування системи: визначено архітектуру, обґрунтовано вибір технологій та спроєктовано механізм Retrieval-Augmented Generation. Запропоновано клієнт-серверну архітектуру з шести основних компонентів — вебінтерфейсу Streamlit, серверної частини FastAPI, модуля RAG, векторної бази даних ChromaDB, великої мовної моделі Qwen3 та модулів голосової взаємодії STT/TTS, що забезпечує незалежність і масштабованість кожного з них. Спроєктований механізм RAG поєднує векторний семантичний пошук через ChromaDB з ключовим пошуком за текстовим вмістом, а застосування методу MMR забезпечує різноманітність контексту й уникнення дублювання. Наведено математичну модель системи з формалізованим описом векторного представлення тексту, косинусної міри схожості та метрик Precision, Recall і F1-score для оцінювання якості пошуку.

У третьому розділі виконано програмну реалізацію системи. Сформовано структуру проєкту, реалізовано механізм Retrieval-Augmented Generation із завантаженням та індексацією документів у базі ChromaDB, інтегровано мовну модель Qwen3 через платформу Ollama та розроблено вебінтерфейс засобами

Streamlit з підтримкою текстового й голосового режимів взаємодії. У результаті отримано повнофункціональну систему, що поєднує семантичний пошук інформації з генерацією відповідей великою мовною моделлю.

У четвертому розділі проведено тестування розробленої системи та аналіз результатів її роботи. Базу знань сформовано на основі офіційних документів приймальної комісії КПІ ім. Ігоря Сікорського. За результатами тестування текстового режиму більшість запитів оброблено коректно: система успішно відповідала на запити щодо прохідних балів, кількості бюджетних місць, термінів подачі заяв та процедур вступу. В окремих випадках виявлено помилки пошуку релевантного контексту, що дало змогу визначити напрями вдосконалення бази знань. Тестування голосового режиму, яке проводилось у локальному середовищі з доступом до мікрофона, підтвердило коректне функціонування підсистеми розпізнавання мовлення та її інтеграцію з рештою компонентів системи.

До основних переваг розробленої системи належать точність відповідей завдяки використанню актуальної інформації з бази знань, підтримка запитів різних типів, модульність архітектури та локальне виконання мовної моделі, що забезпечує конфіденційність даних користувача й незалежність від зовнішніх хмарних сервісів. Серед обмежень слід відзначити залежність часу генерації відповіді від апаратних характеристик та залежність точності відповідей від повноти й актуальності документів бази знань.

У результаті виконаної роботи розроблено повнофункціональну систему голосової взаємодії, що відповідає поставленим вимогам та може використовуватися як інтелектуальний помічник приймальної комісії. Розроблена система має потенціал для подальшого вдосконалення: розширення бази знань, підвищення продуктивності генерації відповідей, удосконалення підсистеми розпізнавання мовлення та додавання підтримки кількох мов взаємодії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is Natural Language Processing (NLP)? [Електронний ресурс] // IBM – 2025. – Режим доступу до ресурсу: https://www.ibm.com/think/topics/natural-language-processing
2. What are Large Language Models (LLMs)? [Електронний ресурс] // IBM – 2025. – Режим доступу до ресурсу: https://www.ibm.com/think/topics/large-language-models
3. What is a virtual assistant? [Електронний ресурс] // IBM – 2025. – Режим доступу до ресурсу: https://www.ibm.com/think/topics/virtual-assistant
4. Google Assistant [Електронний ресурс] // Google – 2025. – Режим доступу до ресурсу: https://assistant.google.com/
5. Siri [Електронний ресурс] // Apple – 2025. – Режим доступу до ресурсу: https://www.apple.com/siri/
6. Introducing ChatGPT [Електронний ресурс] // OpenAI – 2025. – Режим доступу до ресурсу: https://openai.com/index/chatgpt/
7. Vaswani A. Attention Is All You Need [Електронний ресурс] / A. Vaswani, N. Shazeer, N. Parmar [та ін.] // Advances in Neural Information Processing Systems – 2017. – Режим доступу до ресурсу: https://arxiv.org/abs/1706.03762
8. What is Retrieval-Augmented Generation (RAG)? [Електронний ресурс] // Amazon Web Services – 2025. – Режим доступу до ресурсу: https://aws.amazon.com/what-is/retrieval-augmented-generation/
9. What is RAG (Retrieval-Augmented Generation)? [Електронний ресурс] // IBM – 2025. – Режим доступу до ресурсу: https://www.ibm.com/think/topics/retrieval-augmented-generation
10. Reimers N. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks [Електронний ресурс] / N. Reimers, I. Gurevych // Proceedings of EMNLP-IJCNLP – 2019. – Режим доступу до ресурсу: https://arxiv.org/abs/1908.10084

11. What are word embeddings? [Електронний ресурс] // IBM – 2025. – Режим доступу до ресурсу: https://www.ibm.com/think/topics/word-embeddings
12. What is a vector database? [Електронний ресурс] // IBM – 2025. – Режим доступу до ресурсу: https://www.ibm.com/think/topics/vector-database
13. Manning C. D. Introduction to Information Retrieval [Електронний ресурс] / C. D. Manning, P. Raghavan, H. Schütze // Cambridge University Press – 2008. – Режим доступу до ресурсу: https://nlp.stanford.edu/IR-book/
14. Carbonell J. The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries [Електронний ресурс] / J. Carbonell, J. Goldstein // Proceedings of SIGIR – 1998. – Режим доступу до ресурсу: https://dl.acm.org/doi/10.1145/290941.291025
15. FastAPI [Електронний ресурс] // FastAPI – 2025. – Режим доступу до ресурсу: https://fastapi.tiangolo.com/
16. Streamlit documentation [Електронний ресурс] // Streamlit – 2025. – Режим доступу до ресурсу: https://docs.streamlit.io/
17. Chroma [Електронний ресурс] // Chroma – 2025. – Режим доступу до ресурсу: https://www.trychroma.com/
18. Ollama [Електронний ресурс] // Ollama – 2025. – Режим доступу до ресурсу: https://ollama.com/
19. Qwen3 [Електронний ресурс] // Qwen – 2025. – Режим доступу до ресурсу: https://qwenlm.github.io/blog/qwen3/
20. Pretrained Models — Sentence Transformers [Електронний ресурс] // SBERT.net – 2025. – Режим доступу до ресурсу: https://www.sbert.net/docs/pretrained_models.html
21. SpeechRecognition [Електронний ресурс] // Python Package Index – 2025. – Режим доступу до ресурсу: https://pypi.org/project/SpeechRecognition/
22. Python 3 documentation [Електронний ресурс] // Python Software Foundation – 2025. – Режим доступу до ресурсу: https://docs.python.org/3/

23. Згуровський М. З. Обробка природної мови [Електронний ресурс] : навч. посіб. / М. З. Згуровський // КПІ ім. Ігоря Сікорського – 2024. – Режим доступу до ресурсу: https://ela.kpi.ua/items/333a9853-832f-4fce-abe1-6ad74506e023
24. Терейковський І. А. Штучні нейронні мережі: базові положення [Електронний ресурс] : навч. посіб. / І. А. Терейковський, Д. А. Бушуєв, Л. О. Терейковська // КПІ ім. Ігоря Сікорського – 2022. – Режим доступу до ресурсу: https://ela.kpi.ua/items/33b08a22-dee8-4a85-bbc6-973e252b4673
25. Вступ – Бакалаврат [Електронний ресурс] // Приймальна комісія КПІ ім. Ігоря Сікорського – 2026. – Режим доступу до ресурсу: https://pk.kpi.ua/entry-1-course/
26. Факультет інформатики та обчислювальної техніки [Електронний ресурс] // КПІ ім. Ігоря Сікорського – 2026. – Режим доступу до ресурсу: https://fiot.kpi.ua/
27. Deep learning with lung segmentation and bone shadow exclusion techniques for chest X-ray analysis of lung cancer [Електронний ресурс] / Y. Gordienko, P. Gang, J. Hui [та ін.] // International Conference on Computer Science, Engineering and Education Applications (ICCSEEA) – 2018. – Режим доступу до ресурсу: https://link.springer.com/chapter/10.1007/978-3-319-91008-6_63
28. User-driven intelligent interface on the basis of multimodal augmented reality and brain-computer interaction for people with functional disabilities [Електронний ресурс] / P. Gang, J. Hui, S. Stirenko, Y. Gordienko [та ін.] // Future of Information and Communication Conference (FICC) – 2018. – С. 612–631. – Режим доступу до ресурсу: https://link.springer.com/chapter/10.1007/978-3-030-03402-3_43
29. Deep learning for fatigue estimation on the basis of multimodal human-machine interactions [Електронний ресурс] / Y. Gordienko, S. Stirenko, Y.

Kochura [та ін.] // arXiv preprint arXiv:1801.06048 – 2017. – Режим доступу до ресурсу: https://arxiv.org/abs/1801.06048

30. Scaling analysis of specialized tensor processing architectures for deep learning models [Електронний ресурс] / Y. Gordienko, Y. Kochura, V. Taran, N. Gordienko [та ін.] // Deep Learning: Concepts and Architectures – 2019. – С. 65–99. – Режим доступу до ресурсу: https://link.springer.com/chapter/10.1007/978-3-030-31756-0_3

ДОДАТОК А

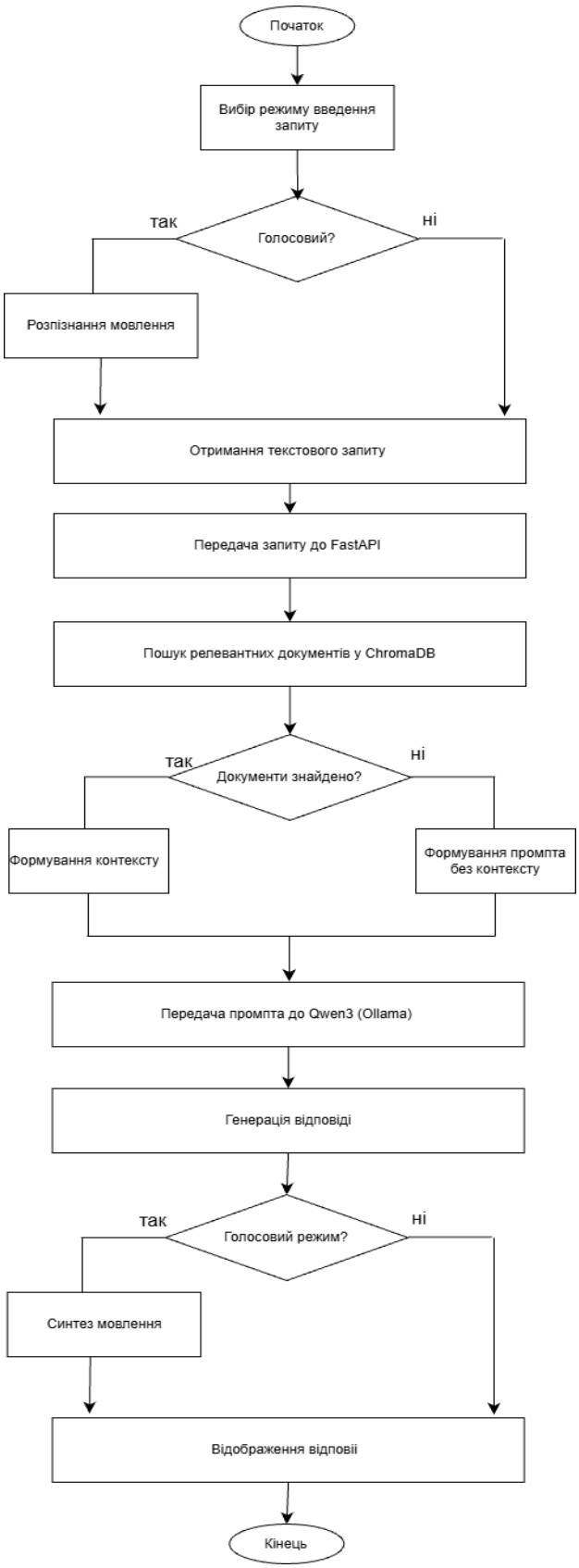
Система голосової взаємодії на основі методів штучного інтелекту

Принципова схема (схема алгоритму)

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2026 р.



					ІАЛЦ.467200.004 Д1	
		№ докум.	Підпис	Дата		
Розробив		Бондар А. А.			Система голосової взаємодії на основі методів штучного інтелекту **Принципова схема (схема алгоритму)**	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Стіренко С. Г.				
Реценз.						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-21
Н. Контр.		Коренко Д.В.				
Затвердив		Волокита А. М.				

ДОДАТОК Б

Система голосової взаємодії на основі методів штучного інтелекту

Функціональна схема (діаграма компонентів)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2026 р.

					ІАЛЦ.467200.005 Д2		
		№ докум.	Підпис	Дата			
Розробив		Бондар А. А.			Система голосової взаємодії на основі методів штучного інтелекту **Функціональна схема (діаграма компонентів)**	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Стіренко С. Г.					
Реценз.						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-21	
Н. Контр.		Коренко Д. В.					
Затвердив		Волокита А. М.					

ДОДАТОК В

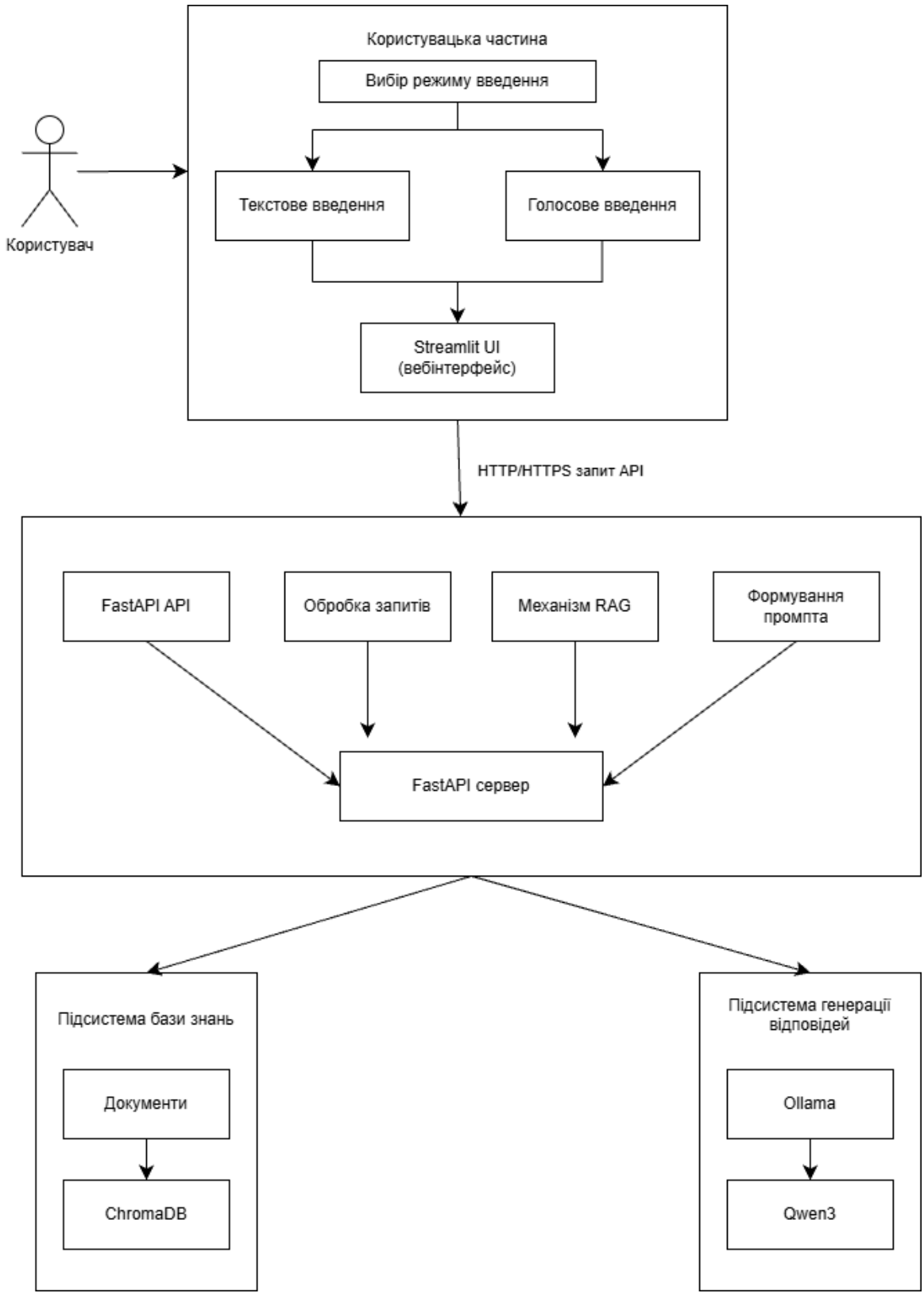
Система голосової взаємодії на основі методів штучного інтелекту

Структурна схема системи

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2026 р.



					ІАЛЦ.467200.006 Д3		
		№ докум.	Підпис	Дата			
Розробив		Бондар А. А.			Система голосової взаємодії на основі методів штучного інтелекту **Структурна схема системи**	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Стіренко С. Г.					
Реценз.						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-21	
Н. Контр.		Коренко Д. В.					
Затвердив		Волокита А. М.					

ДОДАТОК Г

Система голосової взаємодії на основі методів штучного інтелекту

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 16

Київ 2026 р.

```
# Модуль завантаження та індексації документів (ingest.py)
import shutil
from pathlib import Path

from langchain_community.document_loaders import TextLoader, PyPDFLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_ollama import OllamaEmbeddings
from langchain_chroma import Chroma
from langchain_core.documents import Document

DOCS_DIR = Path("documents")
DB_DIR = Path("chroma_db")

EMBED_MODEL = "nomic-embed-text"


def load_documents():
    documents = []
    if not DOCS_DIR.exists():
        return documents

    for file_path in DOCS_DIR.iterdir():
        if file_path.name == ".gitkeep":
            continue

        suffix = file_path.suffix.lower()

        if "infopk" in file_path.name.lower() or "вартість" in file_path.name.lower():
            if suffix == ".txt":
                try:
                    text = file_path.read_text(encoding="utf-8")
                    lines = [line.strip() for line in text.split("\n") if line.strip()]
                    for line in lines:
```

					ІАЛЦ.467200.007 Д4			
		№ докум.	Підпис	Дата				
Розробив		Бондар А. А.			Система голосової взаємодії на основі методів штучного інтелекту **Текст програмного коду**	Літ.	Аркуш	Аркушів
Перевірив		Стіренко С. Г.					1	16
Реценз.						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-21		
Н. Контр.		Коренко Д. В.						
Затвердив		Волокита А. М.						

```
                    documents.append(
                     Document(
                        page_content=line,
                        metadata={"source": file_path.name}
                     )
                    )
                print(f"Loaded Table TXT (line-by-line): {file_path.name}")
                continue
            except Exception as e:
                print(f"Error loading table {file_path.name}: {e}")
                continue

        if suffix == ".txt":
            try:
                loader = TextLoader(str(file_path), encoding="utf-8")
                documents.extend(loader.load())
                print(f"Loaded TXT: {file_path.name}")
            except Exception as e:
                print(f"Error loading TXT {file_path.name}: {e}")

        elif suffix == ".pdf":
            try:
                loader = PyPDFLoader(str(file_path))
                documents.extend(loader.load())
                print(f"Loaded PDF: {file_path.name}")
            except Exception as e:
                print(f"Error loading PDF {file_path.name}: {e}")

        else:
            print(f"Skipped unsupported file: {file_path.name}")

    return documents


def main():
    if not DOCS_DIR.exists():
        DOCS_DIR.mkdir(parents=True)
```

```
    documents = load_documents()

    if not documents:
        print("No documents found in documents/")
        return

    if DB_DIR.exists():
        shutil.rmtree(DB_DIR)
        print("Old chroma_db removed")

    splitter = RecursiveCharacterTextSplitter(
        chunk_size=600,
        chunk_overlap=120,
    )

    chunks = splitter.split_documents(documents)

    embeddings = OllamaEmbeddings(model=EMBED_MODEL)

    Chroma.from_documents(
        documents=chunks,
        embedding=embeddings,
        persist_directory=str(DB_DIR),
    )

    print(f"Successfully indexed {len(chunks)} chunks into {DB_DIR}")


if __name__ == "__main__":
    main()
```

Модуль логіки обробки запитів та механізму RAG (app.py)

```
from pathlib import Path
import re
from langchain_chroma import Chroma
from langchain_ollama import OllamaEmbeddings, ChatOllama
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.documents import Document

DB_DIR = "chroma_db"
DOCS_DIR = Path("documents")

LLM_MODEL = "qwen3:8b"  # Або "qwen3:1.5b", якщо тимчасово працюєш на CPU
EMBED_MODEL = "nomic-embed-text"

# ЄДИНИЙ СИСТЕМНИЙ ПРОМПТ ДЛЯ ВСІЄЇ СИСТЕМИ
PROMPT = """
Ти — офіційний штучний інтелект-асистент для абітурієнтів Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ).
Твоє завдання — відповідати на питання користувача виключно на основі наданого контексту українською мовою.

Залежно від вказаного РЕЖИМУ ВЗАЄМОДІЇ, формуй відповідь за такими правилами:

СТАНДАРТНИЙ РЕЖИМ (text):
- Відповідай детально, чітко та структуровано.
- Використовуй марковані списки для переліків, абзаци та точні цифри.
- Обов'язково вказуй шифри спеціальностей та офіційні назви програм.

ГОЛОСОВИЙ РЕЖИМ (voice):
- Формуй відповідь як коротку, розмовну репліку для телефону (максимум 2-3 речення).
- Пиши ТІЛЬКИ суцільний текст. Жодних списків (*, -, 1.), дужок, лапок, знаків плюс чи абзаців.
- Замість технічних шифрів пиши слова (наприклад, замість "121" пиши "сто двадцять перша спеціальність").
- Округляй прохідні бали до цілих (наприклад, замість "158.615" кажи "близько ста п'ятдесяти дев'яти балів").

Загальні залізобетонні бізнес-правила:
1. ФІОТ — це факультет інформатики та обчислювальної техніки. На ньому є спеціальності 121, 123 та 126.
2. Кафедра ОТ — це лише одна з кафедр всередині ФІОТ, яка готує ТІЛЬКИ за спеціальностями 121 та 123.
```

3. Спеціальність 126 — це загальнофакультетський напрям ФІОТ, до кафедри ОТ вона не належить.

4. Якщо інформації взагалі немає в контексті, в режимі "text" скажи: "У наданих документах я не знайшов точної відповіді на це питання.", а в режимі "voice" скажи коротко: "На жаль, у мене немає цих даних. Бажаєте, я з'єднаю вас з оператором?".

Контекст:

{context}

РЕЖИМ ВЗАЄМОДІЇ: {mode}

Питання користувача: {question}

Відповідь асистента:

"""

```
def normalize(text: str) -> str:
    return text.lower().replace("’", "").replace("`", "")

def question_keywords(question: str):
    q = normalize(question)
    keywords = re.findall(r"[a-zA-Za-яA-ЯіїєґІЇЄҐ0-9]+", q)
    extra = []
    if "фіот" in q or "факультет" in q:
        extra += ["фіот", "факультет", "f2", "f6", "f7", "121", "126", "123"]
    if "кафедр" in q or "кафедра от" in q:
        extra += ["кафедра", "от", "f2", "f7", "121", "123"]
    if "варт" in q or "кошту" in q or "ціна" in q or "контракт" in q:
        extra += ["вартість", "навчання", "грн", "f2", "f6", "f7", "121", "126", "123"]
    if "прохід" in q or "бал" in q:
        extra += ["прохідний", "бал", "2024", "2023", "фіот", "f2", "f6", "f7"]
    return list(dict.fromkeys(keywords + extra))

def score_text(text: str, keywords: list[str]) -> int:
    t = normalize(text)
    score = 0
    for kw in keywords:
        if kw and kw in t:
            score += 1
    return score
```

```
def load_full_txt_docs():
    docs = []
    if not DOCS_DIR.exists():
        return docs
    for path in sorted(DOCS_DIR.glob("*.txt")):
        try:
            text = path.read_text(encoding="utf-8").strip()
            if text:
                docs.append(Document(page_content=text, metadata={"source": str(path), "source_name": path.name}))
        except Exception:
            continue
    return docs

def load_vectorstore():
    embeddings = OllamaEmbeddings(model=EMBED_MODEL)
    return Chroma(persist_directory=DB_DIR, embedding_function=embeddings)

def load_llm():
    return ChatOllama(model=LLM_MODEL, temperature=0.1)

def get_keyword_docs(question: str, limit: int = 4):
    keywords = question_keywords(question)
    candidates = []
    for doc in load_full_txt_docs():
        score = score_text(doc.page_content, keywords)
        if "fiot_summary" in doc.metadata.get("source", ""):
            continue
        if score > 0:
            candidates.append((score, doc))
    candidates.sort(key=lambda x: x[0], reverse=True)
    result = []
    seen = set()
    for _, doc in candidates:
        key = (doc.metadata.get("source", ""), doc.page_content[:300])
        if key not in seen:
            seen.add(key)
            result.append(doc)
        if len(result) >= limit:
```

```
            break
    return result


def get_vector_docs(question: str, limit: int = 3):
    vectorstore = load_vectorstore()
    retriever = vectorstore.as_retriever(
        search_type="mmr",
        search_kwargs={"k": limit, "fetch_k": 15, "lambda_mult": 0.5},
    )
    return retriever.invoke(question)


def deduplicate_docs(docs):
    result = []
    seen = set()
    for doc in docs:
        key = (doc.metadata.get("source", ""), doc.page_content[:300])
        if key not in seen:
            seen.add(key)
            result.append(doc)
    return result


def format_context(docs):
    parts = []
    for i, doc in enumerate(docs, start=1):
        source = doc.metadata.get("source_name", doc.metadata.get("source", "Документ"))
        parts.append(f"\n\n--- ДОКУМЕНТ {i} (Джерело: {source}) ---\n{doc.page_content}")
    return "\n".join(parts)


def retrieve_docs(question: str):
    q = normalize(question)
    docs = []
    if any(word in q for word in ["фіот", "факультет", "спеціальн", "бал", "варт", "ціна", "контракт", "121",
"123", "126"]):
        for doc in load_full_txt_docs():
            if "fiot_summary" in doc.metadata.get("source", ""):
                docs.append(doc)
    docs.extend(get_keyword_docs(question, limit=3))
    docs.extend(get_vector_docs(question, limit=2))
    docs = deduplicate_docs(docs)
```

```
    return docs[:4]

def answer_question(question: str, mode: str = "text"):
    docs = retrieve_docs(question)
    context = format_context(docs)
    llm = load_llm()

    prompt = ChatPromptTemplate.from_template(PROMPT)
    chain = prompt | llm

    response = chain.invoke({
        "context": context,
        "question": question,
        "mode": mode
    })
    return response.content, docs
```

```
# Серверний модуль FastAPI (api.py)
import os
from fastapi import FastAPI, HTTPException
from fastapi.responses import FileResponse
from pydantic import BaseModel, Field
from gtts import gTTS
from app import answer_question

app = FastAPI(
    title="KPI Admission API",
    description="Єдиний бекенд для обробки текстових та голосових запитів",
    version="2.0.0"
)

AUDIO_DIR = "static_audio"
os.makedirs(AUDIO_DIR, exist_ok=True)

class QuestionRequest(BaseModel):
    question: str
    mode: str = "text"

class ApiResponse(BaseModel):
    mode: str
    answer: str
    sources: list[str]

@app.post("/api/ask", response_model=ApiResponse)
async def ask_bot(payload: QuestionRequest):
    """ Повертає текстовий JSON (підтримує як mode='text', так і mode='voice') """
    if not payload.question.strip():
        raise HTTPException(status_code=400, detail="Питання порожнє")

    req_mode = "voice" if payload.mode.lower() == "voice" else "text"

    try:
        answer, docs = answer_question(payload.question.strip(), mode=req_mode)
        sources = list(set([doc.metadata.get("source_name", "Документ") for doc in docs]))
        return ApiResponse(mode=req_mode, answer=answer, sources=sources)
    except Exception as e:
```

```
        raise HTTPException(status_code=500, detail=str(e))

@app.post("/api/ask_voice_stream")
async def ask_bot_voice_stream(payload: QuestionRequest):
    """ Приймає питання і одразу повертає .mp3 файл з лаконічною аудіо-відповіддю """
    if not payload.question.strip():
        raise HTTPException(status_code=400, detail="Питання порожнє")

    try:
        answer, _ = answer_question(payload.question.strip(), mode="voice")

        tts = gTTS(text=answer, lang='uk', slow=False)
        audio_path = os.path.join(AUDIO_DIR, "response.mp3")
        tts.save(audio_path)

        return FileResponse(audio_path, media_type="audio/mp3", filename="response.mp3")
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

```
# Модуль користувацького інтерфейсу Streamlit (ui.py)
import os
import re
import time

import requests
import speech_recognition as sr
import streamlit as st
from gtts import gTTS
from mutagen.mp3 import MP3


API_URL_TEXT =  "http://77.47.130.18:8000/api/ask"

AUDIO_DIR = "static_audio"
os.makedirs(AUDIO_DIR, exist_ok=True)


st.set_page_config(page_title="AI Admission Assistant", layout="centered")
st.title("AI Admission Assistant ФІОТ КПІ")
st.caption("Уніфікований діалоговий інтерфейс приймальної комісії: текстовий і голосовий режим")


if "messages" not in st.session_state:
    st.session_state.messages = []

if "voice_active" not in st.session_state:
    st.session_state.voice_active = False

if "should_listen" not in st.session_state:
    st.session_state.should_listen = False


def clean_answer(answer: str) -> str:
    if not answer:
        return ""

    forbidden_patterns = [
        r"Бажаєте,?\s*я\s*з[’`]?єднаю\s*вас\s*з\s*оператором\??",
```

```
        r"Я\s*можу\s*з[''`]?єднати\s*вас\s*з\s*оператором\.?",
        r"З[''`]?єднати\s*вас\s*з\s*оператором\??",
        r"Хочете,?\s*я\s*з[''`]?єднаю\s*вас\s*з\s*оператором\??",
    ]

    cleaned = answer

    for pattern in forbidden_patterns:
        cleaned = re.sub(pattern, "", cleaned, flags=re.IGNORECASE)

    cleaned = re.sub(r"\s{2,}", " ", cleaned).strip()

    if cleaned.endswith(","):
        cleaned = cleaned[:-1].strip()

    return cleaned


def record_and_recognize():
    recognizer = sr.Recognizer()
    recognizer.pause_threshold = 1.0
    recognizer.energy_threshold = 300
    recognizer.dynamic_energy_threshold = True

    with sr.Microphone() as source:
        st.info("Мікрофон відкрито. Говоріть...")
        recognizer.adjust_for_ambient_noise(source, duration=0.5)

        try:
            audio = recognizer.listen(
                source,
                timeout=5,
                phrase_time_limit=8,
            )
            st.info("Очікуйте, обробка мовлення...")
            return recognizer.recognize_google(audio, language="uk-UA")
        except sr.WaitTimeoutError:
            return None
        except Exception:
```

```
            return None


def make_audio(answer: str) -> str:
    filename = f"reply_{int(time.time())}.mp3"
    audio_path = os.path.join(AUDIO_DIR, filename)

    tts = gTTS(text=answer, lang="uk", slow=False)
    tts.save(audio_path)

    return audio_path


def get_audio_duration_seconds(audio_path: str, fallback_text: str = "") -> float:
    """
    Повертає реальну довжину mp3 у секундах.
    Якщо не вдалося прочитати файл — використовує запасний розрахунок за кількістю слів.
    """
    try:
        audio = MP3(audio_path)
        duration = float(audio.info.length)

        if duration > 0:
            return duration

    except Exception:
        pass

    words_count = len(fallback_text.split())
    return max(3.0, words_count / 2.4 + 1.0)


def send_to_backend(question: str, mode: str):
    st.session_state.messages.append(
        {
            "role": "user",
            "text": question,
            "audio_path": None,
        }
```

```
)

try:
    res = requests.post(
        API_URL_TEXT,
        json={
            "question": question,
            "mode": mode,
        },
        timeout=180,
    )

    if res.status_code != 200:
        st.error(f"Помилка сервера бекенду: {res.status_code} — {res.text}")
        return None, None

    answer = res.json().get("answer", "")
    answer = clean_answer(answer)

    if not answer:
        answer = "На жаль, у моїй базі знань немає точної відповіді на це питання."

    audio_path = make_audio(answer)

    st.session_state.messages.append(
        {
            "role": "assistant",
            "text": answer,
            "audio_path": audio_path,
        }
    )

    return answer, audio_path

except Exception as e:
    st.error(f"Не вдалося з'єднатися з api.py: {e}")
    return None, None
```

```
col1, col2 = st.columns(2)

with col1:
    if not st.session_state.voice_active:
        if st.button("Розпочати голосовий чат", use_container_width=True):
            st.session_state.voice_active = True
            st.session_state.should_listen = True
            st.rerun()
    else:
        if st.button("Припинити голосовий чат", use_container_width=True):
            st.session_state.voice_active = False
            st.session_state.should_listen = False
            st.rerun()

with col2:
    if st.button("Очистити історію чату", use_container_width=True):
        st.session_state.messages = []
        st.session_state.should_listen = st.session_state.voice_active
        st.rerun()



st.write("---")



for msg in st.session_state.messages:
    if msg["role"] == "user":
        with st.chat_message("user"):
            st.write(msg["text"])
    else:
        with st.chat_message("assistant"):
            st.write(msg["text"])
            if msg.get("audio_path") and os.path.exists(msg["audio_path"]):
                st.audio(msg["audio_path"], format="audio/mp3")



if st.session_state.voice_active and st.session_state.should_listen:
    voice_question = record_and_recognize()

    if voice_question:
```

```
            st.session_state.should_listen = False
            answer_text, audio_path = send_to_backend(voice_question, mode="voice")

            if answer_text:
                st.rerun()
        else:
            time.sleep(1)
            st.rerun()


elif st.session_state.voice_active and not st.session_state.should_listen:
    if st.session_state.messages and st.session_state.messages[-1]["role"] == "assistant":
        last_msg = st.session_state.messages[-1]

        if last_msg.get("audio_path") and os.path.exists(last_msg["audio_path"]):
            st.audio(last_msg["audio_path"], format="audio/mp3", autoplay=True)

            audio_duration = get_audio_duration_seconds(
                last_msg["audio_path"],
                fallback_text=last_msg["text"],
            )

            # Невелика пауза після завершення аудіо, щоб мікрофон не вмикався занадто рано.
            time.sleep(audio_duration + 0.7)

            st.session_state.should_listen = True
            st.rerun()


elif not st.session_state.voice_active:
    if text_question := st.chat_input("Задайте ваше питання тут..."):
        send_to_backend(text_question, mode="text")
        st.rerun()
```