Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

«Інтерактивна система підтримки діяльності викладача в дистанційному навчанні»

08-54.БКР.024.00.000 ПЗ

Виконав: студент 4 курсу, групи 2СП-21б
спеціальності
123 Комп'ютерна інженерія,
(шифр і назва напряму підготовки, спеціальності)
освітня програма
«Системне програмування»
Гордійчук І.М.
(прізвище та ініціали)

Керівник: доц. каф. ОТ
Снігур А.В.
(прізвище та ініціали)

Рецензент: д.т.н., проф. Голова секції УБ каф. МБІС
Яремчук Ю.Є.
(прізвище та ініціали)

ДОПУЩЕНО
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д

"16" 06 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень бакалавр
Галузь знань – 12 – Інформаційні технології
Спеціальність – 123 – Комп'ютерна інженерія
Освітньо-професійна програма – Системне програмування

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д
_____
" 21 " березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Гордійчуку Іллі Миколайовичу

1 Тема роботи: «Інтерактивна система підтримки діяльності викладача в дистанційному навчанні», керівник роботи Снігур А.В. к.т.н., доцент кафедри ОТ затверджені наказом ВНТУ від «20» 03 2025 року №97.

2 Термін подання студентом роботи: 13.05.2025.

3 Вихідні дані до роботи: платформа – Ollama для доступу до великих мовних моделей; інтерфейс програми це веб-застосунок, який розроблений мовою програмування Python в поєднані з бібліотекою для створення користувацького інтерфейсу Streamlit; передбачено створити – веб-застосунок, що буде використовуватися викладачами для інтерактивної підтримки та виконання рутинних задач під час проведення дистанційного освітнього процесу.

4 Зміст розрахунково-пояснювальної записки: вступ, огляд існуючих технологій та засобів інтерактивної підтримки, особливості створення програмного забезпечення для підсистеми інтерактивної підтримки, програмна

реалізація ключових компонентів підсистеми, тестування та оцінка ефектив
підсистеми, висновки, список використаних джерел і додатки.

5 Перелік графічного матеріалу (з точним зазначенням обов'яз
креслень): блок-схема функціонування застосунку; структурна схема алгор
обробки PDF-документів; схема програми.

6 Консультанти розділів роботи наведені у таблиці 1.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	завдан прийн
Спеціальна частина	Снігур А.В., доцент кафедри ОТ	21.03.2025	10.05.2
Нормоконтроль	ас. каф. ОТ Швець С. І.	14.05.2025	30.05.2

7 Дата видачі завдання «21» 03 2025 р. Календарний план наведен
таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів	Прим
1	Постановка задачі роботи	21.03.2025	Вик.
2	Пошук матеріалів та інформаційних джерел	21.03.25— 30.03.25	вик.
3	Аналіз існуючих прототипів та визначення вимог до створюваної системи	21.03.25— 30.03.25	вик.
4	Обґрунтування структури та принципів роботи системи	31.03.25— 13.04.25	вик.
5	Розробка й тестування апаратно-програмної системи	14.04.25— 27.04.25	вик.
6	Оформлення пояснювальної записки та ілюстративного матеріалу	28.04.25— 11.05.25	вик.
7	Перевірка якості виконання бакалаврської кваліфікаційної роботи та усунення недоліків	13.05.25	вик.

Студент _____ (підпис) Гордійчук І.М.

Керівник роботи _____ (підпис) Снігур А.В.

АНОТАЦІЯ

УДК 004.5

Гордійчук І.М. Розробка підсистеми інтерактивної підтримки викладача у дистанційному навчанні. Бакалаврська кваліфікаційна робота зі спеціальності 123 – Комп'ютерна інженерія, освітня програма – Системне програмування. Вінниця: ВНТУ, 2025. 72 с.

На укр. мові. Бібліогр.:20; рис.: 27, табл.: 2.

У бакалаврській кваліфікаційній роботі розроблено підсистему інтерактивної підтримки викладача для дистанційного навчання на основі штучного інтелекту з використанням мови програмування Python. Система інтегрує Велику мовну модель (LLM) Gemma3, застосовує підхід Retrieval-Augmented Generation (RAG) для обробки PDF-документів, реалізує алгоритм роботи з базою даних студентів і забезпечує зручний та інтуїтивно зрозумілий інтерфейс користувача.

Метою роботи є створення ефективного та простого у використанні інструменту для викладачів, адаптованого до потреб університетського середовища, що дозволяє автоматизувати рутинні завдання та надавати швидкі відповіді на типові запитання.

Ключові слова: штучний інтелект, Python, LLM, RAG

ABSTRACT

Hordiichuk I.M. Development of an Interactive Teacher’s Support Subsystem for Distance Learning. Bachelor's thesis in specialty 123 – Computer Engineering, educational program – Computer Engineering. Vinnytsia: VNTU, 2025. 72 p.

In Ukrainian language. Bibleographer:20; Fig.: 27, Tab.: 2.

This bachelor’s thesis presents the development of an interactive instructor support subsystem for distance learning, based on artificial intelligence and implemented in Python. The system integrates the large language model (LLM) Gemma3, utilizes the Retrieval-Augmented Generation (RAG) approach for processing PDF documents, includes an algorithm for managing a student database, and provides a user-friendly, intuitive interface.

The primary goal of the project is to create an efficient and easy-to-use tool tailored to the university environment, enabling automation of routine tasks and delivering quick responses to frequently asked questions.

Keywords: artificial intelligence, Python, LLM, RAG

ЗМІСТ

ВСТУП

Інтерактивна система підтримки діяльності викладача в дистанційному навчанні є сукупністю програмних засобів, орієнтованих на автоматизацію рутинних завдань викладача, надання швидких відповідей на типові запити та ефективне використання навчальних матеріалів. Із розвитком великих мовних моделей (LLM) та сучасних підходів до обробки природної мови, стало можливим створення інтелектуальних систем, здатних імітувати живу комунікацію та надавати інформаційну підтримку в реальному часі. Завдяки поєднанню LLM з методами Retrieval-Augmented Generation (RAG), такі системи здатні працювати з неструктурованими джерелами знань, включаючи документи у форматі PDF, бази даних тощо.

Актуальність теми бакалаврської кваліфікаційної роботи зумовлена зростаючими потребами в автоматизації освітнього процесу, особливо у форматі дистанційного навчання, яке набуло особливої ваги в останні роки. Викладачі все частіше зіштовхуються з великими обсягами однотипних запитів, необхідністю працювати з різними джерелами даних та потребою в інструментах, що спрощують роботу без втрати якості. Розробка чат-бота на основі LLM із можливістю доступу до навчальних матеріалів, списків студентів та іншої інформації дозволяє суттєво підвищити продуктивність викладача та знизити навантаження.

Однією з переваг таких систем є гнучкість у роботі з контекстом, здатність до самонавчання та адаптації під конкретні запити користувача. Завдяки відкритим платформам, як-от Ollama, стало можливим локальне розгортання мовних моделей без потреби у зовнішніх API, що важливо для захисту даних та роботи в обмежених мережевих умовах.

Об’єктом дослідження є процеси підтримки викладача в умовах дистанційного навчання.

Предметом дослідження є методи створення інтерактивних програмних рішень на основі LLM для автоматизації освітніх процесів.

Метою бакалаврської кваліфікаційної роботи є розробка та впровадження

підсистеми інтерактивної підтримки викладача, яка базується на великій мовній моделі Gemma3 та реалізує підхід RAG для роботи з PDF-документами, а також включає модуль для обробки бази даних студентів і зручний веб-інтерфейс для взаємодії.

Для досягнення мети роботи необхідно вирішити такі завдання:

– аналіз сучасних тенденцій у сфері застосування великих мовних моделей у освіті;

– виконати огляд існуючих рішень підтримки викладача та порівняння їх функціональності;

– вибрати та обґрунтувати архітектуру системи та технології реалізації;

– розробити програмний компонент з використанням обраного стеку технологій;

– провести тестування системи, оцінити її ефективність та зручність використання.

Апробація та публікація результатів бакалаврської кваліфікаційної роботи здійснена шляхом підготовки та подання заявки на патент України на корисну модель «Інтерактивна система підтримки діяльності викладача в дистанційному навчанні». Станом на момент захисту роботи заявка перебуває на етапі публікації.

1 ОГЛЯД ІСНУЮЧИХ ТЕХНОЛОГІЙ ТА ЗАСОБІВ ПІДТРИМКИ ДІЯЛЬНОСТІ ВИКЛАДАЧА

1.1 Модель взаємодії викладача з мовною моделлю у навчальному процесі

Використання великих мовних моделей у сфері дистанційного навчання стає потужним інструментом для автоматизації рутинних процесів і підвищення якості освітнього досвіду. Завдяки динамічній генерації відповідей на запити викладачів та інтеграції з різноманітними джерелами даних, такі системи забезпечують адаптивність, оперативність і зручність у щоденній роботі. Це відкриває нові можливості для персоналізації взаємодії з навчальним матеріалом та оптимізації часу, витраченого на адміністративні завдання.

У сучасному освітньому середовищі чат-боти можуть забезпечувати підтримку як для учнів, так і для педагогів. В Україні та за її межами впроваджено низку таких інструментів, які сприяють покращенню доступу до інформації, організації навчального процесу та наданню психологічної підтримки. Чат-боти відіграють особливу роль у кризових ситуаціях, зокрема під час пандемії COVID-19 або війни, коли оперативний доступ до освітніх ресурсів є критично важливим.

EducationUaBot – офіційний освітній чат-бот, розроблений Міністерством освіти і науки України за підтримки Швейцарського проєкту DECIDE. Він надає актуальну інформацію про освітній процес в Україні та за кордоном, допомагає знайти освітні заклади, організувати онлайн-навчання, відновити документи про освіту та продовжити викладання під час війни. Цей бот доступний у Telegram та Viber, що забезпечує широкий доступ до нього для користувачів [1].

Чат-бот "Спільно для вчителів", створений освітньою організацією "Навчай для України" за підтримки ЮНІСЕФ, допомагає вчителям у щоденній роботі та кризових ситуаціях. Він надає доступ до навчальних ресурсів, дозволяє фіксувати спостереження про учнів, отримувати поради щодо дій у різних ситуаціях, а також містить поради з емоційного вигорання та підтримки ментального здоров’я освітян.

NoMoreGringe.Powerbank – освітній бот-тренажер, спрямований на підвищення гендерної грамотності та розвиток інклюзивного мислення серед підлітків. Цей

Telegram-бот пропонує інтерактивні завдання, вікторини та відео, що допомагають молоді краще розуміти питання інклюзії та гендерної рівності. Його використання в навчальних програмах дозволяє інтегрувати принципи толерантності та різноманіття в освітній процес [2].

На міжнародному рівні EduChat – чат-бот, заснований на великих мовних моделях, спрямований на підтримку персоналізованого та інтелектуального навчання. Він надає можливості для відкритого запитання-відповіді, оцінювання есе, сократичного навчання та емоційної підтримки, що робить його корисним інструментом для вчителів, студентів та батьків [3]. Завдяки використанню штучного інтелекту, EduChat може адаптуватися до індивідуальних стилів навчання та пропонувати відповідний темп засвоєння матеріалу.

Системи автоматизованої підтримки викладача модернізують освітній процес, забезпечуючи ефективне управління навчальними матеріалами, моніторинг успішності студентів та оптимізацію адміністративних завдань. В Україні та світі впроваджено низку таких систем, які сприяють підвищенню якості освіти та полегшенню роботи педагогів.

Однією з найпоширеніших платформ є Moodle – система управління навчанням з відкритим кодом, яка дозволяє викладачам створювати курси, організовувати тестування, відстежувати прогрес студентів та взаємодіяти з ними через форуми та чати. Moodle широко використовується в українських вищих навчальних закладах завдяки своїй гнучкості та можливості адаптації до специфічних потреб навчальних програм [4]. Додатково, вона підтримує підключення зовнішніх плагінів, що дозволяє інтегрувати інструменти на основі штучного інтелекту без потреби у повному оновленні платформи.

Іншою популярною системою є Google Classroom, яка інтегрується з іншими сервісами Google, такими як Google Docs, Sheets та Drive, забезпечуючи зручне середовище для створення та розповсюдження навчальних матеріалів, збору завдань та надання зворотного зв’язку студентам. Її простота у використанні та доступність роблять її привабливою для багатьох освітніх установ. Google постійно впроваджує

нові функції, наприклад, автоматичну оцінку завдань, що особливо цінується в умовах великого навантаження на викладачів.

Canvas – ще одна система управління навчанням, яка набуває популярності завдяки сучасному інтерфейсу та широкому спектру функцій, включаючи мобільні додатки, аналітику навчального процесу та можливості інтеграції з іншими освітніми інструментами. Canvas підтримує адаптивне навчання та дозволяє викладачам персоналізувати освітній досвід для кожного студента, що особливо важливо в умовах змішаного та дистанційного навчання.

Крім того, в Україні розробляються національні платформи, такі як Єдина державна електронна база з питань освіти (ЄДЕБО), яка забезпечує централізоване зберігання та обробку даних про освітній процес, сприяючи автоматизації адміністративних процедур та підвищенню прозорості в освіті. ЄДЕБО інтегрується з іншими державними реєстрами, що дозволяє спростити перевірку документів, подання заяв і контроль за освітніми програмами.

У сукупності ці інструменти створюють нову екосистему дистанційного та гібридного навчання, у якій великі мовні моделі відіграють дедалі важливішу роль – як інструмент підтримки, оцінювання, генерації змісту й підвищення інклюзивності освітнього середовища.

1.2 Актуальність застосування LLM для підтримки діяльності викладача

Великі мовні моделі, зокрема GPT, Claude, LLaMA, Gemma та інші, стрімко інтегруються в освітній процес на всіх рівнях – від шкільної освіти до вищої школи. Їх широкі можливості щодо генерації, аналізу та трансформації текстів природною мовою відкривають нові горизонти для підвищення якості викладання, індивідуалізації навчання, а також автоматизації рутинних педагогічних завдань.

Серед основних напрямів застосування LLM в освіті слід відзначити автоматизоване створення навчальних матеріалів. Це охоплює не лише генерацію текстів лекцій, планів занять, словників термінів та методичних вказівок, але й адаптацію їх до різного рівня підготовки студентів. Наприклад, модель може

сформулювати одне і те саме поняття простою мовою для першокурсника та на академічному рівні – для аспіранта. Це забезпечує гнучкість у поданні знань і створює умови для індивідуального навчального підходу [5].

Викладачі дедалі частіше використовують LLM для створення завдань, тестів, контрольних робіт і навіть кейсів для ситуаційного аналізу. Моделі здатні не лише генерувати запитання, а й автоматично оцінювати відповіді студентів. Це дозволяє суттєво скоротити час на перевірку великої кількості робіт, особливо в умовах дистанційного навчання або при роботі з масовими онлайн-курсами (MOOC).

Варто також згадати про застосування LLM у програмуванні: моделі на кшталт Code Llama або Copilot здатні аналізувати студентський код, знаходити помилки та пропонувати рекомендації щодо їх виправлення. У такий спосіб мовна модель виконує функцію інтерактивного менторства, що особливо цінне під час індивідуального навчання, коли відсутній постійний доступ до викладача.

У контексті дистанційного або змішаного навчання важливою є функція віртуального помічника – LLM, що відповідає на запитання студентів у чаті, підказує ресурси, допомагає з навігацією навчального контенту. Такий інструмент суттєво знижує навантаження на викладачів і підвищує залученість студентів до освітнього процесу.

Однак широке впровадження LLM у сферу освіти супроводжується рядом етичних та технічних викликів. Одним із головних є питання достовірності генерованої інформації. Мовна модель, яка не володіє розумінням у людському сенсі, може створити правдоподібний, логічно побудований, але хибний текст. Це явище дістало назву "галюцинації" моделі. У контексті освіти така помилка може призвести до поширення хибних знань, що особливо небезпечно при підготовці майбутніх фахівців.

Ще однією важливою проблемою є когнітивні упередження, які LLM успадковують з даних, на яких їх було навчено. Такі моделі можуть відображати гендерні, культурні, расові або політичні стереотипи, що порушує принципи академічної доброчесності та нейтральності. Це створює ризик для об’єктивності

навчального матеріалу та може негативно вплинути на студентів, особливо в мультикультурному освітньому середовищі [6].

Крім того, LLM можуть бути вразливими до цільових атак, зокрема prompt injection – техніки, за допомогою якої зловмисник може вставити у запит до моделі спеціально сформульовану інструкцію для отримання небажаної, некоректної або навіть небезпечної відповіді. У навчальному середовищі це може використовуватись для обходу систем перевірки знань або генерації відповідей на екзаменаційні запитання.

Ще один етичний аспект, що викликає дискусії, – це вплив мовних моделей на навички студентів. Існує загроза того, що надмірне використання генеративного ШІ знижуватиме здатність до самостійного мислення, критичного аналізу та дослідницької роботи. Замість того, щоб розв’язувати задачу самостійно, студент може просто попросити модель сформулювати відповідь, і таким чином пропустити важливий когнітивний етап навчання.

Не менш важливою є й проблема енергоспоживання. Тренування LLM вимагає потужних обчислювальних ресурсів, а отже – споживає великі обсяги електроенергії. Це викликає занепокоєння з точки зору сталого розвитку та екологічної відповідальності. Наприклад, одне повноцінне навчання великої трансформерної моделі здатне виробити вуглецевий слід, співставний із кількома перельотами через Атлантику [7].

Для пом’якшення цих викликів сучасна дослідницька спільнота пропонує низку підходів: стандартизацію навчальних даних, впровадження фільтрів упередженого контенту, розробку energy-efficient моделей, інтеграцію механізмів пояснення (explainability) у LLM тощо. Крім того, все більше університетів та розробників ШІ наполягають на впровадженні етичного кодексу для використання мовних моделей в освіті.

У підсумку, великі мовні моделі відкривають потужні можливості для трансформації освітнього процесу, але вимагають зваженого, відповідального та обґрунтованого використання. Лише за умови ретельного контролю, етичного

регулювання і чіткого розуміння їхніх обмежень, LLM можуть стати ефективним інструментом модернізації освіти.

1.3 Програмні засоби для обробки текстових запитів та знань

Великі мовні моделі (LLM) – це потужні алгоритми штучного інтелекту, здатні обробляти, аналізувати та генерувати текст, подібно до людської мови. Ці моделі навчаються на величезних обсягах текстових даних, що дозволяє їм розпізнавати складні мовні патерни та контексти, забезпечуючи високий рівень розуміння та генерації природної мови.

LLM базуються на архітектурі трансформерів, яка була представлена в 2017 році в дослідженні "Attention Is All You Need" [8]. Ця архітектура використовує механізм самоуваги (self-attention), що дозволяє моделі ефективно обробляти довгі послідовності тексту, фокусуючись на найбільш релевантних частинах вхідних даних. Однією з важливою особливостей LLM є їх здатність до генерації тексту, що має логічну послідовність та відповідає заданому контексту. Це досягається завдяки попередньому навчання на великих корпусах текстів, таких як Wikipedia, новинні статті, форуми та інші джерела.

Відомими прикладами LLM є BERT, GPT-3 й GPT-4 та Gemini. Розроблений Google у 2018 році, BERT використовує двосторонній контекст для розуміння значення слів у реченні, що значно покращує результати в задачах аналізу природної мови [9]. GPT-3 та GPT-4 були створені OpenAI. Вони мають мільярди параметрів і здатні генерувати текст, що важко відрізнити від написаного людиною. GPT-4, зокрема, є мультимодальною моделлю, яка може обробляти як текст, так і зображення. Gemini, розроблена Google DeepMind, є мультимодальною та здатна обробляти різні типи даних, включаючи текст, зображення, аудіо та відео. Gemini була представлена у 2023 році як наступник моделі PaLM 2.

Великі мовні моделі демонструють значний потенціал у трансформації різноманітних галузей, включаючи освіту, медицину, право, бізнес, фінанси, промисловість та електронну комерцію. Їх здатність до глибокого аналізу тексту та

генерації змістовних відповідей відкриває нові можливості для автоматизації процесів, покращення обслуговування клієнтів та прийняття обґрунтованих рішень.

У бізнесі та фінансах LLM використовуються для аналізу великих обсягів даних, виявлення тенденцій на ринку, прогнозування фінансових показників та виявлення шахрайських операцій. Вони також покращують обслуговування клієнтів, забезпечуючи швидкі та точні відповіді на запитання, що підвищує задоволеність клієнтів та ефективність роботи компаній [10].

У медицині LLM сприяють аналізу медичної документації, допомагають у постановці діагнозів та виборі оптимальних методів лікування. Вони також використовуються для створення віртуальних помічників, які надають пацієнтам інформацію про стан здоров'я, нагадують про прийом ліків та відповідають на поширені запитання, що підвищує рівень залученості пацієнтів у процес лікування [11].

У юридичній сфері LLM автоматизують аналіз контрактів, виявляючи потенційні ризики та невідповідності, а також прискорюють процес юридичних досліджень, надаючи швидкий доступ до релевантної інформації. Це дозволяє юристам зосередитися на стратегічних аспектах справ, зменшуючи час, витрачений на рутинні завдання.

Незважаючи на значні досягнення, LLM мають певні обмеження. Наприклад, моделі можуть генерувати неправдиву або некоректну інформацію, що виглядає достовірно. LLM можуть відтворювати упередження, присутні в даних, на яких вони навчалися. Також, навчання та використання LLM вимагає значних обчислювальних потужностей, що може бути недоступним для невеликих організацій. Теж важливо зазначити те, що використання персональних даних для навчання моделей викликає занепокоєння щодо захисту приватності.

Взаємодія з великою мовною моделлю відбувається переважно через текстові запити, які користувач формулює у вигляді природної мови. Цей процес, відомий як "prompt engineering", полягає у створенні чітких і специфічних запитів, що дозволяє моделі генерувати бажані відповіді. Ефективне використання LLM вимагає розуміння

того, як формулювання запиту впливає на результат, оскільки навіть незначні зміни у формулюванні можуть суттєво змінити відповідь моделі. Для полегшення взаємодії з LLM розробляються спеціалізовані інтерфейси, такі як графічні користувацькі інтерфейси або інтеграції з існуючими програмами, що дозволяє користувачам без глибоких технічних знань ефективно використовувати можливості моделі. Таким чином, взаємодія з LLM є процесом, що поєднує лінгвістичну інтуїцію користувача з технічними можливостями моделі, забезпечуючи потужний інструмент для вирішення широкого спектра завдань [12].

1.4 Порівняльний аналіз моделей та інструментів

Розробка застосунків на основі великих мовних моделей та обробки природної мови (NLP) у середовищі Python спирається на потужний інструментарій відкритого коду. Однією з найпоширеніших бібліотек є Transformers від Hugging Face – вона надає доступ до великої кількості попередньо натренованих моделей, зокрема GPT, BERT, T5, Gemma та інших, і має зручний інтерфейс для інтеграції моделей у Python-додатки. Разом із бібліотекою Datasets ця екосистема спрощує роботу з текстовими корпусами, а Tokenizers дозволяє ефективно перетворювати текст у формат, придатний для обробки моделлю.

Іншою важливою бібліотекою є LangChain, яка спеціалізується на створенні додатків на основі LLM, що взаємодіють із зовнішніми джерелами даних, базами знань або користувацьким вводом. Саме LangChain часто використовується в реалізації RAG-систем (Retrieval-Augmented Generation) – підходу, що дозволяє покращити точність відповідей шляхом залучення додаткового контексту. Для локального розгортання моделей популярністю користується llama-cpp-python, яка підтримує запуск моделей прямо з Python-коду без потреби у великих обчислювальних ресурсах [13].

Для переробки та аналізу тексту широко застосовуються spaCy та NLTK – вони забезпечують інструменти для токенізації, лематизації, визначення частин мови, побудови залежностей тощо. У поєднанні з бібліотеками для роботи з API (requests,

FastAPI, Flask) та створення веб-інтерфейсів (Streamlit, Gradio) вони формують повноцінне середовище для розробки LLM-додатків освітнього спрямування. Такий набір дозволяє ефективно будувати чат-ботів, автоматизованих помічників викладача й інші інтерактивні системи підтримки.

Серед провідних LLM-рішень варто виділити ChatGPT від OpenAI, Claude від Anthropic, Gemini від Google та Gemma – відкриту модель від Google, орієнтовану на дослідницьке використання.

ChatGPT, зокрема його версія GPT-4o, є одним з найпопулярніших LLM-рішень, відомим своєю здатністю до ведення природних діалогів, генерації креативного контенту та вирішення технічних завдань. Модель підтримує широкий спектр мов, включаючи українську, та інтегрується з різноманітними платформами через API. У сфері освіти ChatGPT може використовуватися для створення навчальних матеріалів, автоматичного оцінювання завдань та надання миттєвих відповідей на запитання студентів.

Claude від Anthropic вирізняється своєю орієнтацією на етичність та безпеку, завдяки впровадженню концепції "Конституційного AI", яка забезпечує дотримання етичних норм без необхідності постійного людського контролю. Модель Claude 3.5 Sonnet демонструє високі результати в задачах, що вимагають глибокого аналізу та логічного мислення, що робить її особливо корисною в освітніх контекстах, де важлива точність та надійність інформації [14].

Gemini від Google – це потужна мультимодальна модель, здатна обробляти текст, зображення, аудіо та відео. Завдяки інтеграції з екосистемою Google, включаючи Google Workspace та Android, Gemini забезпечує зручний доступ до інструментів для створення та аналізу навчальних матеріалів. Модель підтримує понад 40 мов, що робить її придатною для використання в багатомовних освітніх середовищах [15].

Gemma – це відкритий LLM-проєкт від Google, орієнтований на дослідницьке та академічне використання. Моделі Gemma 2 та Gemma 3 підтримують багатомовність (до 140 мов), мають довгі контекстні вікна (до 128 тис. токенів) та

здатні до мультимодальної обробки даних. Завдяки відкритому коду, Gemma надає можливість гнучкого налаштування та адаптації під специфічні потреби освітніх установ та дослідницьких проєктів.

1.5 Порівняльний аналіз систем підтримки викладача

Існує низка рішень, які відносяться до інтерактивних систем підтримки викладача для автоматизації рутинних завдань, генерації навчальних матеріалів та персоналізації навчального процесу. Нижче представлено порівняльний аналіз кількох таких систем.

Аналіз сучасного стану інтерактивних освітніх інструментів, зокрема чат-ботів і систем автоматизованої підтримки, свідчить про активне впровадження технологій штучного інтелекту в навчальний процес. В Україні вже діють кілька ініціатив, спрямованих на інформаційну та організаційну підтримку викладачів і студентів (таблиця 1.1). Водночас більшість із них мають обмежену функціональність: переважно надають довідкову інформацію, доступ до ресурсів або базову психологічну підтримку, не використовуючи потенціалу мовних моделей повною мірою.

Використання великих мовних моделей відкриває новий рівень взаємодії між педагогом і цифровим інструментом. На відміну від класичних ботів, LLM здатні аналізувати складні запити, працювати з динамічними джерелами даних, формувати осмислені відповіді на основі PDF-документів або баз студентів. Це перетворює освітню підтримку на інтелектуальний діалоговий процес, який може адаптуватися до конкретного контексту і завдань викладача.

Порівняння доступних рішень показало, що для реалізації локального асистента в умовах університету найбільш придатними є відкриті моделі, такі як Gemma3, які підтримують українську мову, швидко працюють без інтернет-з'єднання та мають відкритий код. У поєднанні з бібліотеками типу LangChain і базами векторів на кшталт Chroma, це дозволяє створити гнучку систему підтримки, яка може працювати з PDF-документами, структурованими таблицями (CSV) та навчальними запитами.

Таблиця 1.1 – Порівняльний аналіз систем інтерактивної підтримки викладача

Характеристика	MagicSchool AI	Eduaide.Ai	Khanmigo (Khan Academy)	Brisk Teaching	Застосунок, що розробляється
Працює без інтернету	-	-	-	-	+
Підтримка української мови	-	-	-	-	+
Інтеграція з PDF та CSV	-	-	-	-	+
Можливість локального запуску	-	-	-	-	+
Безкоштовність	±	±	+	-	+
Підтримка сценаріїв освітньої взаємодії	+	+	+	+	+
Орієнтований на роботу викладача	+	+	+	+	+

Водночас, проведений аналіз ринку показав, що на момент розробки не існувало безкоштовного, повністю локального рішення, яке було б спеціально адаптоване до потреб викладачів українських вищих навчальних закладів. Існуючі інструменти або потребували постійного доступу до інтернету, або були частиною комерційних продуктів, що не дозволяло інтегрувати їх у локальну інфраструктуру університету без фінансових чи юридичних обмежень. Більшість доступних рішень

орієнтовані на англомовне середовище і не забезпечують належної підтримки української мови, а також не мають інтерфейсів, адаптованих до потреб вітчизняної освітньої практики.

Особливо відчутним був брак простого у використанні, автономного та україномовного інструменту, здатного працювати без підключення до інтернету – що критично важливо в умовах нестабільного доступу до мережі або за вимог внутрішньої ізоляції навчальних серверів. Крім того, не було системи, яка могла б працювати з навчальними документами в PDF-форматі (методичками, конспектами тощо) та відповідати на запити викладача щодо студентів, використовуючи структуровані дані з локальних таблиць CSV. Таким чином, не вистачало інструменту, який би одночасно поєднував можливості генеративного ШІ, роботу з контекстними освітніми документами, захист персональних даних та локальну обробку запитів – без залучення зовнішніх серверів або хмарних сервісів.

Ця прогалина в наявних технологічних рішеннях і стала підставою для розробки власного бакалаврського проєкту. Було поставлено завдання створити універсальну програмну систему, яка б поєднувала потужність великої мовної моделі з локальним розгортанням, адаптованою україномовною взаємодією, можливістю роботи з навчальними матеріалами у вигляді PDF-документів і базою студентських даних у форматі CSV. Важливим критерієм стало забезпечення повної автономності системи без потреби підключення до інтернету, що гарантує високий рівень конфіденційності та дозволяє використовувати рішення у внутрішніх закритих мережах освітніх закладів.

Таким чином, вибір архітектури, програмних засобів та моделей у межах цієї роботи є обґрунтованим з погляду функціональності, безпеки, адаптивності до локальних умов і потенціалу подальшого розвитку. Створена система не лише демонструє технічну реалізованість ідей, закладених у концепції інтерактивної підтримки викладача, а й слугує основою для розгортання подібних рішень у вищих навчальних закладах України, де критично важливо забезпечити якісний доступ до цифрових інструментів на умовах незалежності від зовнішніх сервісів та платформ.

2 ОСОБЛИВОСТІ СТВОРЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ІНТЕРАКТИВНОЇ СИСТЕМИ

2.1 Аналіз архітектури систем на основі великих мовних моделей

Типовий LLM-додаток складається з кількох основних компонентів: інтерфейсу користувача (наприклад, веб або мобільного), модуля обробки запитів, самої великої мовної моделі (локальної чи хмарної), системи збереження контексту (пам’яті), зовнішніх джерел знань (як-от бази даних або векторні сховища), модуля оркестрації (наприклад, LangChain), а також механізмів безпеки та захисту даних. Разом ці складові забезпечують гнучку, масштабовану та безпечну роботу системи з можливістю обробки складних запитів у реальному часі.

Інтерфейс користувача виконує роль точки доступу до системи, через яку користувачі взаємодіють із LLM. Він забезпечує зручність введення запитів, перегляду відповідей і може містити інтерактивні елементи, такі як кнопки, меню або контекстні підказки. У складніших реалізаціях інтерфейс може адаптуватися до поведінки користувача, пропонуючи персоналізовані шаблони запитів, можливість зберігати історію взаємодії, автоматичне доповнення запитів тощо. Крім того, сучасні системи можуть включати голосовий ввід, мультимодальні компоненти (наприклад, завантаження зображень чи документів) або навіть підтримку доповненої реальності в мобільних застосунках.

Модуль обробки запитів відповідає за попередню обробку введення користувача, валідацію даних, форматування запиту для LLM та обробку результатів до виведення. Це може включати токенізацію, нормалізацію, синтаксичний аналіз або автоматичну класифікацію наміру (intent detection). На цьому етапі також можливе виявлення ключових слів, категоризація запитів, застосування шаблонів prompt-інженерії для кращої якості відповіді. Після отримання результату від LLM виконується післяобробка: виділення сутностей, візуалізація структури відповіді (наприклад, у вигляді списків, таблиць, діаграм) та вбудована перевірка релевантності або достовірності.

Велика мовна модель (LLM) є центральним компонентом, що забезпечує генерацію природномовного тексту на основі запиту користувача. Вона може бути локальною (розгорнутою в інфраструктурі користувача) або хмарною (через API-провайдерів, таких як OpenAI, Anthropic чи Mistral). Вибір між цими варіантами залежить від вимог до приватності, продуктивності та доступності. Локальні моделі забезпечують повний контроль над обробкою даних і можуть працювати в ізольованих середовищах, що важливо для організацій з підвищеними вимогами до конфіденційності. Хмарні рішення натомість пропонують більшу масштабованість, доступ до найновіших версій моделей та спрощене оновлення.

Система збереження контексту або пам’яті дозволяє LLM враховувати попередні взаємодії, що особливо важливо в багатокрокових або діалогових сценаріях. Така пам’ять може бути короткостроковою (в межах одного сеансу) або довготривалою (зберігати історію запитів, профілі користувачів тощо). Довготривала пам’ять часто реалізується через векторні сховища, пов’язані з Retrieval-Augmented Generation, або окремі бази даних, до яких зберігається стислий зміст попередніх взаємодій у формі summary, embedding або дерева рішень. Додатково можуть застосовуватись алгоритми управління контекстом, що автоматично визначають релевантні фрагменти пам’яті для підключення до поточного запиту.

Зовнішні джерела знань – це компоненти, які розширюють базу знань LLM за рахунок динамічного підключення до баз даних, API, документів, файлів або спеціалізованих векторних сховищ. У більшості сучасних архітектур ці джерела використовуються через Retrieval-Augmented Generation.

Технологія Retrieval-Augmented Generation (RAG) значно розширює можливості великих мовних моделей (LLM), дозволяючи їм ефективно взаємодіяти з нестандартними джерелами даних, такими як векторні бази даних, графи знань або спеціалізовані документи. На відміну від традиційних LLM, які обмежуються лише знаннями, отриманими під час навчання, RAG дозволяє моделі динамічно отримувати релевантну інформацію з цих джерел під час обробки запиту. Це забезпечує більш

точні, актуальні та контекстуально відповідні відповіді, знижуючи ймовірність виникнення "галюцинацій" – генерації неправдивої або неперевіреної інформації.

Завдяки інтеграції з векторними базами даних, такими як Pinecone, Chroma, Weaviate або FAISS, RAG дозволяє здійснювати семантичний пошук, що забезпечує точнішу відповідність між запитом користувача та релевантними фрагментами даних. У цьому процесі дані попередньо перетворюються у векторні представлення (ембединги) за допомогою моделей трансформерного типу, а потім зберігаються у векторному сховищі, що дає змогу швидко виконувати пошук за схожістю. Це особливо важливо в умовах великого обсягу неструктурованої інформації, де традиційний ключовий пошук є недостатньо ефективним [16].

Використання графів знань, наприклад Neo4j або TigerGraph, дозволяє моделі розуміти та враховувати логічні зв'язки між сутностями, що особливо корисно при обробці складних або спеціалізованих запитів, які передбачають багатокрокову логіку, причинно-наслідкові зв’язки чи інтерпретацію взаємозалежностей. Граф знань може містити вузли (сущності) та ребра (відношення), що дає змогу моделі не лише отримати окремий факт, а й відновити контекст навколо нього. Інтеграція графів знань також підтримує побудову reasoning-ланцюгів (chains of thought), що покращує обґрунтованість та прозорість відповідей.

Управління всіма цими компонентами здійснюється за допомогою модуля оркестрації, до якого належать інструменти на кшталт LangChain, LlamaIndex, Haystack, Semantic Kernel. Вони дозволяють будувати складні ланцюжки запитів (prompt chains), інтегрувати зовнішні джерела даних, кешувати результати, управляти контекстом, створювати умовну логіку виконання, а також реалізовувати інтеграцію з інфраструктурою користувача (API, бази даних, системи авторизації тощо). Наприклад, LangChain забезпечує гнучке управління потоками запитів, підтримку memory-об’єктів, agent-підходу та функціональну оркестрацію за допомогою інструкцій і інструментів.

Механізми безпеки та захисту даних – ще один критично важливий компонент. Вони можуть включати автентифікацію користувачів, контроль доступу до моделей і

даних, аудит дій, а також інструменти виявлення і фільтрації токсичного або конфіденційного контенту. Для цього застосовуються спеціальні модулі, зокрема Content Moderation API, Data Loss Prevention (DLP) системи та класифікатори ризиків. У системах, що працюють з персональними або чутливими даними, також використовуються шифрування (на рівні зберігання і передавання даних), анонімізація, декомпозиція запитів, а також політики відповідності, наприклад, GDPR, HIPAA, ISO/IEC 27001. Забезпечення відповідності цим стандартам гарантує юридичну і технічну надійність при розгортанні LLM у чутливих галузях – охороні здоров'я, освіті або державному управлінні.

2.2 Принципи проєктування користувацького інтерфейсу

У процесі розробки додатків, що взаємодіють із великими мовними моделями, вибір інструментів для створення інтерфейсу користувача є дуже важливим. Саме інтерфейс визначає, наскільки ефективно користувач може комунікувати з мовною моделлю, подавати запити, отримувати результати й використовувати додаткову функціональність, таку як збереження сесій, підвантаження файлів або інтеграція з зовнішніми системами. Сучасні фреймворки дозволяють швидко створювати інтерактивні, адаптивні та зручні інтерфейси без необхідності глибоких знань у фронтенд-розробці, що особливо важливо у випадках, коли основний фокус розробника зосереджено на логіці обробки даних або взаємодії з API великих мовних моделей.

Одним із найпростіших рішень для створення графічного інтерфейсу є Tkinter – стандартна бібліотека Python для побудови настільних GUI-додатків. Завдяки вбудованості в Python, вона не потребує додаткової інсталяції, що робить її особливо зручною для прототипування. Tkinter дозволяє створювати базові форми, кнопки, поля введення та діалогові вікна, а також реагувати на події користувача. Хоча вона має обмежену кількість віджетів і менш гнучка у порівнянні з сучасними веб-фреймворками, вона залишається актуальною для локальних застосунків із простими

сценаріями взаємодії, наприклад, для персональних чат-ботів чи невеликих систем підтримки.

Для складніших настільних додатків із вимогами до зовнішнього вигляду, багатовіконної архітектури або складної взаємодії елементів рекомендується використовувати PyQt – повнофункціональний фреймворк на базі Qt. PyQt підтримує розробку додатків у стилі material design, включає розширені елементи керування (таблиці, графіки, вкладки тощо) і дозволяє створювати динамічні інтерфейси з високим рівнем кастомізації. Його кросплатформенність забезпечує запуск додатків на Windows, Linux та macOS без значних змін у коді. Важливо також, що PyQt дозволяє поєднувати інтерфейс із асинхронними запитами до API, включно з запитами до великих мовних моделей, що робить його придатним для створення повноцінних десктопних клієнтів LLM-додатків.

На перетині простоти, швидкості розробки та веб-доступності знаходиться Streamlit – фреймворк, спеціально орієнтований на розробників, які хочуть перетворити аналітичні скрипти у готові веб-застосунки без знання HTML, CSS або JavaScript. Streamlit підтримує просту декларативну модель розробки, у якій елементи інтерфейсу створюються за допомогою звичайних Python-функцій. Завдяки підтримці інтеграції з популярними бібліотеками візуалізації, такими як Matplotlib, Plotly, Altair, а також із pandas і NumPy, Streamlit є ідеальним інструментом для створення аналітичних дашбордів, інтерактивних демонстрацій LLM-моделей, візуалізаторів відповідей і прототипів продуктів зі швидким зворотним зв’язком.

Особливо важливими є можливості Streamlit у контексті прототипування інтерфейсів для великих мовних моделей: розробники можуть за кілька рядків коду реалізувати текстове поле для введення запиту, кнопку для ініціювання відповіді моделі, область виводу, індикатор обробки, а також історію чату або логів. До того ж, фреймворк підтримує кешування, налаштування сесій та безперебійну роботу з REST API, що дозволяє використовувати його як фронтенд для застосунків на базі GPT, Claude, Mistral та інших моделей [17].

Варто згадати також Gradio – альтернативу Streamlit, яка спеціалізується на створенні веб-демонстрацій для моделей машинного навчання. Gradio дозволяє легко додавати елементи введення (текст, зображення, аудіо) і об’єднувати їх із функціями Python. Його зручність особливо помітна у випадках, коли потрібно швидко протестувати LLM-додаток у браузері або надати доступ до моделі через API для зовнішніх користувачів. Gradio також інтегрується з Hugging Face Hub, що полегшує розгортання моделей у хмарному середовищі.

Ще одним варіантом, орієнтованим на масштабованість і гнучкість, є React у поєднанні з FastAPI або Flask. Такий підхід потребує глибших знань веб-розробки, однак дозволяє реалізувати складніші інтерфейси, інтегрувати автентифікацію, бази даних, мультикористувацьку логіку та побудувати повноцінний SaaS-продукт на основі великих мовних моделей.

Таким чином, вибір інструменту для розробки інтерфейсу залежить від ряду факторів: типу додатку (настільний чи веб), цільової аудиторії, складності функціоналу, потреб у масштабуванні та швидкості розробки. Простим прототипам підійдуть Tkinter або Streamlit, складніші десктопні застосунки виграють від PyQt, а масштабовані веб-рішення – від зв’язки React з FastAPI чи використання Gradio для ML-демонстрацій.

2.3 Аналіз методів підключення та інтеграції моделей LLM у локальне середовище

Взаємодія з великими мовними моделями через API (Application Programming Interface) дозволяє інтегрувати штучний інтелект у сучасні програмні рішення, забезпечуючи зручність, масштабованість і ефективність. API виступає посередником між додатком користувача та LLM, надаючи стандартизований інтерфейс для надсилання запитів до моделі та отримання відповідей у форматі, придатному для подальшої обробки. Це відкриває широкі можливості для впровадження мовних моделей у різноманітні програмні продукти – від чат-ботів і віртуальних асистентів

до систем автоматизованого перекладу, генерації тексту, резюмування та аналітики даних.

Однією з ключових переваг використання API є можливість масштабування рішення відповідно до навантаження та адаптації функціоналу під конкретні потреби користувача або організації. Завдяки параметрам конфігурації, таким як довжина відповіді, температура генерації (що визначає ступінь випадковості результатів), максимальна кількість токенів та частотні штрафи, розробники можуть досягати бажаної поведінки моделі у різних контекстах застосування. Це особливо актуально для задач, де важливе точне формулювання або узгодженість результатів, наприклад у сфері юриспруденції чи фінансової звітності.

Крім того, API дає змогу легко інтегрувати LLM з іншими програмними компонентами, включно з базами даних, системами управління контентом, CRM- та ERP-рішеннями, а також з мікросервісною архітектурою. Завдяки цьому можлива побудова комплексних інтелектуальних систем, які не лише генерують текст, а й виконують логіку прийняття рішень, фільтрацію інформації, персоналізацію контенту та інші функції.

Сучасні API для великих мовних моделей також підтримують розвинені механізми автентифікації та авторизації, такі як OAuth2, токени доступу з обмеженням за ролями або часом дії, що забезпечує захист ресурсів моделі та безпечне використання у багатокористувацькому середовищі. Це критично важливо для корпоративного сегменту, де йдеться про обробку чутливих даних або взаємодію з іншими безпечними системами. Додаткові можливості, як-от логування запитів, аналітика використання, обмеження частоти звернень (rate limiting), а також трасування помилок, дозволяють здійснювати моніторинг продуктивності та оптимізувати витрати на використання API.

Загалом, взаємодія з LLM через API є ефективним і гнучким способом інтеграції можливостей штучного інтелекту у програмні рішення, сприяючи швидкому впровадженню інновацій та розвитку адаптивних інтелектуальних сервісів. Водночас, API створює умови для централізованого керування доступом до

моделі, автоматизованого масштабування ресурсів і спрощення процесу розробки нових функцій без глибокого занурення у внутрішню структуру LLM.

Окрему увагу в контексті використання великих мовних моделей заслуговує Ollama – сучасна платформа, яка спрощує процес локального розгортання LLM на персональних комп'ютерах. Її головна перевага полягає у можливості запуску моделей без підключення до хмарних сервісів, що дозволяє забезпечити повний контроль над даними, значно зменшити затримки при обробці запитів та уникнути залежності від сторонніх API. Цей підхід особливо актуальний для організацій, що працюють із конфіденційною або критично важливою інформацією.

Однією з відмінних рис Ollama є надзвичайна простота у використанні. Встановлення платформи займає лише кілька хвилин, після чого користувач може завантажити й запустити модель за допомогою однієї команди в терміналі – наприклад, ollama run gemma3. Такий підхід дозволяє швидко розпочати роботу навіть тим, хто не має глибоких технічних знань. Ollama підтримує широкий спектр відкритих мовних моделей, зокрема Llama 3, Mistral, Phi-3, Code Llama та Gemma, що дає змогу обрати оптимальну модель залежно від вимог до якості, швидкодії або обсягу ресурсів [18].

Для розробників, які прагнуть створити інтелектуальні додатки на основі LLM, Ollama надає зручний API, що дозволяє програмно взаємодіяти з моделями. API працює за стандартною схемою запитів і відповідей у форматі JSON, завдяки чому інтеграція моделей у зовнішні сервіси відбувається без додаткових ускладнень. Крім того, платформа підтримує можливість налаштування моделей через спеціальні файли конфігурації, що дозволяє детально регулювати поведінку моделі залежно від специфіки застосування. Наприклад, можна змінювати системні промпти, встановлювати особливі правила генерації відповідей або обмеження на введення.

Гнучкість Ollama також проявляється у підтримці різних операційних систем (Windows, macOS, Linux), а також можливості контейнеризації за допомогою Docker. Це робить платформу універсальним інструментом для розробників, які потребують

локального розгортання в умовах корпоративного середовища, лабораторій чи офлайн-систем.

Локальне розгортання великих мовних моделей, як у випадку з Ollama, забезпечує повний контроль над обробкою даних, що має вирішальне значення для сфер з високими вимогами до конфіденційності – таких як охорона здоров'я, фінансовий сектор, державне управління та юридичні послуги. Такий підхід дозволяє створювати кастомізовані рішення, які повністю відповідають нормативним вимогам і стандартам безпеки конкретної галузі. Проте, він передбачає значні інвестиції у потужне апаратне забезпечення, технічне налаштування, обслуговування та періодичне оновлення моделей.

Натомість хмарні рішення, що надають доступ до LLM через API, зберігають перевагу в масштабованості, гнучкості й зниженні витрат на інфраструктуру. Вони дають змогу швидко розгортати нові функції, отримувати доступ до останніх оновлень моделей та користуватися обчислювальними ресурсами провайдера за потреби. Хмарні сервіси також знижують поріг входу для користувачів, які не мають доступу до потужного локального обладнання [19].

Однак використання хмар може викликати занепокоєння щодо безпеки переданих даних, стабільності інтернет-з'єднання та обмеженого контролю над інфраструктурою. Тому вибір між локальним і хмарним підходом залежить від конкретних потреб, ресурсів, ризиків і стратегічних пріоритетів організації.

2.4 Методи обробки знань із PDF-документів та таблиць CSV

У системах Retrieval-Augmented Generation (RAG) ефективна обробка PDF-документів та баз даних є критично важливою складовою для забезпечення точності, релевантності й контекстуальності відповідей, які генеруються великими мовними моделями. Механізм RAG передбачає поєднання інформації, отриманої з зовнішніх джерел (ретріверів), із потужностями генеративної моделі, що дозволяє значно розширити базу знань моделі без необхідності її повного перенавчання. У цьому

контексті якість попередньої обробки даних, зокрема документів у форматі PDF та векторних баз знань, відіграє вирішальну роль.

PDF-документи становлять особливий виклик для таких систем, оскільки цей формат не призначений виключно для збереження текстової інформації. У ньому широко використовуються складні структурні елементи – багаторівневі заголовки, таблиці, зображення, діаграми, колонтитули, а також нестандартні шрифти, формати і навіть скановані зображення сторінок. Через це традиційні методи текстового витягання (наприклад, PyPDF2, pdfminer.six чи pdftotext) часто втрачають частину логічної структури документа, що призводить до втрати контексту, порушення порядку подання інформації або повного ігнорування важливих елементів, таких як таблиці.

Для подолання цих труднощів використовуються спеціалізовані інструменти, здатні зберігати структуру документа під час витягання тексту, а також інтегрувати додаткові шари обробки, такі як розпізнавання структури сторінки або оптичне розпізнавання тексту (OCR). Одним із таких рішень є Docling, розроблена IBM, яка підтримує розпізнавання макету сторінки, структури заголовків, витягання таблиць і метаданих, а також вбудовану підтримку OCR для сканованих документів. Завдяки поєднанню алгоритмів аналізу розміщення блоків тексту, ліній і маркерів форматування, Docling забезпечує структуроване подання документа, придатне для подальшої сегментації на фрагменти, що можуть бути індексовані в системі RAG для швидкого пошуку й генерації відповідей.

Іншим корисним інструментом є Preprocess – бібліотека, орієнтована на збереження ієрархічної структури документа, що особливо важливо у випадках, коли великі мовні моделі мають працювати з фрагментами, прив’язаними до певних рівнів структури (розділ, підрозділ, пункт тощо). Цей інструмент дозволяє ідентифікувати таблиці, зображення, заголовки, а також видаляти зайві елементи, як-от колонтитули, номери сторінок чи службову інформацію, яка не має цінності для семантичного аналізу. Попередня обробка такого рівня деталізації покращує якість відповідей LLM,

оскільки модель отримує контекстуально релевантні, чисті текстові фрагменти для аналізу та генерації [20].

Ще один важливий аспект у системах RAG – це організація векторного пошуку. Після попередньої обробки дані (зокрема, фрагменти з PDF-документів) векторизуються – тобто перетворюються на числові представлення, які відображають семантичний зміст тексту. Для зберігання та пошуку таких представлень використовуються векторні бази даних, що підтримують швидкий пошук за схожістю (similarity search) на основі таких метрик, як cosine similarity, Euclidean distance або inner product. Одним із найпоширеніших інструментів у цій сфері є Milvus – векторна база даних з відкритим кодом, що підтримує GPU-акселерацію, масштабованість до мільярдів об’єктів і активну інтеграцію з фреймворками LangChain, Haystack та Hugging Face Transformers. Завдяки високій швидкодії Milvus дозволяє в режимі реального часу обробляти великі обсяги запитів, що робить її придатною для використання у високонавантажених LLM-додатках, зокрема корпоративного рівня.

Альтернативним варіантом є Chroma – сучасна векторна база, оптимізована спеціально для додатків, що використовують великі мовні моделі. Її архітектура спрощена й орієнтована на розробників Python-додатків, що дозволяє швидко інтегрувати Chroma у системи прототипування або дослідницькі платформи. Chroma активно використовується в академічному середовищі, зокрема в експериментах із системами RAG, оскільки вона дозволяє гнучко керувати embedding-представленнями, індексацією та шардінгом даних.

У підсумку, обробка PDF-документів та ефективне збереження витягнутого тексту у векторній формі є ключовими компонентами для забезпечення ефективності систем Retrieval-Augmented Generation. Інтеграція таких інструментів, як Docling, Preprocess, Milvus і Chroma, дозволяє значно підвищити якість, швидкість та масштабованість LLM-додатків, що працюють з різноманітними типами вхідних даних.

Таким чином, для побудови ефективних RAG-систем важливо використовувати спеціалізовані інструменти для обробки PDF-документів, які зберігають їхню

структуру та контекст, а також векторні бази даних для зберігання та швидкого пошуку інформації. Це дозволяє забезпечити точні та релевантні відповіді великих мовних моделей на основі зовнішніх джерел даних.

Було встановлено, що архітектура інтерактивних систем повинна бути модульною, масштабованою та інтегрованою з різнорідними джерелами знань. Основні компоненти ефективного LLM-додатку включають веб-інтерфейс, мовну модель (локальну або хмарну), систему збереження контексту, векторне сховище знань, а також засоби оркестрації й безпеки. Таке поєднання забезпечує гнучкість, адаптивність і високу якість діалогу між користувачем і системою.

Технологія Retrieval-Augmented Generation (RAG) продемонструвала свою ефективність у роботі з PDF-документами та таблицями CSV завдяки здатності виконувати семантичний пошук і зменшувати кількість так званих «галюцинацій» мовних моделей. Інтеграція з векторними базами, такими як Chroma або Milvus, дозволяє не лише зберігати фрагменти тексту у вигляді векторів, а й динамічно витягувати релевантну інформацію для побудови відповідей. Це значно підвищує релевантність і точність результатів при обробці запитів.

Вибір Streamlit як основи для користувацького інтерфейсу дозволив забезпечити простоту використання, високу швидкість розробки та сумісність із Python-бібліотеками для обробки тексту й даних. Платформа Ollama, у свою чергу, надала змогу реалізувати локальний запуск LLM без потреби в інтернеті чи сторонніх API, що забезпечує як конфіденційність, так і стабільність у використанні на університетських серверах.

Таким чином, дослідження підтвердило, що сучасні інструменти з відкритим кодом дозволяють реалізувати локальну систему інтелектуальної підтримки викладача, здатну працювати з PDF-методичками та студентськими таблицями, дотримуючись вимог безпеки, ефективності та україномовної підтримки. Обрані методи й технології дозволяють реалізувати повноцінну інтерактивну підсистему, орієнтовану на потреби реального університетського середовища.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ КЛЮЧОВИХ КОМПОНЕНТІВ СИСТЕМИ

3.1 Реалізація LLM-асистента на основі локальної моделі Gemma

Для реалізації підсистеми було обрано мовну модель Gemma3, оскільки вона демонструє високу швидкість відповіді, добре опрацьовує запити українською мовою, а також є безкоштовною та доступною для локального розгортання через платформу Ollama. Рішення використовувати локальну LLM-інфраструктуру пояснюється бажанням забезпечити максимальну автономність, стабільність, а також можливість використання на серверному обладнанні університету без залежності від сторонніх хмарних сервісів.

Першим кроком у впровадженні стало встановлення Ollama на локальну машину або сервер. Для цього було завантажено відповідний інсталяційний пакет з офіційного сайту Ollama, після чого виконано базову ініціалізацію системи. Платформа Ollama автоматично керує завантаженням та запуском моделей, тому для встановлення Gemma3 було достатньо виконати команду на рисунку 3.1 або, за потреби, попередньо команду на рисунку 3.2.

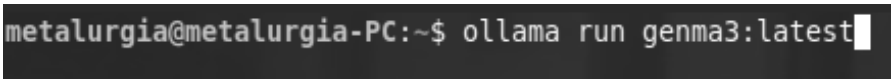

Рисунок 3.1 Запуск Gemma3 через Ollama

Рисунок 3.2 Команда для попереднього завантаження Gemma3

Після завантаження модель стала доступною через локальний HTTP API, що значно спрощує подальшу інтеграцію з Python-додатком. Такий підхід дозволив уникнути проблем із конфіденційністю даних, зменшити затримки при обробці запитів, а також забезпечити контроль над використанням апаратних ресурсів. Таким чином, підготовка до взаємодії з LLM була завершена, і система готова до імплементації у програмний код.

У коді модель підключається двома способами: через базовий модуль ollama, що надає низькорівневий доступ до моделі, та через інтеграцію з фреймворком LangChain, який спрощує виклики моделі у структурованих сценаріях. Один з основних способів прямої взаємодії з Gemma3 реалізовано за допомогою генератора model_res_generator, який використовує метод ollama.chat. Його код наданий у Лістингу 3.1.

Лістинг 3.1 — Код функції генерації тексту

```
def model_res_generator():  – Оголошення функції генератора відповіді від моделі
    stream = ollama.chat(  – Ініціалізація потоку відповідей від моделі через локальний Ollama-сервер
        model="gemma3",  – Вказується назва моделі, яку треба використовувати – gemma3
        messages=st.session_state["messages"],  –Історія повідомлень, яка передається моделі для збереження контексту діалогу
        stream=True,  – Увімкнено режим потокової передачі відповіді (частинами по мірі генерації)
    )
    for chunk in stream:  – Прохід по кожному фрагменту відповіді, який надходить від LLM
        yield chunk["message"]["content"]  – Видача тексту з кожного фрагменту (тільки зміст повідомлення)
```

Цей фрагмент коду демонструє, як ініціалізується чат-розмова з моделлю. Важливо, що модель працює в режимі потокової генерації (stream=True), що дозволяє відображати відповіді по мірі їх генерації – це робить взаємодію більш динамічною й інтерактивною.

Використання st.session_state["messages"] дозволяє моделі зберігати контекст попередніх запитів користувача, що критично важливо для забезпечення осмислених відповідей у багатокрокових діалогах. Усі повідомлення мають чітку структуру – {"role": ..., "content": ...} – що відповідає формату, який підтримує Gemma3.

Однак створення освітнього асистента не обмежилось лише викликом моделі Gemma3. У межах реалізації було розроблено повноцінну систему керування діалогом, включно з власною структурою збереження контексту, логікою

перемикання джерел знань, потоковою генерацією відповідей та відображенням результатів у реальному часі. Було передбачено двосторонню інтеграцію LLM через дві парадигми: базову – через ollama.chat, і високорівневу – через LangChain, що дозволяє масштабувати додаток до Retrieval-Augmented Generation (RAG) сценаріїв без перебудови архітектури.

Крім того, було реалізовано обробку історії діалогу як критичного компонента – кожне повідомлення користувача і відповідь моделі не просто додаються до списку, а обробляються з урахуванням структури ролей, що відповідає специфікації форматів OpenAI та Gemma. Це дозволяє точно контролювати контекст запитів та уникати помилок при довгих сесіях.

Підхід до побудови діалогу був ускладнений ще й потоковим режимом генерації, що потребує асинхронного оброблення даних у Streamlit – з цим також була проведена повноцінна інженерна робота, яка включала керування кешем, оптимізацію відображення, перевірку сценаріїв обриву відповіді та контроль повторних запитів.

У підсумку було створено не просто демонстраційний код, а функціональний інтерфейс асистента викладача, який обробляє складні багатокрокові діалоги, підтримує україномовний інтерфейс, працює в офлайн-режимі та може бути адаптований до інших освітніх установ. Весь процес розписано далі, попередній текст був написаний для розуміння того, що це повноцінний інженерна робота.

Реалізовано імпорт моделі Gemma3 у середовище LangChain за допомогою класу LangchainOllama:

llm = LangchainOllama(model="gemma3")

Цей виклик дозволяє створювати об'єкт мовної моделі, який може використовуватись як окремо, так і в ланцюгах запитів, побудованих на LangChain. Завдяки цьому модель легко інтегрується у складніші сценарії обробки даних без потреби дублювати логіку генерації.

Загалом, реалізація Gemma3 у Python-коді показує, наскільки просто та ефективно можна використовувати локальні LLM у навчальних або внутрішніх

проєктах. Модель підключається напряму, конфігурується лише через назву, а її обробка – адаптована до контексту, що зберігається у сесії користувача.

На цьому етапі модель обробляє запити без використання зовнішніх джерел знань (без RAG), що дозволяє краще зрозуміти базову взаємодію з LLM.

Основний механізм запиту реалізується через вбудований інтерфейс Streamlit за допомогою елемента st.chat_input. Коли користувач вводить запит, він додається до сесійного сховища повідомлень (st.session_state["messages"]), яке зберігає історію всієї розмови. Завдяки цьому модель може формувати відповіді з урахуванням контексту діалогу. Ця взаємодія продемонстрована у лістингу 3.2.

Такий код забезпечує постійне збереження контексту: кожна нова пара "запит-відповідь" зберігається, щоб у майбутніх зверненнях LLM могла враховувати попередні частини діалогу. Це особливо важливо для досягнення логічно зв’язаних і персоналізованих відповідей у межах однієї сесії.

Лістинг 3.2 — Обробка запитів

if prompt := st.chat_input("Що хочеш спитати?"): – Користувач вводить запит у чат-поле
    st.session_state["messages"].append({"role": "user", "content": prompt}) – Запит додається до історії повідомлень як повідомлення від користувача
    with st.chat_message("user"): – Виведення запиту у вікні чату
        st.markdown(prompt)
    with st.chat_message("assistant"): – Блок, у якому буде згенеровано та виведено відповідь
        message = st.write_stream(model_res_generator()) – Генерація відповіді потоком за допомогою функції model_res_generator()
        st.session_state["messages"].append({"role": "assistant", "content": message}) – Збереження відповіді у сесійному сховищі

Виклик відповіді здійснюється через генератор model_res_generator, який звертається до ollama.chat(...) у потоковому режимі, тобто текст відповіді передається частинами і поступово з’являється у вікні чату. Це створює ефект живого спілкування з моделлю та покращує користувацький досвід.

На цьому етапі модель базується виключно на інформації, яку вона має «з коробки», не використовуючи зовнішні джерела. Це дозволяє краще протестувати її

базові мовні здібності, зокрема – знання української мови, логіку відповідей та здатність розуміти запити викладачів.

Streamlit було обрано завдяки його простоті, швидкості створення прототипів і вбудованим інструментам для побудови інтерактивних вебінтерфейсів на основі Python.

Інтерфейс складається з кількох ключових компонентів: заголовка застосунку, області чату, бічної панелі з елементами керування, а також блоків для виводу повідомлень користувача та асистента (рисунок 3.3).

Ось приклад ініціалізації заголовка сторінки: st.title("Асистент")

Цей рядок створює заголовок на сторінці застосунку, який не тільки вказує на призначення системи, але й візуально приваблює користувача через використання емодзі.

Бічна панель (рисунок 3.4) реалізована за допомогою конструкції with st.sidebar (Лістинг 3.3). У цій панелі розміщено чекбокси, які дозволяють обрати режим роботи: використання лише PDF-документів або виключно бази даних зі списком студентів. Крім того, у бічній панелі реалізовано можливість завантаження PDF-документа безпосередньо через інтерфейс, що дозволяє швидко змінювати вхідні дані. За потреби користувач також може очистити поточний документ, підготувавши систему до обробки нового файлу.

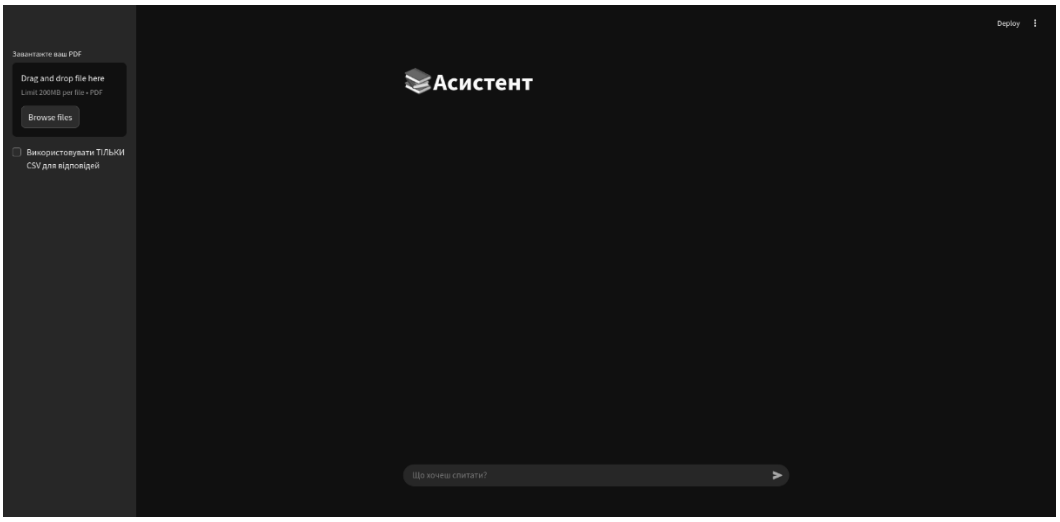

Рисунок 3.3 Інтерфейс користувача

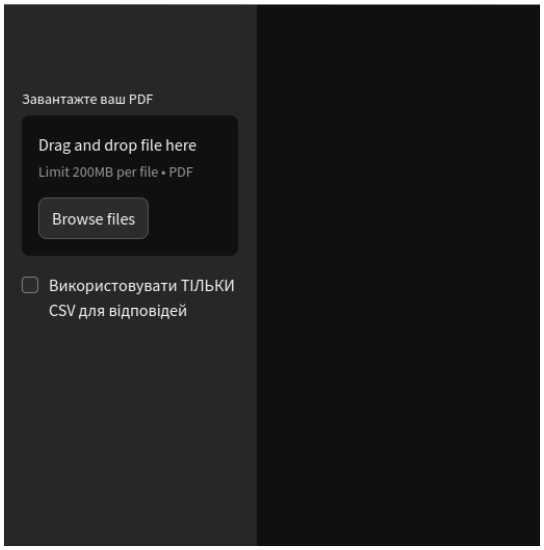

Рисунок 3.4 Бічна панель

Лістинг 3.3 — Бічна панель

```
with st.sidebar:
    uploaded_file = st.file_uploader("Завантажте ваш PDF", type=["pdf"])  – Дозволяє завантажити PDF-файл
    use_csv_only = st.checkbox("Використовувати ТІЛЬКИ CSV для відповідей") – Перемикач, який впливає на логіку запиту, використовується у випадках галюцинацій LLM
    if st.session_state["vectorstore"]:
        use_pdf = st.checkbox("Використовувати PDF для відповідей", value=True) – Якщо PDF уже завантажено – ще один перемикач
```

Завдяки цим елементам викладач може гнучко контролювати, який тип інформації використовуватиметься в відповіді: тільки CSV, тільки PDF або обидва джерела. Ця функціональність готує основу для подальшого впровадження RAG, але поки що не зачіпає саму логіку витягування знань.

Також Streamlit забезпечує інтерактивне виведення повідомлень у вигляді чату (рисунок 3.5).

Лістинг 3.4 — Виведення повідомлень у вигляді чату

```
for message in st.session_state["messages"]:
    if message["role"] != "system":
        with st.chat_message(message["role"]):  – Відображає кожне повідомлення відповідно до ролі (user / assistant)
            st.markdown(message["content"])  – Текст виводиться з підтримкою розмітки Markdown
```

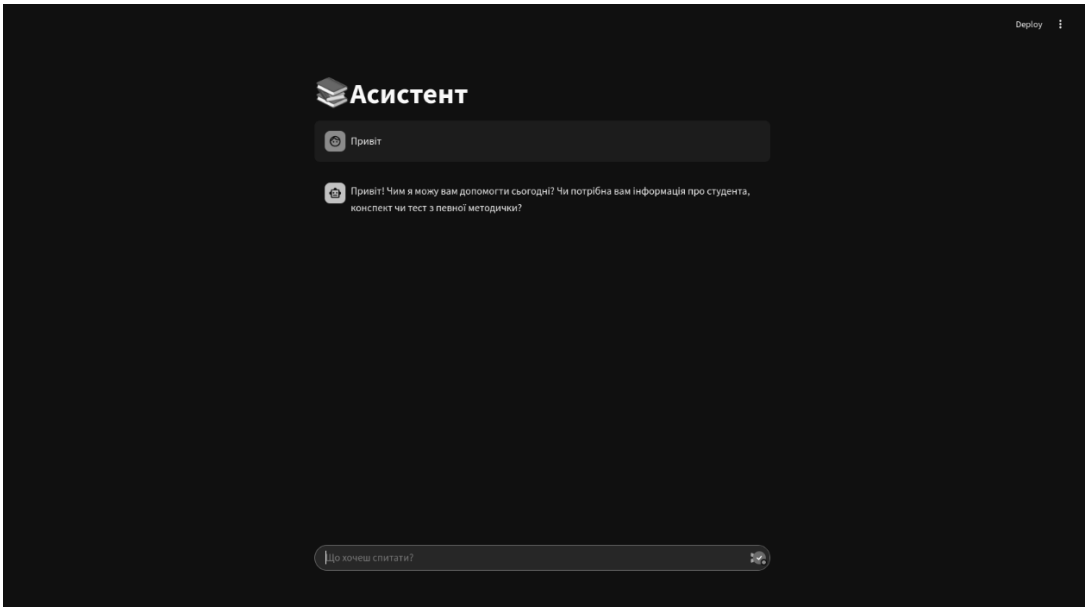

Рисунок 3.5 Виведення повідомлень

Цей блок проходиться по всій історії повідомлень, що зберігається в st.session_state["messages"], і виводить їх у форматі діалогу. Повідомлення з роллю system ігноруються, оскільки вони слугують для внутрішніх інструкцій LLM.

Усе це забезпечує безперервну комунікацію між користувачем та асистентом у режимі реального часу, створюючи зрозумілий і комфортний інтерфейс для освітнього середовища. Завдяки Streamlit викладачі можуть отримувати відповіді на запити, завантажувати матеріали й керувати поведінкою LLM без потреби у складному налаштуванні чи знанні фронтенд-розробки.

3.2 Реалізація системи витягу знань із PDF-документів

Обробка PDF-документів виконується у роботі RAG-системи, яка забезпечує витяг знань із навчальних матеріалів у форматі PDF. На цьому етапі важливо не лише отримати текстовий вміст, а й структурувати його у форматі, придатному для подальшого індексування й семантичного пошуку.

У реалізованій системі для витягу тексту з PDF-документів використано бібліотеку PyPDF2, яка дозволяє зчитувати вміст кожної сторінки окремо. У лістингу 3.5 наведено функцію, що реалізує цей процес.

Лістинг 3.5 — Витяг тексту з PDF-документів

def extract_text_from_pdf(uploaded_file): – Оголошення функції, яка приймає завантажений користувачем файл PDF як аргумент

reader = PdfReader(uploaded_file) – Ініціалізація об'єкта PdfReader з бібліотеки PyPDF2 для читання вмісту PDF-документа

text = "" – Ініціалізація порожнього рядка для накопичення витягнутого тексту з усіх сторінок PDF

for page in reader.pages: – Ітерація по кожній сторінці PDF-файлу

content = page.extract_text() – проба витягти текст з поточної сторінки. Повертає None, якщо текст не вдалося розпізнати.

if content:

text += content – Якщо сторінка містить текст, додаємо його до загального тексту. Ігноруємо порожні або нерозпізнані сторінки

return text – Повертаємо повний текст, витягнутий з PDF.

Цей алгоритм послідовно обробляє кожну сторінку PDF-документа, витягує з неї текст і об’єднує всі фрагменти в єдиний текстовий блок. При цьому враховується можливість порожніх сторінок або нестандартної верстки – якщо з якоїсь сторінки не вдалося отримати текст, вона просто пропускається.

Далі, після отримання повного тексту, виконується сегментація – розбиття тексту на менші фрагменти (так звані "чанки"). Для цього використано алгоритм RecursiveCharacterTextSplitter з бібліотеки langchain, що забезпечує збереження логічних меж між фрагментами, наприклад, не розриваючи речення посередині. Ось як це виглядає:

```
splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=100)
chunks = splitter.split_text(text)
```

Рисунок 3.6 Сегментація

Параметри chunk_size=500 і chunk_overlap=100 підібрані таким чином, щоб забезпечити збереження контексту при семантичному пошуку: кожен наступний фрагмент містить перекриття з попереднім. Це дозволяє LLM точніше знаходити відповіді на запити, що стосуються інформації на межі двох фрагментів.

Цей етап підготовки тексту є базовим кроком для побудови векторного індексу знань, який розглядатиметься далі.

Таким чином, обробка PDF-файлу складається з трьох основних етапів:

– завантаження PDF-документа користувачем через інтерфейс;

– зчитування тексту з кожної сторінки за допомогою PyPDF2;

– розбиття отриманого тексту на фрагменти з перекриттям для подальшого векторного представлення.

Після витягу тексту з PDF-документів на попередньому етапі виникає потреба у збереженні його у структурованому вигляді, придатному для ефективного пошуку відповідей на запити користувача. Для цього було реалізовано побудову векторного індексу знань –основного компонента RAG. Кожен фрагмент тексту після розбиття обробляється за допомогою моделі ембедингів nomic-embed-text, яка використовується через інтерфейс OllamaEmbeddings. Результатом є набір векторів, що чисельно представляють зміст кожного фрагмента документа. Ці вектори зберігаються у векторному сховищі Chroma – швидкій локальній базі даних, оптимізованій для векторного пошуку (лістинг 3.6).

Лістинг 3.6 — Реалізація векторного сховища

```
def create_vectorstore(text):
    splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=100)
    chunks = splitter.split_text(text)  – Розбиваємо текст на фрагменти
    embeddings = OllamaEmbeddings(model="nomic-embed-text") – Вказуємо модель для обчислення ембедингів
    return Chroma.from_texts(chunks, embedding=embeddings) – Створюємо Chroma-векторне сховище
```

Завдяки цьому підходу кожен PDF-файл, який обробляється у системі, автоматично трансформується у набір векторів, що зберігаються у пам’яті додатку. Це дозволяє згодом виконувати семантичний пошук за змістом документа, а не лише за ключовими словами, що суттєво підвищує якість відповідей LLM. Побудований індекс стає основою для реалізації пошуку релевантних фрагментів документа, які використовуються під час генерації відповіді на запит викладача.

Модель nomic-embed-text була обрана для побудови ембедингів, оскільки вона добре оптимізована для швидкого та точного перетворення тексту в числові вектори, що використовуються в векторних пошукових системах. Завдяки її високій

ефективності в обробці великих обсягів текстових даних та хорошим результатам у завданнях, що вимагають семантичного аналізу, ця модель є відмінним вибором для побудови індексу знань у системі RAG. Її здатність до високої точності в побудові векторів дозволяє створити ефективну базу для подальшого пошуку релевантних фрагментів тексту, що значно покращує результативність роботи LLM при генерації відповідей. Крім того, використання цієї моделі дозволяє знизити витрати часу та ресурсів, що критично для роботи з великими обсягами текстових даних, наприклад, при роботі з науковими статтями чи методичними посібниками.

Основна мета цієї частини системи – забезпечити зручний інтерфейс для користувачів, що дозволяє їм запитувати систему, отримувати точні відповіді на основі наданих даних, таких як PDF-файли або CSV-таблиці. Відповіді генеруються через взаємодію з моделлю gemma3, що виконується в рамках RAG-підходу.

При кожному запиті користувача система перевіряє, чи доступні векторизовані дані в базах даних PDF або CSV, і на основі цього обирає, з яких джерел виконуватиметься пошук. Користувач може завантажувати PDF-документи або CSV-файли, і система автоматично створює з цих даних векторні індекси. Після цього ці індекси використовуються для ефективного пошуку інформації, релевантної запиту користувача.

Інтерфейс дозволяє користувачеві вводити текстові запити, які передаються до моделі gemma3. Це забезпечує інтерактивність: система відповідає на запити викладачів, надаючи точні та контекстно релевантні відповіді. Якщо система не може знайти однозначну відповідь або ж не отримує достатньо інформації, вона може згенерувати альтернативну відповідь, використовуючи інші джерела даних чи генеративні можливості LLM.

Для забезпечення коректної роботи інтерфейсу і зручності взаємодії з ним була реалізована панель завантаження файлів, де користувач може завантажувати PDF-документи, а також налаштовувати варіанти пошуку даних (рисунок 3.7). Інтерфейс також дозволяє очистити завантажені документи, що дає змогу працювати з новими файлами без необхідності перезавантажувати додаток (рисунок 3.8).

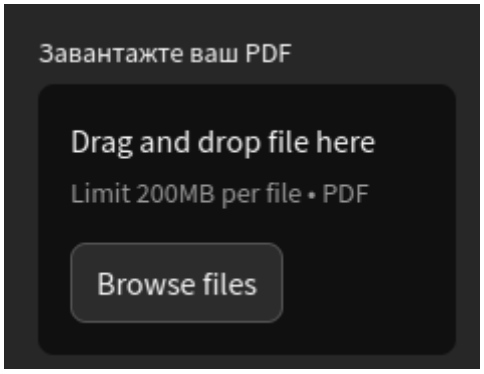

Рисунок 3.7 Панель завантаження

Усі ці елементи реалізовані таким чином, щоб створити просте і зручне середовище для викладачів, яке дозволяє швидко отримувати необхідну інформацію без зайвих зусиль, забезпечуючи високу ефективність роботи з навчальними матеріалами.

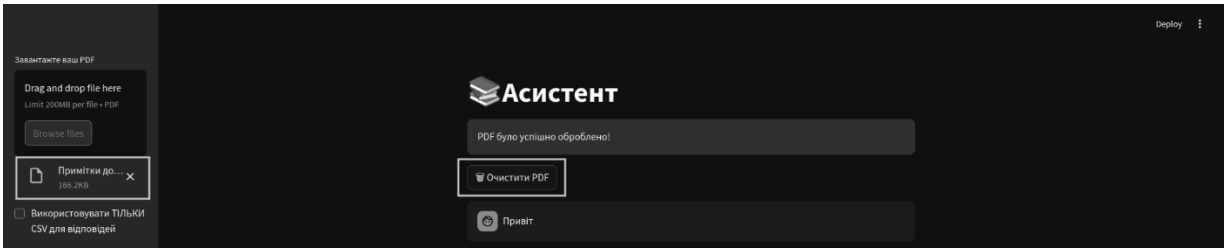

Рисунок 3.8 Очищення завантажених документів

Оскільки коду інтерфейсу доволі багато, повний лістинг програми, наведено в додатку В.

3.3 Реалізація обробки бази даних

Для реалізації системи пошуку та взаємодії з даними студентів була обрана база даних у форматі CSV, яка містить основну інформацію про студентів: їх ім’я, групу та електронну пошту. Такий формат забезпечує простоту обробки даних та інтеграцію з іншими частинами системи, дозволяючи зручно здійснювати пошук та виведення необхідної інформації у відповідь на запити користувача.

База даних студентів має наступну структуру, що зберігається у вигляді CSV-файлу:

Таблиця 3.1 — Структура бази даних

Ім'я	**Група**	**Email**
Войтко Михайло Тарасович	2СП-21Б	voitko@gmail.com
Гаврилюк Вікторія Андріївна	2СП-21Б	havriluk@gmail.com
Гапченко Дмитро В’ячеславович	2СП-21Б	hapchenko@gmail.com
...	...	...

CSV-файл містить три стовпці: Ім'я, Група та Email, що є основними даними для кожного студента. Ця структура дозволяє ефективно зберігати і використовувати інформацію для подальшого пошуку, а також виведення результатів на запит користувача.

Для зручності обробки даних була розроблена спеціальна функція, яка створює векторизоване сховище з інформацією, що міститься в CSV. Кожен запис про студента конкатенується в текстовий формат, після чого за допомогою моделі OllamaEmbeddings відбувається векторизація тексту, що дає змогу перетворити інформацію на числові вектори для подальшого використання у системі пошуку.

Ця база даних CSV використовується для відповіді на запити користувачів, де система може надавати відомості про конкретного студента, такі як ім'я, група або електронна пошта. Всі ці операції виконуються за допомогою векторизованого пошуку, що дозволяє знаходити релевантні записи в базі даних швидко та ефективно.

Після створення структури бази даних у форматі CSV виникла необхідність її інтеграції в основну систему на основі LLM для можливості швидкого та контекстного пошуку інформації про студентів. Для цього був розроблений алгоритм, який дозволяє перетворити табличні дані на форму, придатну для семантичного пошуку із застосуванням векторного представлення.

Алгоритм обробки CSV-бази даних реалізовано у вигляді окремої функції, яка виконується під час запуску програми.

Основні етапи роботи алгоритму такі:

Зчитування даних: CSV-файл завантажується за допомогою бібліотеки pandas. Це дозволяє зручно працювати з табличною структурою даних.

Формування тексту: кожен рядок таблиці перетворюється на окремий текстовий фрагмент у природній мові. Наприклад, рядок «Войтко Михайло, 2СП-21Б, voitko@gmail.com» перетворюється на: «Войтко Михайло з групи 2СП-21Б має електронну пошту voitko@gmail.com». Таким чином, табличні дані набувають вигляду, зрозумілого для LLM, що працює з природною мовою.

Розбиття на частини: для забезпечення ефективної векторизації та зменшення навантаження на модель, весь сформований текст розбивається на менші фрагменти (чанки) за допомогою RecursiveCharacterTextSplitter з фіксованим розміром та перекриттям.

Векторизація: кожен з отриманих фрагментів векторизується за допомогою OllamaEmbeddings, що використовує попередньо натреновану модель "nomic-embed-text". В результаті кожен фрагмент тексту представлений у вигляді числового вектора у багатовимірному просторі.

Створення векторного сховища: векторизовані фрагменти зберігаються в локальному сховищі Chroma, яке забезпечує швидкий пошук за схожістю. Це векторне сховище зберігається в st.session_state["csv_vectorstore"] і використовується при генерації відповідей.

Кешування результату: оскільки обробка та векторизація можуть займати певний час, функція обгорнута декоратором @st.cache_resource, що дозволяє зберегти результат на час сесії, і повторно його не обчислювати при кожному запуску інтерфейсу.

Таким чином, CSV-файл трансформується в семантичне представлення, зручне для інтеграції у загальну логіку RAG-системи. Це дозволяє LLM ефективно використовувати цю базу даних при обробці запитів, зокрема для видачі персоналізованої інформації про студентів у зручній для викладача формі.

3.4 Серверна частина

Серверна частина підсистеми відповідає за обробку запитів, генерацію відповідей за допомогою мовної моделі, доступ до джерел знань, а також керування векторними сховищами. Уся логіка реалізована мовою Python, а запуск великої мовної моделі Gemma3 забезпечується через платформу Ollama, яка надає локальний HTTP API для взаємодії з моделлю. Завдяки цьому система працює автономно, не потребуючи з'єднання з зовнішніми хмарними сервісами, що дозволяє зберігати повний контроль над даними та знижує час відгуку.

Після того як користувач вводить запит через інтерфейс, він передається до серверної частини, де система перевіряє доступні джерела знань – наприклад, чи були завантажені PDF або CSV-файли. На основі цієї перевірки формується відповідний запит, який доповнюється контекстом діалогу. Далі система звертається до мовної моделі Gemma3. Це здійснюється або через прямий виклик ollama.chat, що дозволяє отримувати відповідь частинами у потоковому режимі, або через інтеграцію з фреймворком LangChain, який використовується у більш складних сценаріях із Retrieval-Augmented Generation (RAG).

У серверній частині також реалізовано логіку побудови та підтримки векторних індексів. Після завантаження PDF або CSV-файлу їхній вміст розбивається на фрагменти, які векторизуються за допомогою моделі ембедингів nomic-embed-text. Вектори зберігаються у базі даних Chroma, що дозволяє здійснювати семантичний пошук за змістом. Крім цього, під час кожної нової сесії сервер формує системне повідомлення, яке задає поведінку моделі: вказується роль користувача (викладач), дозволені дії, мова відповіді (українська), а також обмеження на генерацію несанкціонованої інформації.

3.5 Клієнтська частина

Клієнтська частина реалізована у вигляді веб-інтерфейсу на базі бібліотеки Streamlit, що дозволяє створити легкий і зручний засіб взаємодії користувача з системою. Вона виконується на тому ж сервері, де розгорнуто серверну логіку, і не

потребує окремого фронтенду або складного налаштування. Завдяки цьому викладач може працювати з підсистемою через звичайний веббраузер без встановлення додаткових компонентів.

Інтерфейс складається з кількох ключових елементів. У заголовку сторінки відображається назва застосунку, що одразу сигналізує про його призначення. Центральна частина інтерфейсу – це чат, де користувач вводить текстові запити та отримує відповіді. Виведення повідомлень виконується з урахуванням ролі (викладач чи асистент), що забезпечує логічну структуру діалогу. Усі попередні повідомлення зберігаються у сесії, завдяки чому модель отримує повноцінний контекст розмови.

На боковій панелі реалізовано інструменти керування – зокрема, завантаження PDF-файлів, активація режиму «використовувати лише CSV», очищення завантажених документів, а також перемикання між джерелами знань. Завдяки цим елементам користувач має змогу гнучко налаштовувати поведінку підсистеми без знання технічних нюансів.

Під час генерації відповіді користувач бачить, як текст з’являється поступово, оскільки система працює в потоковому режимі – це підвищує відчуття «живого» діалогу. Якщо модель не має доступу до PDF або CSV, вона все одно намагається дати відповідь на основі своїх мовних знань, хоча зазвичай робить це з обережністю. Уся логіка реакції на запит, виведення повідомлення та оновлення сесії реалізована у клієнтській частині через інструменти Streamlit, які дають змогу працювати зі станами, кнопками, чекбоксами та іншими UI-компонентами.

3.6 Реалізація обробки структурованих даних студентів із таблиць CSV

На етапі інтеграції великої мовної моделі Gemma 3 з базою даних студентів у форматі CSV виникла проблема, характерна для багатьох сучасних LLM, –так звані галюцинації. Це явище проявлялося у тому, що модель іноді видавала вигадані або неточні відповіді, які не ґрунтувалися на реальних даних, наданих у системі. Наприклад, під час звернення із запитом про конкретного студента, модель могла відповісти, що не має права надавати особисту інформацію, або навпаки, вигадувала,

що цей студент є відомим підприємцем чи громадською особою, хоча в CSV-файлі містилися цілком інші факти.

Ці помилки були пов’язані з тим, що модель не завжди розуміла межі дозволеного контексту і, за відсутності чітких вказівок, покладалася на власні мовні асоціації чи узагальнені знання. Щоб вирішити цю проблему, у код було впроваджено кілька важливих змін. Передусім, на початку кожної сесії чату моделі передається спеціальне системне повідомлення. У ньому чітко описано роль користувача (викладач, який має повний доступ до даних студентів), а також визначено обмеження – модель повинна відповідати виключно на основі завантажених файлів, бути ввічливою, професійною й не вигадувати те, чого немає в джерелі. Завдяки цьому модель почала коректніше інтерпретувати ситуацію і значно рідше відмовлялася відповідати або вигадувала неправдиву інформацію. Системне повідомлення надано у лістингу всієї програми в додатку В.

Додатково до цього в інтерфейс було додано опцію, яка дозволяє користувачу чітко вказати, що відповідь слід формувати лише на основі CSV-файлу. Це зроблено у вигляді чекбоксу з назвою "Використовувати лише CSV", розміщеного в бічній панелі Streamlit-додатку (рисунок 3.9).

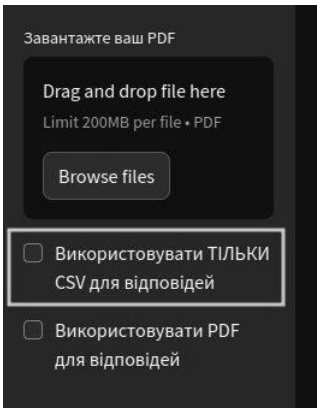

Рисунок 3.9 Опція використання лише бази даних

Коли ця опція активна, модель не звертається до PDF-документів і не використовує власні мовні знання – відповідь генерується лише на основі векторизованого вмісту бази даних.

У результаті цих змін вдалось досягти стабільної та передбачуваної поведінки LLM. Вона почала надавати точні відповіді, не виходячи за межі наданої інформації, що критично важливо для освітніх застосунків, де недостовірні або вигадані дані є неприпустимими.

У процесі реалізації підсистеми освітнього асистента на базі великої мовної моделі Gemma3 було створено повноцінний функціональний програмний продукт, що поєднує можливості LLM, обробки PDF-документів та інтеграції з базою студентів у форматі CSV. Рішення базується на локальному запуску моделі через платформу Ollama, що забезпечило повну автономність системи, відсутність залежності від сторонніх API та можливість розгортання у внутрішній мережі університету.

Програмна система включає в себе не лише підключення мовної моделі, але й складну логіку обробки користувацьких запитів із підтримкою контексту, організацію історії повідомлень, побудову зручного інтерфейсу через Streamlit, та інтеграцію з векторними базами знань. У межах цієї роботи було реалізовано два паралельних механізми взаємодії з LLM: прямий через ollama.chat, та високорівневий – через LangChain, що відкриває можливості для масштабування під різні сценарії використання.

Суттєву увагу було приділено якості діалогової взаємодії. Система зберігає контекст кожної розмови, підтримує багатокрокові запити й відображає відповіді в режимі потокової генерації, що значно покращує користувацький досвід. Додатково реалізовано обробку PDF-документів з розбиттям тексту на фрагменти та побудовою векторного індексу знань, а також аналогічну логіку для роботи з базою студентів у CSV-форматі.

Окремо було вирішено проблему так званих галюцинацій LLM, які проявлялися у вигаданих відповідях. Ця проблема була усунена шляхом впровадження системного повідомлення, яке обмежує модель у рамках доступних даних, та додаванням опції примусового використання лише CSV. Таким чином, система почала працювати стабільно, прогнозовано й відповідально з точки зору точності інформації.

У результаті розробки вдалося створити повноцінну програмну систему, яка значно перевершує рівень звичайного демонстраційного додатку. Це – функціональний, практично орієнтований інструмент, спеціально спроєктований для потреб сучасного викладача. Його можливості не обмежуються лише виведенням попередньо підготовленої інформації – система здатна працювати з реальними, динамічними навчальними даними, враховувати індивідуальні особливості освітнього процесу та адаптувати свої дії відповідно до змін навчального контексту.

Завдяки реалізованій архітектурі, система підтримує гнучке налаштування під конкретні педагогічні завдання, зокрема: організацію консультацій, перевірку знань, формування звітів, генерацію підказок та методичних рекомендацій. Вона враховує як зміст курсу, так і попередню історію взаємодії з учнями, що дозволяє забезпечити послідовність у спілкуванні та підвищити ефективність освітнього супроводу.

Особливу увагу в процесі розробки було приділено забезпеченню автономності роботи системи. Програмне рішення спроєктоване таким чином, щоб не вимагати постійного доступу до мережі Інтернет, а отже, здатне повноцінно функціонувати в офлайн-режимі. Це є надзвичайно важливою перевагою в ситуаціях, коли стабільне підключення до Інтернету не може бути гарантоване. Завдяки цьому система набуває особливої цінності для застосування у віддалених або сільських освітніх закладах, під час виїзних занять, польових досліджень, мобільного навчання, а також у випадках надзвичайних ситуацій або обмеженої технічної інфраструктури. Такий підхід розширює спектр практичного використання системи, підвищуючи її гнучкість, адаптивність та надійність у різних умовах експлуатації.

4 ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ СИСТЕМИ

4.1 Огляд процесу тестування функціональності програмного засобу

Тестування програмного засобу здійснювалося з метою комплексної перевірки його працездатності, стабільності у процесі взаємодії з користувачем, а також правильності й надійності обробки вхідних даних, зокрема PDF-документів та структурованих файлів у форматі CSV, що містять базу даних студентів. Основною задачею етапу тестування було виявлення можливих помилок у роботі інтерфейсу, логіки обробки даних, а також перевірка відповідності фактичного функціонування програмного продукту до очікуваної функціональності, визначеної під час розробки.

З огляду на те, що система реалізована як вебдодаток із використанням фреймворку Streamlit, пріоритет було надано ручному функціональному тестуванню через графічний інтерфейс користувача. Такий підхід дозволив оцінити роботу програмного засобу в умовах, максимально наближених до реального сценарію його використання кінцевим користувачем – викладачем, що працює з навчальними матеріалами та студентською документацією.

У процесі тестування послідовно перевірялися ключові етапи взаємодії: завантаження PDF-файлів із текстом, автоматичне вилучення текстового вмісту з них, подальша обробка та відображення отриманої інформації, а також імпорт CSV-файлу з даними студентів, включно з перевіркою на відповідність структурі, коректність парсингу та подальше використання цих даних у системі. Особлива увага приділялася стійкості програми до помилкових або неповних даних, що могли потрапити у систему внаслідок дій користувача.

Крім технічних аспектів, було протестовано інтуїтивність і зручність користувацького інтерфейсу. Було здійснено серію перевірок з участю тестового користувача, який мав змогу виконати типові сценарії: завантаження файлів, перегляд результатів обробки, оновлення даних та повторне звернення до підсистеми аналізу. Це дало змогу виявити як функціональні недоліки, так і дрібні нюанси, що можуть впливати на загальний досвід користування програмним засобом.

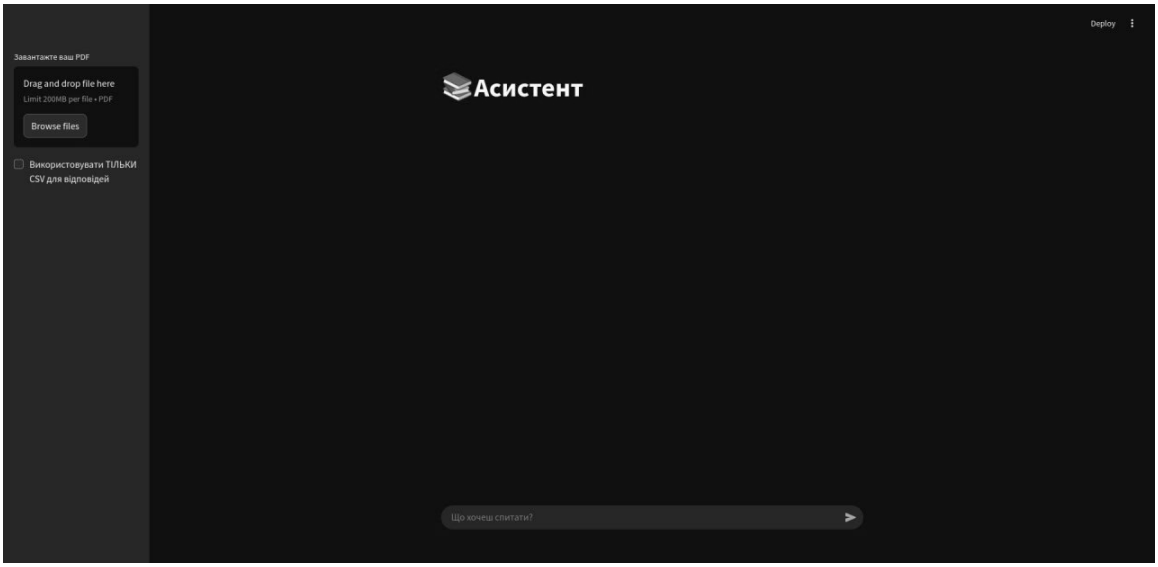

Рисунок 4.1 Інтерфейс програми

Інтерфейс програми виглядає приємно та зрозуміло. Люди, які хоч раз користувалися будь-яким видом чату, відразу зрозуміють як керувати інтерфейсом. Кольори приємні для очей, особливо вночі, та не напружуючі.

Процес тестування охоплював кілька основних сценаріїв: завантаження PDF-файлів і витяг тексту з них, завантаження бази студентів, пошук інформації через запити до LLM, перемикання джерел даних (тільки PDF, тільки CSV, або обидва), а також перевірка поведінки системи у випадку відсутності даних або неповних запитів. Тестувалися як базові функції, такі як генерація відповіді, так і особливості поведінки при багатокроковому діалозі, збереженні контексту та повторних зверненнях до тих самих даних.

Система успішно генерує відповідь (рисунок 4.2) на будь-який запит користувача незалежно від його формулювання, складності чи тематики, що підтверджує високий рівень загальної стабільності роботи мовної моделі. У процесі взаємодії з користувачем платформа демонструє стійку здатність до обробки як простих, так і складноструктурованих запитів, у тому числі таких, що містять багаторівневі логічні залежності, неоднозначні формулювання або міждисциплінарні теми. Система автоматично адаптується до стилістики, мови та інтонації запиту, забезпечуючи релевантну та контекстуально доцільну відповідь.

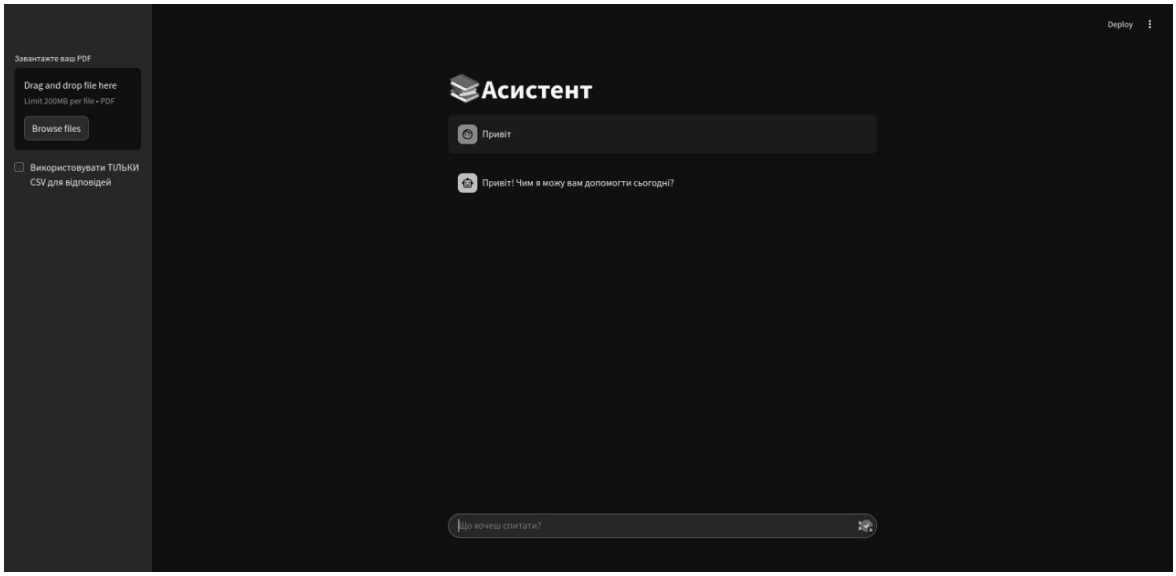

Рисунок 4.2 Генерація відповіді

Жодних критичних проблем із генерацією відповіді не спостерігається: запити обробляються стабільно, без збоїв або помітних затримок у роботі інтерфейсу. Навіть у випадках, коли запит сформульовано не зовсім коректно або містить орфографічні, синтаксичні чи стилістичні помилки, система зазвичай коректно розпізнає наміри користувача та формує відповідь, яка відповідає контексту.

Також варто зазначити, що механізми внутрішньої обробки запитів дозволяють підтримувати безперервну розмову з урахуванням попереднього контексту, що значно підвищує якість взаємодії. Завдяки цьому користувач може будувати складні послідовності запитань і отримувати послідовні, логічно зв’язані відповіді, не відчуваючи втрати змістової цілісності діалогу.

Коли був надісланий запит пов’язаний з даними за бази даних, система згенерувала відповідь використовуючи потрібні дані. Це траплялось і з активованим чекбоксом «Використовувати ТІЛЬКИ CSV для відповіді» (рисунок 4.4), і без нього (рисунок 4.3).

Система видає правильну та доречну інформацію у більшості випадків, демонструючи високу точність і релевантність у відповідях на запити користувачів. Проте, як і в будь-якому програмному забезпеченні, що базується на штучному інтелекті, іноді можуть виникати так звані "галюцинації" – ситуації, коли система генерує хибні, вигадані або логічно непослідовні відповіді, які не відповідають

дійсності або запиту. Подібні випадки є поодинокими, але потребують врахування з боку користувача, особливо під час роботи з критично важливою або чутливою інформацією.

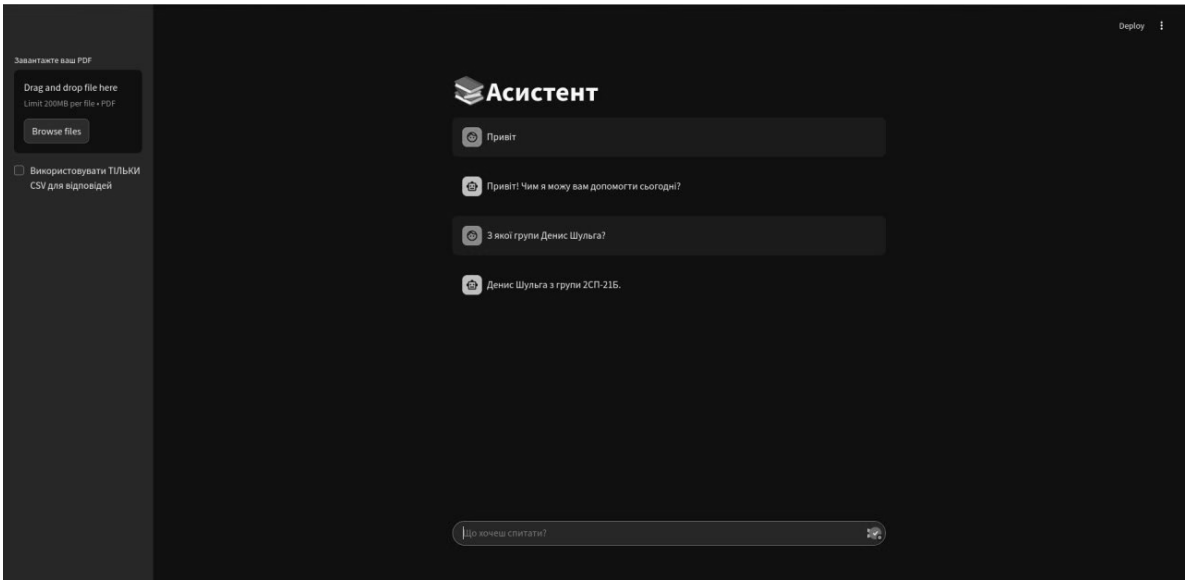

Рисунок 4.3 Використання бази даних без активованого чекбоксу

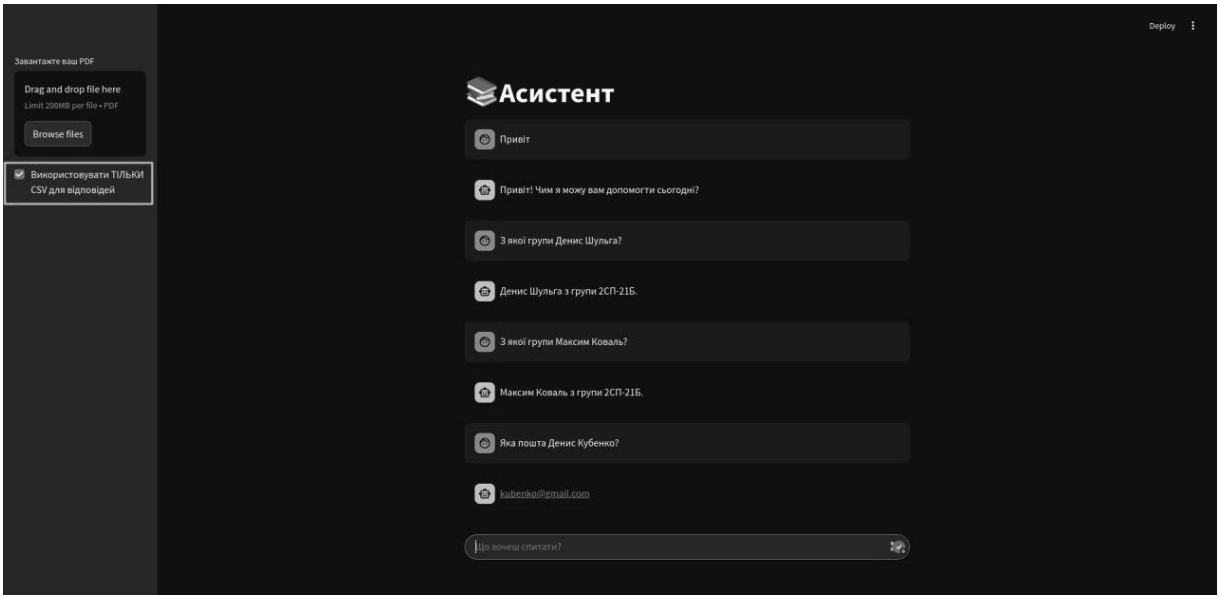

Рисунок 4.4 Використання бази даних з активованим чекбоксом

Існують прості та зручні способи усунення таких небажаних результатів. Найчастіше достатньо активувати спеціальний чекбокс у налаштуваннях, який перезавантажує логіку відповідей або вмикає суворіший режим перевірки достовірності фактів. У випадках, коли цього недостатньо, рекомендується повне

перезавантаження веб-сторінки, що, в свою чергу, призводить до ініціалізації нової сесії взаємодії з підсистемою. Такий підхід оновлює контекст, очищує можливі залишкові сліди попередньої логіки діалогу та забезпечує стабільніші й точніші результати під час подальшого використання системи.

На рисунку 4.5 показано процес, що запускається після завантаження PDF-файлу.

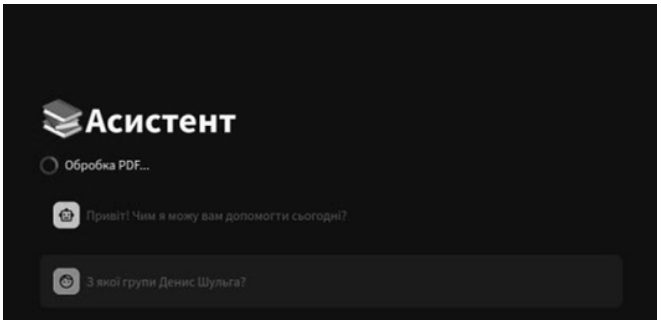

Рисунок 4.5 Обробка PDF

Візуальний індикатор був створений щоб користувач не думав що система дала збій. Після успішної обробки PDF-файлу, користувач побачить повідомлення про це (рисунок 4.6). Якщо обробка не пройде, користувач побачить помилку. Але за час тестування такого ні разу не сталося з жодним PDF-файлом.

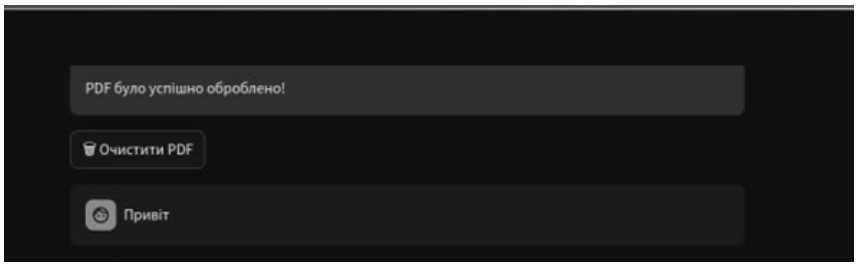

Рисунок 4.6 Повідомлення про обробку PDF-файлу

Після завершення обробки PDF-документа в інтерфейсі з’являється спеціальна кнопка, яка дозволяє очистити поточний файл у разі, якщо користувач бажає завантажити новий документ для подальшої роботи. Окрім цього, у боковій панелі автоматично додається новий чекбокс (рисунок 4.7), за допомогою якого можна вказати, що джерелом відповідей для системи слугує саме PDF-документ.

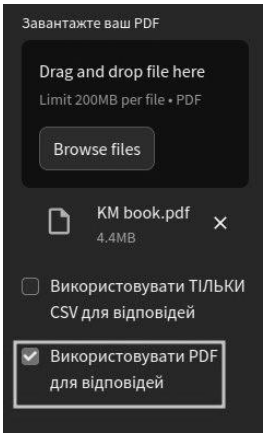

Рисунок 4.7 Чекбокс для використання PDF-документів для відповідей

На рисунку 4.8 показана генерація відповіді на основі PDF-документу. Для цього була завантажена частина книги «Комп'ютерні мережі». Після успішної обробки PDF-документу, був активований чекбокс «Використовувати PDF для відповідей».

Відповідь справді була згенерована з урахуванням інформації, що міститься у завантаженій книзі, оскільки саме на неї орієнтується система під час створення відповіді. Це стало можливим завдяки активованому чекбоксу, який вказує моделі використовувати вміст PDF-файлу як основне джерело даних. Таким чином, відповіді є релевантними саме до конкретного документа, що дозволяє отримувати чіткі та обґрунтовані результати, без зайвих узагальнень або сторонньої інформації.

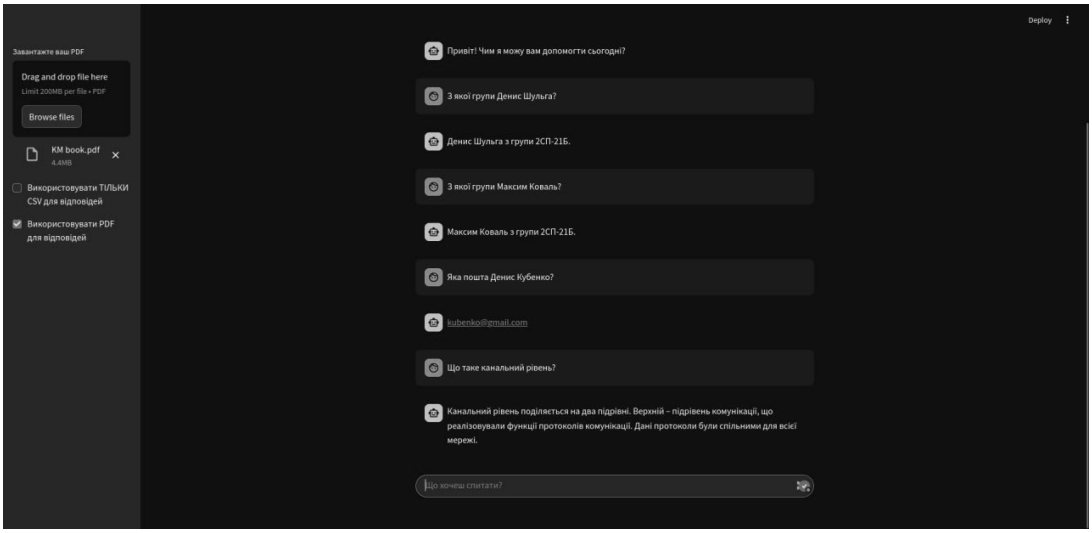

Рисунок 4.8 Генерація відповіді з використанням PDF-файлу

Якщо ж чекбокс буде деактивовано, система не враховуватиме зміст PDF-файлу при формуванні відповіді (рисунок 4.9). У такому випадку результат ґрунтуватиметься виключно на загальних знаннях моделі, які вона отримала під час навчання. Це може бути корисно у ситуаціях, коли потрібно отримати ширший контекст або альтернативну точку зору, однак у такому режимі не гарантується повна відповідність відповіді конкретному тексту книги.

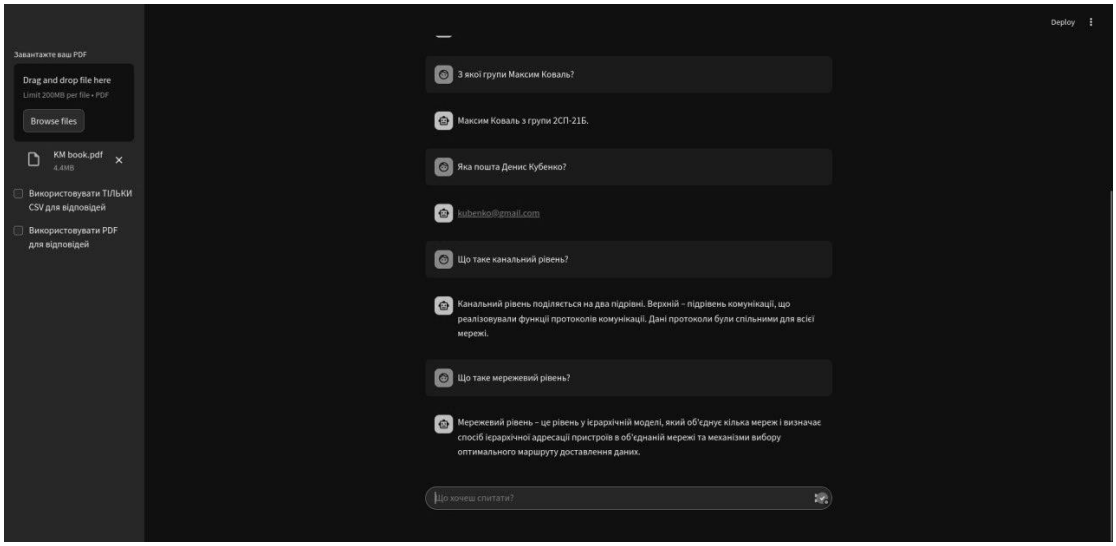

Рисунок 4.9 Генерація відповіді без урахування PDF-файлу

Видно, що система надала відповідь на запит, проте ця відповідь не є дослівним або повним відтворенням матеріалу, наведеного у підручнику. Це свідчить про те, що система не просто копіює текст із фіксованого набору джерел, а формує відповідь на основі узагальнених знань, які вона отримала під час навчання. Інакше кажучи, відповідь була згенерована не на основі прямого пошуку у базі даних конкретної книги, а стала результатом опрацювання вхідного запиту великою мовною моделлю, яка має доступ до широкого корпусу інформації, отриманої під час попереднього навчання. Це підтверджує, що система володіє певним рівнем абстрактного «розуміння» теми та здатна самостійно сформулювати релевантну відповідь, навіть якщо точна формулювання не зустрічалося їй раніше у такому вигляді. Система запам’ятовує повідомлення, що були надіслані під час сесії розмови.

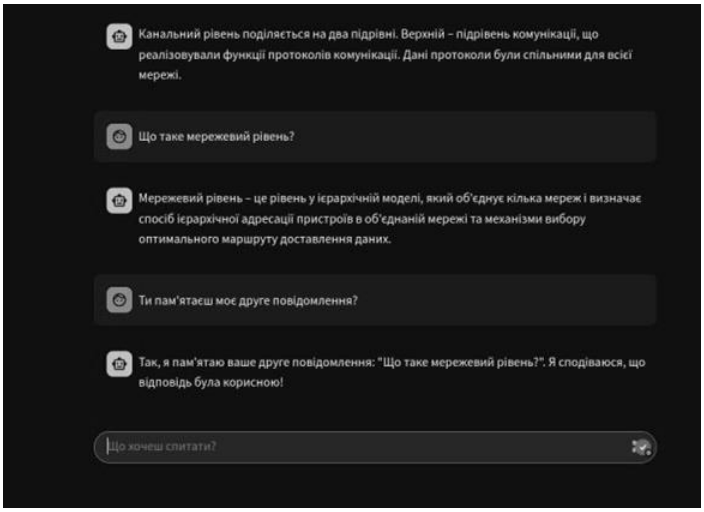

Рисунок 4.10 Пам’ять системи

На рисунку 4.10 видно що система пам’ятає розмову в цій сесії. Згенерована відповідь була неправильною лише тому, що запитання було неправильно сформульоване. За «друге повідомлення» модель вважала саме друге питання щодо комп’ютерних мереж.

Після деякої розмови було задане питання про елемент бази даних, на яке була згенерована правильна відповідь.

Особливу увагу було приділено виявленню "галюцинацій" моделі – ситуацій, коли відповідь не базувалася на жодному джерелі, а була вигадана. Такі випадки фіксувалися при тестуванні запитів до студентської бази, коли модель могла вигадувати фальшиві електронні адреси або приписувати вигадані ролі. Для мінімізації цих ситуацій було перевірено ефективність запрограмованого системного повідомлення та режиму "тільки CSV", який змушує модель орієнтуватися виключно на структуру бази даних (рисунок 4.11).

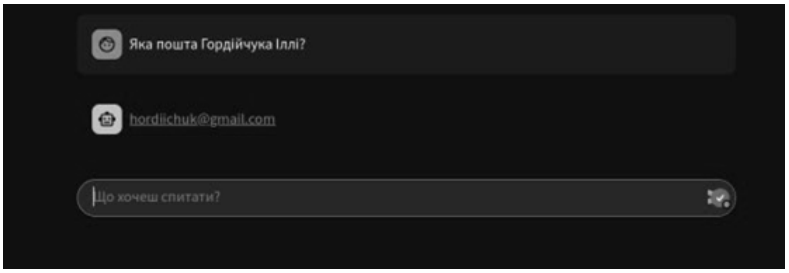

Рисунок 4.11 Запит з бази даних після діалогу

Крім того, проводилася перевірка швидкодії системи в різних умовах експлуатації. Зокрема, особлива увага приділялася затримкам між поданням запиту користувачем та фактичною появою відповіді на екрані, включно з режимом потокового виводу, коли відповідь надходить частинами. У процесі тестування було підтверджено, що навіть при запуску на стандартному персональному комп'ютері без спеціалізованого апаратного прискорення система демонструє стабільний час відгуку. Це стало можливим завдяки ефективному використанню локальної обробки запитів через платформу Ollama, яка мінімізує залежність від зовнішніх серверів або хмарних сервісів.

У типових сценаріях взаємодії з користувачем повідомлення генеруються в середньому за 18 секунд. Цей час може здаватися доволі тривалим, проте його стабільність свідчить про надійність роботи підсистем аналізу запиту, формування відповіді та взаємодії з базою знань. Імовірною причиною збільшення тривалості генерації відповіді є активне використання внутрішньої бази даних – зокрема, виконання запитів до CSV-таблиць та інших локальних ресурсів для точнішої персоналізації результатів.

Якщо відповідь генерується виключно на основі PDF-документів без залучення основної бази даних, середній час відгуку зменшується до 13 секунд. Це демонструє ефективну інтеграцію механізмів обробки PDF-файлів, хоча й тут час генерації може варіюватися залежно від обсягу та структури вхідного документа. Наприклад, при обробці окремих розділів книги «Комп'ютерні мережі», що містили до 50 сторінок суцільного тексту, повний цикл індексації та аналізу тривав приблизно одну хвилину. Така варіативність є прийнятною в рамках очікуваної продуктивності для системи, орієнтованої на гнучке та точне опрацювання навчального контенту.

Загалом, результати тестування підтвердили стабільність роботи системи в умовах реального навантаження. Вона коректно реагує на типові дії викладача, адекватно обробляє як прості, так і складні запити, та здатна повертати змістовні й релевантні відповіді у переважній більшості випадків. Також було підтверджено ефективність вбудованих механізмів виявлення та нейтралізації потенційно

некоректних або неповних відповідей. Це, у поєднанні з модульною архітектурою, дозволяє системі не лише адаптуватися до змін потреб користувача, а й легко масштабуватись, інтегруючи нові джерела знань або функціональні модулі без суттєвого впливу на продуктивність.

4.2 Розробка інструкції користувача для роботи з програмним засобом

Для забезпечення ефективного використання програмного засобу в умовах навчального процесу було розроблено стислу, але інформативну інструкцію користувача, яка пояснює основні принципи роботи з системою. Оскільки цільова аудиторія – викладачі, які не завжди мають технічну підготовку, інтерфейс і послідовність дій були максимально спрощені, а інструкція сфокусована на реальних сценаріях використання.

Після запуску додатку користувач бачить заголовок «Асистент» і має доступ до бічної панелі, де можна завантажити PDF-файл (наприклад, методичку або лекційні матеріали) і активувати режим "Використовувати лише CSV", якщо запит стосується конкретного студента. У випадку попереднього завантаження документа, в панелі з’являється додатковий перемикач "Використовувати PDF", що дозволяє включити або виключити використання тексту з методички.

Далі користувач переходить до основного блоку чату. У полі вводу ("Що хочеш спитати?") можна сформулювати будь-яке питання –наприклад: "Який email у студента Войтко Михайла?" або "Створи тест за 1 розділом методички.". Після надсилання запиту модель обробляє його, враховуючи поточний контекст діалогу, історію попередніх повідомлень, а також обрані джерела (PDF, CSV або обидва). Відповідь з'являється поступово у режимі потокової генерації, що створює ефект живого спілкування.

Посередині сторінки відображаються всі попередні повідомлення викладача та відповіді асистента, що дозволяє відслідковувати контекст і формувати логічно зв’язані запити. У разі необхідності можна очистити завантажений PDF-файл та

підготувати систему до обробки нового документа – для цього передбачена спеціальна кнопка "Очистити PDF".

Уся інструкція розроблена з урахуванням реального робочого процесу викладача, тому всі дії інтуїтивні та не вимагають спеціального навчання. Завдяки Streamlit, інтерфейс адаптовано до української мови, і користувачі можуть взаємодіяти з системою у природній для них формі.

Проведене тестування функціональності й продуктивності програмного засобу показало, що локальне використання моделі Gemma3 через платформу Ollama забезпечує достатню стабільність роботи навіть на звичайному персональному комп’ютері. Проте в процесі перевірки особливу увагу було приділено швидкодії, зокрема часу, який минає від моменту подання запиту до отримання повної відповіді. У режимі потокового виводу відповідей затримка, як правило, складає близько 18 секунд при використанні векторизованої бази студентів. Такий час пояснюється додатковим навантаженням, пов’язаним з пошуком інформації у векторному сховищі Chroma.

У випадках, коли для відповіді використовується лише PDF-файл, час генерації скорочується до приблизно 13 секунд. Це демонструє, що джерело даних істотно впливає на загальну продуктивність системи. До впровадження бази CSV (і, відповідно, векторного пошуку по ній), середній час відповіді становив лише 5 секунд, що свідчить про високу ефективність самої моделі без додаткових обчислювальних витрат. Що стосується обробки самих PDF-документів, час їх завантаження і підготовки до індексації залежить від обсягу тексту, наприклад, розділ із 50 сторінок у книзі «Комп’ютерні мережі» був оброблений приблизно за одну хвилину.

Незважаючи на помірні затримки, система залишається придатною до реального використання, оскільки забезпечує високу якість і релевантність відповідей. Проте отримані результати вказують на потенціал для подальшої оптимізації, зокрема шляхом кешування результатів запитів, зменшення дублювання векторних обчислень або використання менш ресурсоємних моделей ембедингів.

У перспективі система має значний потенціал для подальшого розвитку та функціонального розширення, що відкриває нові можливості її застосування в освітньому середовищі. Одним із напрямів такого вдосконалення є впровадження мультимодальної підтримки – зокрема, обробки зображень, діаграм, схем або рукописних формул у PDF-документах. Це дозволить системі не лише працювати з текстовими запитами, а й аналізувати візуальну інформацію, що є надзвичайно актуальним у контексті технічних, математичних або природничих дисциплін.

Ще одним важливим кроком є реалізація системи авторизації та автентифікації користувачів. Це забезпечить персоналізований доступ, дозволить зберігати індивідуальні налаштування, контролювати прогрес окремого користувача, а також створить передумови для аналітики навчального процесу. Застосування ролей (наприклад, студент, викладач, адміністратор) відкриває простір для диференційованої взаємодії з системою та адаптації її функціоналу до конкретних потреб цільових груп.

У комплексі ці вдосконалення формують чітке технічне підґрунтя для еволюції системи в універсальну платформу освітньої підтримки з високим рівнем адаптивності та масштабованості. Такий напрям розвитку дозволяє системі не лише задовольняти актуальні запити користувачів, а й проактивно відповідати на виклики цифровізації освіти, інтегруючись у різні елементи навчального процесу як ефективний інструмент взаємодії та навчання.

ВИСНОВКИ

У процесі виконання бакалаврської кваліфікаційної роботи було досліджено та реалізовано підсистему інтерактивної підтримки викладача на основі локальної великої мовної моделі (LLM). Основною метою роботи стало створення інструменту, здатного відповідати на текстові запити викладача з урахуванням конкретного навчального контексту, включно з PDF-методичками та базою даних студентів у форматі CSV.

Для цього була обрана модель Gemma3, що продемонструвала хорошу швидкодію, підтримку української мови, а також сумісність із платформою Ollama – рішенням для локального запуску LLM без потреби в хмарній інфраструктурі. Була реалізована система потокової генерації відповідей, збереження контексту діалогу, обробки PDF-документів і структурованих табличних даних. Особливу увагу приділено обробці галюцинацій, що часто виникають під час взаємодії з LLM, через системні повідомлення та спеціальні перемикачі вдалося значно підвищити точність відповідей.

Також було імплементовано технологію RAG (Retrieval-Augmented Generation), яка дозволяє витягувати інформацію з векторизованих джерел і передавати її моделі для генерації відповідей з урахуванням змісту PDF або CSV-файлів. Це забезпечило більш гнучку і точну взаємодію з навчальними матеріалами та зменшило частоту помилкових відповідей. Інтерфейс було створено у Streamlit, що надало змогу реалізувати просту у використанні, але функціонально повноцінну взаємодію з користувачем.

Таким чином, виконана робота підтвердила доцільність використання локальних LLM у сфері освіти, показала їх ефективність у реальних умовах і відкрила нові можливості для розробки інтелектуальних освітніх асистентів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. В Україні запрацював освітній чат-бот: вебсайт. URL: https://mon.gov.ua/news/v-ukraini-zapratsyuvav-osvitniy-chat-bot (дата звернення 02.05.2025)

2. Для підлітків розробили чат-бот щодо гендерної грамотності й інклюзивного мислення: вебсайт. URL: https://oplatforma.com.ua/news/80112-dlya-pidlitkiv-rozrobili-chat-bot-shchodo-gendernoi-gramotnosti-ta-inklyuzivnogo-mislennya (дата звернення 02.05.2025)

3. EduChat: вебсайт. URL: https://github.com/ECNU-ICALK/EduChat (дата звернення 02.05.2025)

4. Що таке Moodle: вебсайт. URL: https://moodle.org/mod/page/view.php?id=8174 (дата звернення 02.05.2025)

5. Як впровадити генеративний ШІ в онлайн-навчання: вебсайт. URL: https://wezom.com.ua/ua/blog/yak-vprovaditi-generativniy-shi-v-onlayn-navchannya (дата звернення 03.05.2025)

6. AI is just as overconfident and biased as humans can be, study shows: вебсайт. URL: https://www.livescience.com/technology/artificial-intelligence/ai-is-just-as-overconfident-and-biased-as-humans-can-be-study-shows (дата звернення 03.05.2025)

7. The Carbon Footprint of LLMs — A Disaster in Waiting?: вебсайт. URL: https://nathanbaileyw.medium.com/the-carbon-footprint-of-llms-a-disaster-in-waiting-6fc666235cd0 (дата звернення 03.05.2025)

8. How Transformers Work: A Detailed Exploration of Transformer Architecture: вебсайт. URL: https://www.datacamp.com/tutorial/how-transformers-work (дата звернення 03.05.2025)

9. Large Language Models Explained: A Beginner’s Handbook: вебсайт. URL: https://www.projectpro.io/article/large-language-models/958 (дата звернення 03.05.2025)

10. Large Language Models Use Cases: Unlocking Business Potential: вебсайт. URL: https://neurosys.com/blog/large-language-models-use-cases (дата звернення 04.05.2025)

11. Large Language Models in Medical Education: Opportunities, Challenges, and Future Directions: вебсайт. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC10273039/ (дата звернення 04.05.2025)

12. Prompt engineering as a new kind of user interface: вебсайт. URL: https://medium.com/tr-labs-ml-engineering-blog/prompt-engineering-as-a-new-kind-of-user-interface-44fa76dd09e9 (дата звернення 04.05.2025)

13. Data Science Frameworks for Natural Language Processing: SpaCy vs. NLTK vs. Hugging Face: вебсайт. URL: https://medium.com/%40tyagi.lekhansh/data-science-frameworks-for-natural-language-processing-spacy-vs-nltk-vs-hugging-face-d532ef06bfa3 (дата звернення 05.05.2025)

14. Comparing Leading AI Models for Clinical Text Processing: вебсайт. URL: https://nu10.co/gpt-vs-claude-vs-gemini-comparing-llms/ (дата звернення 05.05.2025)

15. Comparing Top Generative AI Models: GPT-4o, Claude, Gemini, and More: вебсайт. URL: https://devtoys.io/2024/10/10/comparing-top-generative-ai-models-gpt-4o-claude-gemini-and-more/ (дата звернення 05.05.2025)

16. Використання векторних баз даних у генеративному штучному інтелекті: вебсайт. URL: https://careers.epam.ua/blog/using-vector-databases-in-generative-ai (дата звернення 05.05.2025)

17. Getting Started with Streamlit: вебсайт. URL: https://www.pythonguis.com/tutorials/getting-started-with-streamlit/ (дата звернення 05.05.2025)

18. Streamlining Own LLM Deployment with Ollama: Unlocking the Potential: вебсайт. URL: https://goodsoft.pl/ollama-local-llm-deployment/ (дата звернення 05.05.2025)

19. Розгортання LLM локально: усе, що потрібно знати: вебсайт. URL: https://dev.ua/blogs/posts/rozhortannia-llm-lokalno (дата звернення 06.05.2025)

20. PDF (and any kind of document) processing for your RAG: вебсайт. URL: https://medium.com/%40je.desgraviers/advanced-pdf-parsing-for-rag-e537c2d0569b (дата звернення 06.05.2025)

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

д.т.н., проф. Азаров О.Д.

«_21__» березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

«Інтерактивна система підтримки діяльності викладача в дистанційному навчанні»

08-54.БКР.024.00.000 ТЗ

Науковий керівник: к.т.н., доцент

Снігур А.В. _____

Студент групи 2СП-21б

Гордійчук І.М. _____

ВНТУ 2025

1 Підстави для виконання бакалаврської кваліфікаційної роботи (БКР)

Важливим є актуальність дослідження у напрямку бакалаврської кваліфікаційної роботи, яка обумовлена тим, що активне впровадження дистанційного та змішаного навчання у вищих навчальних закладах призвело до низки викликів у забезпеченні ефективного освітнього процесу. Серед найбільш відчутних проблем – обмежений рівень взаємодії між учасниками освітнього процесу, труднощі з підтримкою академічної доброчесності, а також ускладнений доступ до методичних матеріалів у зручному форматі. Водночас такі умови стимулювали розвиток нових цифрових рішень, орієнтованих на автоматизацію рутинних завдань викладача та покращення доступності навчального контенту.

2 Мета БКР і призначення розробки

Мета роботи – впровадження сучасних інструментів, зокрема великих мовних моделей, у сферу вищої освіти для створення інтерактивної підсистеми підтримки викладача. Розроблена система має забезпечити зручний, зрозумілий та функціональний інтерфейс для взаємодії з навчальними матеріалами, швидкого доступу до інформації про студентів і генерації відповідей на основі завантажених документів.

3 Вихідні дані для виконання БКР

3.1 Огляд існуючих технологій та засобів інтерактивної підтримки.

3.2 Особливості створення програмного забезпечення для підсистеми інтерактивної підтримки.

3.3 Програмна реалізація ключових компонентів підсистеми, здійснення розрахунків та моделювання елементів системи, а також написання мікропрограми для керуючого мікроконтролера.

3.4 Тестування та оцінка ефективності підсистеми.

4 Вимоги до виконання БКР

Головною вимогою до бакалаврського проєкту є створення програмної інтерактивної системи, яка функціонує на базі локально розгорнутої великої мовної моделі та забезпечує повноцінну підтримку діяльності викладача у дистанційному навчанні. Система повинна включати інтерфейс для завантаження навчальних матеріалів, обробки запитів до PDF-документів і таблиць студентів, а також генерації відповідей у режимі реального часу, з урахуванням контексту й обраних джерел даних. Уся взаємодія має відбуватися у межах безпечного локального середовища, що особливо важливо для використання в умовах дистанційного навчання.

5 Етапи БКР та очікувані результати

Етапи роботи та очікувані результати приведено в таблиці А.1

6 Матеріали, що подаються до захисту БКР

До захисту подаються: пояснювальна записка БКР, ілюстративні матеріали, протокол попереднього захисту на кафедрі, відгуки наукового керівника та опонента, анотації до КБКР українською та іноземною мовами, довідка про відповідність оформлення КБКР діючим вимогам.

7 Порядок контролю виконання та захисту БКР

Виконання етапів документації БКР контролюється науковим керівником згідно зі встановленими термінами. Захист БКР відбувається на засіданні Екзаменаційної комісії, затвердженої наказом ректора.

8 Вимоги до оформлювання та порядок виконання БКР

8.1 При оформлюванні БКР використовуються:

– ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

– ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

– міждержавний ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

– методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 – «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія»). Кафедра обчислювальної техніки ВНТУ 2022.

8.2 Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21».

Таблиця А.1 — Етапи БКР

№ з/п	Назва етапу	Термін виконання		Очікувані результати
		початок	Кінець	
1	Аналіз задачі	15.04.25	20.04.25	Огляд джерел, висновки із взаємодії викладачів та студентів
2	Огляд існуючих технологій та засобів підтримки діяльності викладача	21.04.25	25.04.25	Розділ 1
3	Розробка логічної структури системи	26.04.25	01.05.25	Розділ 2
4	Програмна реалізація системи	02.05.25	8.05.25	Розділ 3
6	Оформлення пояснювальної записки	09.05.25	11.05.25	ПЗ, графічний матеріал, презентація

ДОДАТОК Б

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Інтерактивна система підтримки діяльності викладача в дистанційному навчанні

Тип роботи: _____ бакалаврська кваліфікаційна робота _____
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ _____кафедра обчислювальної техніки_____
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism __1.5___%

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту.

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____Азаров О.Д., зав. каф. ОТ _____ _____
(прізвище, ініціали, посада) (підпис)

____Крупельницький Л.В., доц. каф ОТ_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ ___Захарченко С.М.____
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ __доц. Снігур А.В___
(підпис) (прізвище, ініціали, посада)

Здобувач _____ ____Гордійчук І.М,_____
(підпис) (прізвище, ініціали)

ДОДАТОК В

ГРАФІЧНА ЧАСТИНА

ІНТЕРАКТИВНА СИСТЕМА ПІДТРИМКИ ДІЯЛЬНОСТІ ВИКЛАДАЧА В ДИСТАНЦІЙНОМУ НАВЧАННІ

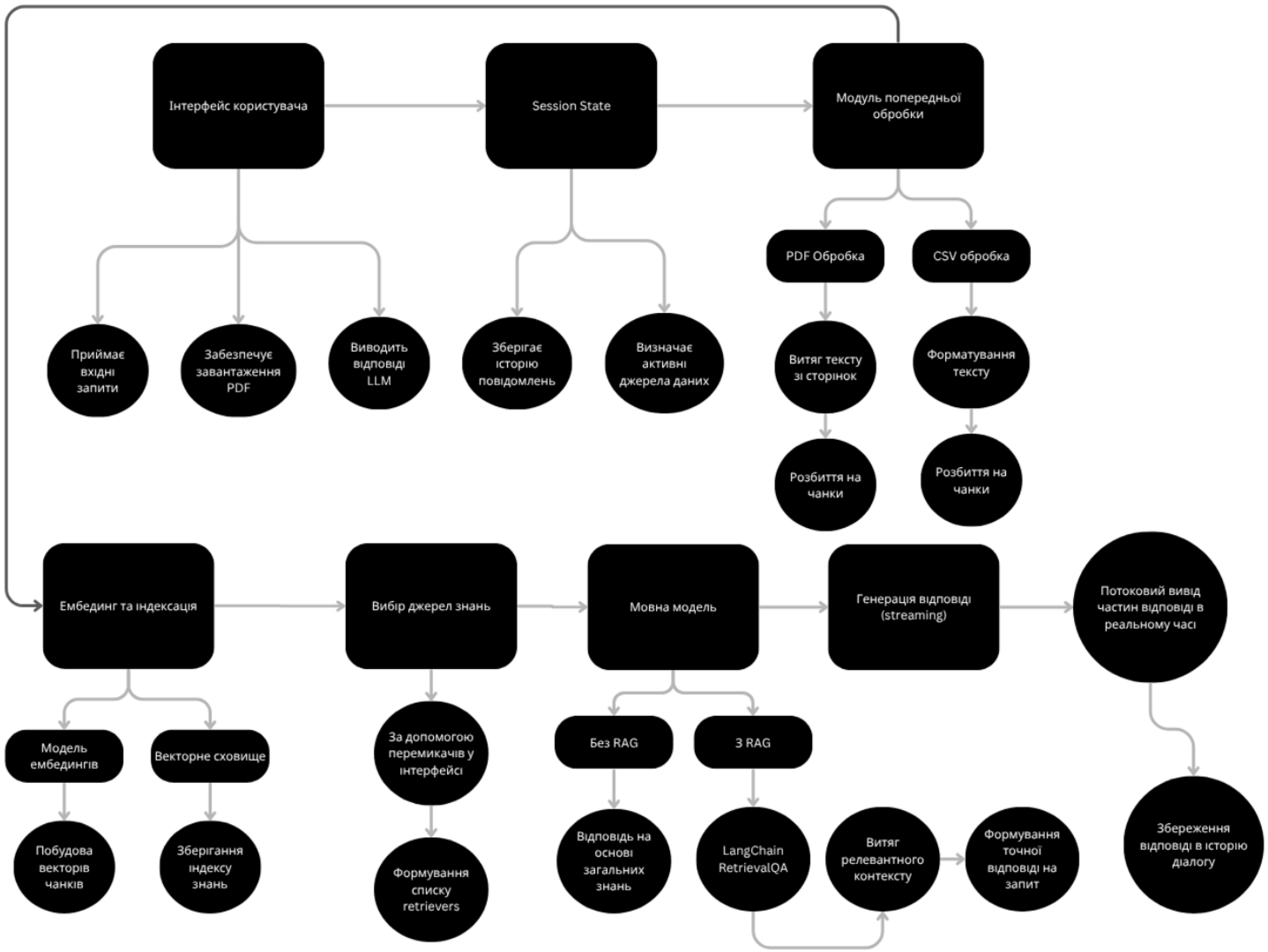

Рисунок В.1 — Блок-схема алгоритму роботи системи

ДОДАТОК Г

Лістинг програмного забезпечення

```
st.title("📚Асистент")
if "messages" not in st.session_state:
    st.session_state["messages"] = []
if not any(msg["role"] == "system" for msg in st.session_state["messages"]):
    st.session_state["messages"].insert(0, {
        "role": "system",
        "content": (
            """
                Ти – корисний асистент, який допомагає відповідати на запити, використовуючи інформацію з PDF та бази даних, яка вже надана.
                Твій користувач - викладач, якому потрібно або дізнатися інформацію про одного зі студентів (наприклад пошту),
                або створити конспект чи тести з pdf методички
                Будь ввічливим, професійним та лаконічним.
                Якщо не знаєш відповіді - чесно кажи, що не знаєш.
                Спілкуйся виключно українською мовою.
                """
        )
    })
if "vectorstore" not in st.session_state:
    st.session_state["vectorstore"] = None
if "csv_vectorstore" not in st.session_state:
    st.session_state["csv_vectorstore"] = None
with st.sidebar:
    uploaded_file = st.file_uploader("Завантажте ваш PDF", type=["pdf"])
    use_csv_only = st.checkbox("Використовувати ТІЛЬКИ CSV для відповідей")
    if st.session_state["vectorstore"]:
        use_pdf = st.checkbox("Використовувати PDF для відповідей", value=True)
@st.cache_resource
def create_csv_vectorstore():
    df = pd.read_csv("students.csv")
    text = ""
    for _, row in df.iterrows():
        text += f"{row['Ім\'я']} з групи {row['Група']} має електронну пошту {row['Email']}.\n"
```

splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=100)
.

```
            chunks = splitter.split_text(text)
            embeddings = OllamaEmbeddings(model="nomic-embed-text")
            return Chroma.from_texts(chunks, embedding=embeddings)
        def extract_text_from_pdf(uploaded_file):
            reader = PdfReader(uploaded_file)
            text = ""
            for page in reader.pages:
                content = page.extract_text()
                if content:
                    text += content
            return text
        def create_vectorstore(text):
            splitter = RecursiveCharacterTextSplitter(chunk_size=500,
chunk_overlap=100)
            chunks = splitter.split_text(text)
            embeddings = OllamaEmbeddings(model="nomic-embed-text")
            return Chroma.from_texts(chunks, embedding=embeddings)
        if uploaded_file and not st.session_state["vectorstore"]:
            with st.spinner("Обробка PDF..."):
                pdf_text = extract_text_from_pdf(uploaded_file)
                vectorstore = create_vectorstore(pdf_text)
                st.session_state["vectorstore"] = vectorstore
                st.success("PDF було успішно оброблено!")
        if st.session_state["vectorstore"]:
            if st.button("🗑 Очистити PDF"):
                st.session_state["vectorstore"] = None
                st.success("PDF очищено. Ви можете завантажити новий файл.")
        def model_res_generator():
            stream = ollama.chat(
                model="gemma3",
                messages=st.session_state["messages"],
                stream=True,
            )
            for chunk in stream:
                yield chunk["message"]["content"]
        if st.session_state["csv_vectorstore"] is None:
            st.session_state["csv_vectorstore"] = create_csv_vectorstore()
        for message in st.session_state["messages"]:
            if message["role"] != "system":
                with st.chat_message(message["role"]):
                    st.markdown(message["content"])
        if prompt := st.chat_input("Що хочеш спитати?"):
            st.session_state["messages"].append({"role": "user", "content": prompt})
```

```
        with st.chat_message("user"):
            st.markdown(prompt)
        with st.chat_message("assistant"):
            retrievers = []
            if st.session_state["csv_vectorstore"]:
                retrievers.append(st.session_state["csv_vectorstore"].as_retriever())

            if st.session_state["vectorstore"] and not use_csv_only and ('use_pdf' not in
locals() or use_pdf):
                retrievers.append(st.session_state["vectorstore"].as_retriever())

            if retrievers:
                retriever = EnsembleRetriever(retrievers=retrievers, weights=[1] *
len(retrievers))
                qa = RetrievalQA.from_chain_type(
                    llm=LangchainOllama(model="gemma3"),
                    retriever=retriever,
                    return_source_documents=False
                )
                answer = qa.run(prompt).strip()

                if not answer or "I don't know" in answer or "Sorry" in answer:
                    message = st.write_stream(model_res_generator())
                    st.session_state["messages"].append({"role": "assistant", "content":
message})
                else:
                    st.markdown(answer)
                    st.session_state["messages"].append({"role": "assistant", "content":
answer})
            else:
                message = st.write_stream(model_res_generator())
                st.session_state["messages"].append({"role": "assistant", "content":
message})
```