

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет комп'ютерних наук, фізики та математики
Кафедра комп'ютерних наук та програмної інженерії

**Розробка та застосування AI-ботів для автоматизації навчальних та
адміністративних процесів ЗВО**

Кваліфікаційна робота (проект)
на здобуття ступеня вищої освіти “бакалавр”

Виконав: здобувач IV курсу 12-441 групи
Спеціальності 121 Інженерія програмного
забезпечення
Освітньо-професійної програми «Інженерія
програмного забезпечення»
Бацалай Ілля Сергійович

Керівник: доктор філософії (PhD), доцент
кафедри комп'ютерних наук та програмної
інженерії, Полторацький М. Ю.

Рецензент: докторка філософії
(Інформаційні технології), наукова
співробітниця Інституту кібернетики ім.
В.М.Глушкова НАН України, Тарасіч Ю.Г.

ЗМІСТ

ВСТУП	3
РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ЗАСТОСУВАННЯ AI-ТЕХНОЛОГІЙ В УНІВЕРСИТЕТСЬКОМУ СЕРЕДОВИЩІ.....	8
1.1. Сучасний стан AI у вищій освіті: світові тенденції та практики	8
1.2. Класифікація та архітектура AI-чат-ботів: огляд підходів і технологій	11
1.3. Порівняльний аналіз варіантів розгортання AI-систем: хмарне vs локальне.....	14
1.4. Технологія Retrieval-Augmented Generation (RAG) як основа університетського AI-асистента.....	18
1.5. Обґрунтування вибору інструментальних засобів.....	21
1.6. Висновок до розділу	23
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ УНІВЕРСИТЕТСЬКОГО AI- АСИСТЕНТА.....	24
2.1. Аналіз вимог до системи: функціональні та нефункціональні вимоги	24
2.2. Загальна архітектура програмного комплексу	28
2.3. Проєктування конвертера університетських документів	32
2.4. Проєктування RAG-конвєсру та векторної бази знань	35
2.5. Проєктування користувацького інтерфейсу та сценарії взаємодії 38	
2.6. Висновок до розділу	41
РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ УНІВЕРСИТЕТСЬКОГО AI-АСИСТЕНТА.....	43
3.1. Розгортання середовища розробки та налаштування компонентів системи.....	43
3.2. Розробка конвертера університетських документів.....	48
3.3. Формування векторної бази знань та налаштування RAG- інтеграції.....	54
3.4. Тестування системи та оцінка якості відповідей	57
3.5. Висновок до розділу	62
ВИСНОВОК	64
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	67

ВСТУП

Стрімкий розвиток технологій штучного інтелекту в останнє десятиліття радикально змінив уявлення про можливості автоматизації інтелектуальної праці. Сфера вищої освіти, попри свою традиційно консервативну природу, також переживає суттєву трансформацію під впливом нових цифрових інструментів. Університети по всьому світу поступово усвідомлюють, що впровадження AI-систем здатне не лише оптимізувати внутрішні адміністративні процеси, а й принципово змінити характер взаємодії між студентами, викладачами та університетськими службами.

Автоматизовані інтелектуальні асистенти — чат-боти на основі великих мовних моделей — вже довели свою ефективність у таких галузях, як клієнтська підтримка, медицина, юриспруденція та фінанси. Університетське середовище, з його складною ієрархічною структурою, великим обсягом документообігу та постійним потоком типових запитів від студентів і персоналу, є надзвичайно перспективним полем для застосування подібних рішень. Щодня тисячі студентів звертаються до деканатів, кафедр і приймальних комісій із питаннями, відповіді на які містяться у стандартних документах — розкладах, навчальних планах, положеннях та регламентах. Ця рутинна робота поглинає значний ресурс адміністративного персоналу, який міг би бути спрямований на вирішення нестандартних і складніших завдань.

Актуальність теми даної кваліфікаційної роботи зумовлена також специфічними економічними реаліями вітчизняних університетів. Використання хмарних комерційних платформ, таких як Microsoft Azure Bot Service, передбачає оплату за кожен окремий запит до системи, що при інтенсивному навчальному навантаженні стає фінансово неприйнятним для більшості державних закладів освіти. Відтак, розробка та впровадження власного AI-асистента на базі локальної інфраструктури університету є не лише технічно обґрунтованим, а й економічно доцільним рішенням, яке забезпечує

незалежність від зовнішніх постачальників послуг і повний контроль над конфіденційними даними установи.

Окремого значення набуває питання роботи з університетськими даними. Сучасна освітня установа акумулює колосальні обсяги структурованої та неструктурованої інформації: навчальні програми, розклади занять, нормативні документи, інформація про персонал і студентів. Ефективна інтеграція цих даних до мовної моделі за допомогою технології Retrieval-Augmented Generation (RAG) дозволяє створити асистента, який не лише розуміє загальний контекст, але й оперує актуальною, специфічною для конкретного університету інформацією. Це принципово відрізняє спеціалізованого університетського AI-асистента від загальнодоступних чат-ботів.

Мета і завдання дослідження. Метою кваліфікаційної роботи є розробка та практична реалізація AI-асистента для автоматизації навчальних та адміністративних процесів університету на основі локально розгорнутої великої мовної моделі з інтеграцією університетської бази знань за технологією RAG.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- 1) провести аналіз сучасного стану застосування AI-технологій у сфері вищої освіти та огляд існуючих рішень у галузі університетських чат-ботів;
- 2) дослідити та порівняти варіанти розгортання мовних моделей — хмарне та локальне — з урахуванням специфічних вимог і обмежень університетського середовища;
- 3) розробити архітектуру системи, яка включає локальну мовну модель, векторну базу знань та програмний конвертер університетських документів;

- 4) розробити програмний застосунок для конвертації університетських документів різних форматів (DOCX, XLSX, PDF, RDF) у формати, придатні для інтеграції до векторної бази знань;
- 5) реалізувати прототип AI-асистента з інтегрованою університетською базою знань та провести його практичне тестування;
- 6) розробити інтерфейс взаємодії користувачів з системою та визначити перспективи подальшого масштабування і розгортання рішення на серверній інфраструктурі університету.

Об'єктом дослідження є процеси автоматизації навчальної та адміністративної діяльності університету засобами штучного інтелекту.

Предметом дослідження є методи та інструменти проєктування, розробки і розгортання AI-асистента на основі великих мовних моделей із застосуванням технології Retrieval-Augmented Generation для роботи з університетською базою знань.

Методи дослідження. У процесі виконання кваліфікаційної роботи застосовано комплекс загальнонаукових і спеціальних методів дослідження. Для аналізу предметної галузі використано методи системного аналізу та синтезу, що дозволило визначити ключові вимоги до проєктованої системи та структурувати підхід до її розробки. Порівняльний аналіз застосовано при дослідженні альтернативних варіантів розгортання (хмарне vs локальне) та при виборі базової мовної моделі з числа доступних відкритих рішень.

Для практичної реалізації застосовано методи об'єктно-орієнтованого програмування та компонентного проєктування програмного забезпечення. Розробка конвертера університетських документів здійснювалась із використанням методів парсингу структурованих і напівструктурованих даних. При інтеграції бази знань до мовної моделі застосовано принципи RAG-архітектури, що поєднує семантичний пошук у векторному просторі з

генерацією відповідей мовною моделлю. Тестування системи проводилось методом сценарного тестування на основі реальних запитів.

Практичне значення результатів. Практичне значення отриманих результатів визначається тим, що розроблений у рамках даної роботи програмний комплекс є готовим до впровадження прототипом AI-асистента, адаптованим до реальних умов університетського середовища. Розроблена система вирішує конкретну прикладну задачу — автоматизацію обробки типових запитів від студентів і викладачів — і може бути розгорнута на власній серверній інфраструктурі університету без необхідності використання платних хмарних сервісів.

Розроблений програмний конвертер документів є самостійним програмним засобом, який може застосовуватись для підготовки університетських даних до інтеграції в будь-які системи на основі мовних моделей. Підтримка форматів DOCX, XLSX, PDF та RDF робить конвертер універсальним інструментом для роботи з гетерогенними джерелами університетської інформації. Методологічні підходи, застосовані при розробці RAG-конвеєра, можуть бути відтворені для впровадження аналогічних систем в інших навчальних закладах або організаціях зі схожою структурою документообігу.

Економічний ефект від впровадження запропонованого рішення полягає у повній відмові від щомісячних або посерверних платежів за використання комерційних хмарних платформ, що при прогнозованому навантаженні в сотні тисяч запитів на рік дозволить суттєво скоротити операційні витрати університету на цифрову інфраструктуру.

Структура роботи

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків до кожного розділу, загального висновку та списку використаних джерел.

У першому розділі розглянуто теоретичні засади застосування AI-технологій у вищій освіті: проаналізовано сучасний стан розвитку великих мовних моделей, досліджено принципи роботи RAG-архітектури, здійснено порівняльний аналіз підходів до розгортання AI-систем — хмарного та локального — та обґрунтовано вибір інструментальних засобів для реалізації проєкту.

Другий розділ присвячено проєктуванню системи: визначено функціональні та нефункціональні вимоги до AI-асистента, розроблено архітектуру програмного комплексу, описано структуру компонентів системи та механізми їхньої взаємодії, а також представлено детальний опис архітектурних рішень щодо конвертера документів і RAG-конвеєру.

Третій розділ містить опис практичної реалізації системи: розгортання середовища, розробку програмного конвертера з підтримкою форматів DOCX, XLSX, PDF та RDF, процес формування векторної бази знань, налаштування RAG-інтеграції, а також результати тестування та оцінку якості відповідей системи.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ЗАСТОСУВАННЯ АІ-ТЕХНОЛОГІЙ В УНІВЕРСИТЕТСЬКОМУ СЕРЕДОВИЩІ

1.1. Сучасний стан АІ у вищій освіті: світові тенденції та практики

Ще кілька років тому штучний інтелект у контексті освіти сприймався переважно як щось футуристичне — цікава тема для наукових конференцій, але аж ніяк не повсякденний інструмент. Сьогодні ситуація кардинально змінилась. Університети по всьому світу активно впроваджують АІ-рішення у свої внутрішні процеси, і ця тенденція лише набирає обертів. За даними звіту EDUCAUSE “The Impact of AI on Work in Higher Education” (2026) зазначається, що понад 90% провідних університетів США та Європи вже використовують або перебувають у процесі пілотного тестування АІ-інструментів у навчальному або адміністративному контексті [1].

Але що саме розуміють під "АІ у вищій освіті"? Насправді це дуже широке поняття, яке охоплює цілий спектр різноманітних технологій і застосувань. Тут і системи персоналізованого навчання, які адаптують навчальний матеріал під конкретного студента, і інструменти автоматичної перевірки письмових робіт, і платформи предиктивної аналітики, що дозволяють завчасно виявляти студентів у зоні ризику академічної неуспішності. Окремо варто виділити інтелектуальних асистентів і чат-ботів — саме цей напрям є найбільш динамічно зростаючим і безпосередньо пов'язаним з темою даної роботи.

Якщо говорити про реальний досвід конкретних університетів, то прикладів накопичилося чимало. Технологічний інститут Джорджії (Georgia Institute of Technology) ще у 2016 році запустив чат-бота "Jill Watson", розробленого на основі платформи IBM Watson, для відповідей на запитання студентів у онлайн-курсу з Knowledge-Based Artificial Intelligence (KBAI) в

рамках програми Online Master of Science in Computer Science. На курс щосеместру записувалось близько 300 студентів, які генерували до 10 000 повідомлень у форумах — обсяг, непосильний для команди з восьми асистентів-людей. Примітно, що спочатку студенти навіть не здогадувались, що спілкуються з програмою, а не з асистентом-людиною [2]. Цей проєкт продемонстрував принципову можливість використання AI для автоматизації навчальної підтримки і викликав широкий резонанс у академічній спільноті.

Georgia State University (США) у 2016 році запустила чат-бота "Pounce" для підтримки студентів у процесі вступу і зарахування. За підсумками першого сезону, Pounce обмінявся з вступниками майже 200 000 повідомленнями. При цьому менш ніж 1% отриманих від студентів запитань потребував безпосереднього втручання персоналу університету — решта оброблялась автоматично. За оцінками адміністрації, без чат-бота для обробки такого обсягу звернень довелося б найняти 10 штатних співробітників на повну зайнятість [3].

Що стосується вітчизняного досвіду, тут ситуація більш стримана, але теж показова. Ряд провідних українських університетів — Київський політехнічний інститут, Харківський національний університет радіоелектроніки, Національний університет "Львівська політехніка" — починають експериментувати з AI-інструментами, хоча здебільшого мова поки що йде про пілотні проєкти або використання готових комерційних рішень. Фінансові обмеження, характерні для більшості державних закладів освіти в Україні, стають суттєвим бар'єром для масштабного впровадження хмарних AI-сервісів, вартість яких прив'язана до кількості запитів.

Попри всі відмінності між конкретними університетами і проєктами, можна виділити кілька спільних закономірностей, які характеризують поточний стан справ у галузі. По-перше, найбільший попит на AI-автоматизацію спостерігається саме у сфері студентської підтримки — відповіді на типові

запитання, навігація по університетських ресурсах, інформування про дедлайни та правила. Це пояснюється тим, що саме тут обсяг однотипних запитів є найбільшим, а відповіді найлегше формалізувати.

По-друге, університети все більше усвідомлюють важливість конфіденційності даних. Передача інформації про студентів і персонал на сервери третіх сторін викликає законні побоювання з точки зору законодавства про захист персональних даних, зокрема GDPR у Європі. Саме тому локальне розгортання AI-систем, при якому всі дані залишаються всередині університетської мережі, стає все більш привабливою альтернативою хмарним рішенням.

По-третє, і це, мабуть, найцікавіше з технічної точки зору, якість відкритих мовних моделей сьогодні вже наближається до якості комерційних систем для більшості практичних завдань. Якщо ще три-чотири роки тому між GPT-4 від OpenAI та будь-якою відкритою альтернативою лежала прірва, то сьогодні моделі на кшталт Gemma, Llama або Mistral здатні впоратись із завданнями відповіді на запитання у спеціалізованому домені цілком на конкурентному рівні. Це відкриває реальну можливість для розгортання ефективних AI-асистентів без використання платних API.

Таким чином, поточний момент є особливо сприятливим для впровадження власних AI-рішень у університетах: технологічний рівень відкритих моделей досить високий, досвід піонерських проєктів накопичено і задокументовано, а потреба у зниженні операційних витрат робить локальне розгортання економічно обґрунтованим вибором.

1.2. Класифікація та архітектура AI-чат-ботів: огляд підходів і технологій

Щоб розібратись у тому, що саме ми розробляємо, варто спочатку систематизувати саме поняття "чат-бот" і розглянути, як ця технологія еволюціонувала до свого нинішнього стану. Це не просто академічна вправа — розуміння різних класів систем дозволяє усвідомлено обирати архітектурний підхід і правильно визначати можливості та обмеження конкретного рішення.

Якщо дивитись на еволюцію чат-ботів у часі, то виділяють три покоління систем. Перше покоління — так звані rule-based боти, або боти на основі правил. Принцип їхньої роботи гранично простий: розробник заздалегідь прописує набір ключових слів або фраз-тригерів, і коли користувач надсилає повідомлення, система шукає відповідність у своєму словнику і повертає заздалегідь підготовлену відповідь. Такі боти були популярні у 2010-х роках і досі використовуються для дуже простих сценаріїв. Їхня головна перевага — передбачуваність і простота розробки. Недоліки очевидні: будь-яке відхилення від очікуваного формулювання запиту призводить до того, що бот або не розуміє питання, або дає некоректну відповідь.

Друге покоління — боти на основі машинного навчання та NLP (Natural Language Processing). Замість жорстко прописаних правил такі системи навчаються на прикладах реальних діалогів і здатні розпізнавати наміри (intents) користувача навіть при варіативному формулюванні. Саме на цій парадигмі побудовані такі платформи, як Dialogflow від Google, Microsoft Bot Framework (та заснована на ньому Azure Bot Service), IBM Watson Assistant.

Третє покоління — боти на основі великих мовних моделей (Large Language Models, LLM). Це принципово інший підхід: замість того, щоб заздалегідь розмічати можливі наміри і прописувати відповідні відповіді, система спирається на модель, яка навчилася на колосальних обсягах текстових

даних і здатна розуміти контекст, підтримувати зв'язний діалог і генерувати відповіді на запитання, яких ніхто не передбачав заздалегідь. Саме до цього класу належить розроблюваний у даній роботі AI-асистент.

Таблиця 1.1 — Порівняльна характеристика поколінь чат-ботів

Характеристика	Rule-based боти	ML/NLP боти	LLM-боти
Принцип роботи	Шаблони та ключові слова	Класифікація намірів	Генерація на основі контексту
Гнучкість відповідей	Дуже низька	Середня	Висока
Якість розуміння мови	Низька	Середня	Висока
Робота з новими запитами	Не підтримується	Обмежена	Підтримується
Складність налаштування	Низька	Середня	Середня–Висока
Потреба у навчальних даних	Не потрібна	Потрібна розмітка	Залежить від підходу
Вартість інфраструктури	Низька	Середня	Висока (але знижується)

Якщо заглибитись у архітектуру LLM-бота, то ключовим компонентом є, звісно, сама мовна модель — нейронна мережа архітектури Transformer, навчена на мільярдах токенів тексту. Однак для практичного застосування у конкретній предметній області — наприклад, для відповідей на запитання про конкретний університет — однієї лише базової мовної моделі недостатньо. Модель не знає нічого про розклад занять, специфічні правила вступу чи контакти конкретного деканату. Саме тут на сцену виходять додаткові архітектурні компоненти.

Існує кілька підходів до того, як "навчити" мовну модель специфічної для домену інформації. Перший — fine-tuning, або дообучення: модель перенавчається на спеціальному датасеті, після чого ця інформація стає частиною її ваг. Такий підхід дає добрі результати, але вимагає значних обчислювальних ресурсів і, що важливіше, потребує повторного перенавчання при кожному оновленні даних. Для університетського середовища, де інформація (розклади, списки викладачів, навчальні плани) регулярно змінюється, це суттєве обмеження.

Другий підхід — Retrieval-Augmented Generation (RAG). Суть його полягає у тому, що актуальна інформація зберігається окремо від моделі у векторній базі знань, і при кожному запиті система спочатку знаходить релевантні фрагменти з цієї бази, а потім передає їх разом із запитом до мовної моделі для генерації відповіді. Оновлення бази знань не потребує перенавчання моделі — достатньо просто оновити документи у базі. Саме цей підхід обрано для реалізації університетського AI-асистента, і про нього детально йтиметься у підрозділі 1.4.

З точки зору архітектури повноцінного AI-асистента, крім мовної моделі і бази знань, система включає ще кілька важливих компонентів. Це інтерфейс взаємодії з користувачем (веб-чат, Telegram-бот, голосовий інтерфейс), оркестратор запитів, який координує роботу всіх підсистем, та компоненти препроцесингу і постпроцесингу — відповідно для підготовки вхідних даних і форматування вихідних відповідей. Детальна архітектурна схема системи розглядається у розділі 2.

Окремо варто зупинитись на понятті контекстного вікна (context window) — одному з ключових параметрів будь-якої мовної моделі. Контекстне вікно визначає максимальний обсяг тексту (у токенах), який модель може врахувати при генерації відповіді. Це важливо при RAG-підході, адже знайдені релевантні фрагменти з бази знань передаються до моделі у вигляді тексту і займають

частину контекстного вікна. Модель Gemma-3 12B, обрана для реалізації проєкту, підтримує контекстне вікно у 128 тисяч токенів, що є більш ніж достатнім для більшості практичних сценаріїв [4].

1.3. Порівняльний аналіз варіантів розгортання AI-систем: хмарне vs локальне

Одним із перших і найважливіших рішень при розробці будь-якої AI-системи є вибір інфраструктури розгортання. Це рішення має далекосяжні наслідки для всього проєкту — від вартості експлуатації до рівня безпеки даних і можливостей масштабування. У контексті університетського AI-асистента це питання стоїть особливо гостро, оскільки від нього напряду залежить фінансова стійкість проєкту в довгостроковій перспективі.

Хмарне розгортання передбачає використання AI-інфраструктури, яку надають великі технологічні компанії. Найпоширеніші платформи — Microsoft Azure (Azure OpenAI Service, Azure Bot Service), Amazon Web Services (Amazon Bedrock, Amazon Lex), Google Cloud (Vertex AI, Dialogflow CX). Усі ці платформи пропонують потужні, готові до використання рішення з добре документованими API, засобами моніторингу та підтримкою. Не дивно, що саме хмарний підхід є домінуючим у корпоративному секторі.

Однак хмарні рішення мають принципову особливість з точки зору моделі ціноутворення: оплата, як правило, стягується за кожен запит або за кількість оброблених токенів. Наприклад, Azure OpenAI Service для моделі GPT-4o стягує плату окремо за вхідні та вихідні токени. При помірному навантаженні (скажімо, кілька сотень запитів на день від студентів) ця сума може бути цілком прийнятною. Але університет — це установа з тисячами студентів і численним персоналом. При інтенсивному використанні (наприклад,

у сесійний період) кількість щоденних запитів може сягати тисяч, а це вже зовсім інший порядок витрат [5].

Саме така ситуація виникла з попереднім рішенням університету на базі Azure Bot Service. Попри зручність і якість платформи, фінансове навантаження виявилось неприйнятним для бюджету навчального закладу, що і стало причиною пошуку альтернативи. Ця ситуація є типовою, і вітчизняні університети, на відміну від добре фінансованих зарубіжних закладів, просто не мають можливості довгостроково підтримувати значні витрати на хмарні AI-сервіси.

Локальне розгортання (on-premises deployment) є принципово іншим підходом. У цьому випадку вся інфраструктура — сервери, GPU для обчислень, системне програмне забезпечення — знаходиться безпосередньо у власності університету і розміщена у його приміщеннях або дата-центрі. Мовна модель запускається на власному обладнанні, і жодні дані не передаються зовнішнім провайдерам. Основна перевага очевидна: відсутність операційних витрат на кожен запит. Після первинного налаштування система може обробляти скільки завгодно запитів без додаткових витрат.

Таблиця 1.2 — Порівняльний аналіз хмарного та локального розгортання AI-системи

Критерій	Хмарне розгортання	Локальне розгортання
Початкові витрати	Низькі (підписка/pay-as-you-go)	Середні–Високі (обладнання)
Операційні витрати	Постійні, залежать від навантаження	Мінімальні (електроенергія, обслуговування)
Масштабованість	Практично необмежена	Обмежена наявним обладнанням
Конфіденційність даних	Дані передаються провайдеру	Дані не покидають університет
Відповідність GDPR	Потребує перевірки договору	Повний контроль
Швидкість впровадження	Висока	Середня
Залежність від інтернету	Повна	Відсутня
Технічна підтримка	Від провайдера	Власна або підрядна
Актуальність моделі	Провайдер оновлює автоматично	Оновлення вручну
Кастомізація	Обмежена API провайдера	Повна гнучкість

Локальне розгортання має і свої очевидні складнощі. Головна з них — необхідність наявності відповідного апаратного забезпечення. Сучасні великі мовні моделі є обчислювально дуже вимогливими: для комфортної роботи моделі масштабу 12 мільярдів параметрів у режимі інференсу необхідна GPU з достатнім обсягом відеопам'яті — як правило, від 16 ГБ і більше. Це або NVIDIA RTX серії 3000/4000, або серверні GPU класу A100/H100. Більшість

сучасних університетів, однак, вже має певну GPU-інфраструктуру для навчальних і наукових потреб, тому це питання може вирішуватись шляхом розподілу наявних ресурсів.

Ще одним аргументом на користь локального розгортання є питання безперебійності роботи. Хмарні сервіси, попри їхню надійність, іноді переживають технічні збої або планові роботи з обслуговування. При цьому університет повністю залежить від доступності зовнішнього сервісу. Локальна система є більш незалежною в цьому відношенні і може функціонувати навіть при відсутності доступу до інтернету — що може бути важливим, наприклад, в умовах нестабільного підключення.

Варто також розглянути так звані гібридні підходи, при яких деякі компоненти системи розміщуються локально, а інші — у хмарі. Наприклад, основна обробка запитів може відбуватись локально, тоді як функції резервування або масштабування при пікових навантаженнях — у хмарі. Такий підхід дозволяє збалансувати вартість і надійність, однак суттєво ускладнює архітектуру і операційне управління. Для університету, що прагне простоти і передбачуваності витрат, повністю локальне розгортання залишається оптимальним вибором.

Рішення про перехід на локальне розгортання, прийняте університетом у рамках даного проєкту, є стратегічно виваженим і відповідає загальній тенденції до суверенітету даних і контролю над AI-інфраструктурою, яка спостерігається у державному секторі та освіті по всьому світу. Цей вибір дозволяє не лише вирішити поточну фінансову проблему, але й закласти фундамент для довгострокового розвитку університетської AI-інфраструктури.

1.4. Технологія Retrieval-Augmented Generation (RAG) як основа університетського AI-асистента

Retrieval-Augmented Generation — це підхід, який вирішує одну з фундаментальних проблем великих мовних моделей: їхню "нерухомість" у часі. Будь-яка мовна модель навчається на даних, зібраних до певного моменту, і не має уявлення про те, що відбулось після. Крім того, модель нічого не знає про специфічну внутрішню інформацію конкретної організації — будь то університет, підприємство або державна установа. RAG вирішує цю проблему елегантним і практичним способом, не вимагаючи дообучення моделі.

Концепцію RAG було запропоновано дослідниками Facebook AI Research (FAIR) у 2020 році у статті "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" [6]. Ідея полягає у поєднанні двох механізмів: пошуку релевантної інформації з зовнішньої бази знань і генерації відповіді мовною моделлю на основі знайденої інформації. Це дозволяє моделі "знати" те, чому її ніхто не навчав — достатньо просто додати відповідні документи до бази знань.

Розглянемо, як RAG-система працює крок за кроком. Коли користувач надсилає запит — наприклад, "Коли починається зимова сесія?" — система не передає це питання безпосередньо до мовної моделі. Спочатку запит проходить через компонент пошуку. Текст запиту перетворюється на числовий вектор за допомогою моделі ембедингів (embedding model) — спеціальної нейронної мережі, яка навчилася кодувати семантичний зміст тексту у вигляді вектора у багатовимірному просторі. Схожі за змістом тексти матимуть схожі (близькі) вектори.

Цей вектор порівнюється з векторами всіх документів у базі знань, і система знаходить ті фрагменти, вектори яких найближчі до вектора запиту — тобто ті, що найбільш семантично схожі на запит. Важливо розуміти, що це не пошук за ключовими словами: система здатна знайти релевантний фрагмент

навіть якщо в ньому немає жодного слова з оригінального запиту, але він відповідає на поставлене питання по суті.

Знайдені фрагменти (зазвичай кілька найбільш релевантних) додаються до запиту і передаються до мовної моделі у вигляді так званого промпту: "Відповідаючи на питання користувача, використовуй наступну інформацію: [знайдені фрагменти]. Питання користувача: [оригінальний запит]". Мовна модель генерує відповідь, спираючись на надану їй інформацію. Якщо база знань містить актуальні дані про зимову сесію — модель дасть точну відповідь.

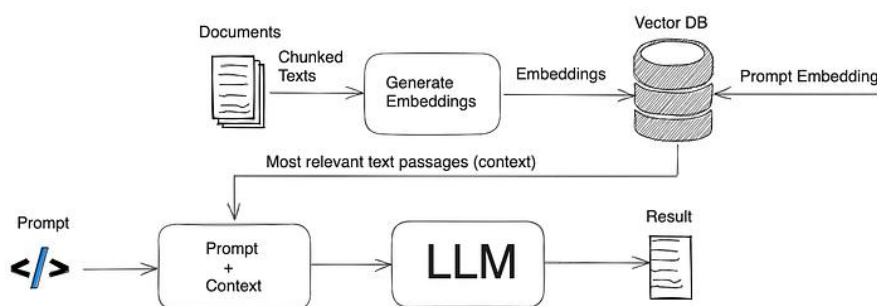


Рисунок. 1.1 – Схема роботи RAG-системи [7]

Для зберігання векторних представлень документів використовуються спеціалізовані бази даних — векторні сховища (vector stores). Серед найпопулярніших: Chroma, Pinecone, Weaviate, FAISS від Meta. Кожна з них реалізує ефективні алгоритми пошуку найближчих сусідів у багатовимірному просторі (Approximate Nearest Neighbor, ANN), що дозволяє знаходити релевантні фрагменти серед мільйонів документів за лічені мілісекунди.

Критично важливим етапом при побудові RAG-системи є підготовка документів — їхнє завантаження, парсинг і розбиття на фрагменти (chunking). Цей крок часто недооцінюється, хоча саме він значною мірою визначає якість кінцевої системи. Якщо документи погано розпарсені або фрагменти підібрані невдало (занадто великі або занадто дрібні), система буде знаходити нерелевантні фрагменти або не матиме достатньо контексту для формування якісної відповіді.

Саме ця проблема постала у повний зріст при роботі з університетськими документами. Файли у форматах DOCX, XLSX і PDF мають складну внутрішню структуру, і простий витяг тексту з них часто дає результат, непридатний для якісної індексації. Таблиці перетворюються на безструктурний рядок цифр, заголовки втрачають свій ієрархічний зв'язок з текстом, формули і спеціальні символи відображаються некоректно. Власне, необхідність вирішення цієї проблеми і стала поштовхом до розробки спеціалізованого конвертера документів, описаного у практичній частині роботи.

Якість RAG-системи оцінюється за кількома ключовими метриками. Точність пошуку (retrieval precision) показує, наскільки релевантні фрагменти знаходить система. Повнота пошуку (recall) — чи не пропускає система важливої інформації. Відповідність відповіді знайденим фрагментам (faithfulness) — наскільки модель спирається на надану інформацію, а не "вигадує" від себе. Нарешті, загальна якість відповіді оцінюється з точки зору корисності для кінцевого користувача. Практичне тестування розробленої системи за цими метриками описано у розділі 3.

Порівняно з альтернативними підходами — зокрема, з fine-tuning — RAG має низку суттєвих переваг саме для університетського контексту. Оновлення бази знань відбувається практично миттєво: досить завантажити новий документ, і система одразу починає використовувати його при відповідях. Не потрібно жодного перенавчання моделі. Крім того, RAG природним чином підтримує посилання на джерела — система може вказати, з якого саме документа взята та чи інша інформація, що підвищує довіру до відповідей і спрощує перевірку.

1.5. Обґрунтування вибору інструментальних засобів

Перед тим як переходити до проєктування і реалізації системи, необхідно обґрунтувати вибір конкретних інструментів і технологій. Цей вибір не є довільним — він продиктований як технічними вимогами проєкту, так і специфічними обмеженнями університетського середовища: відсутністю значних фінансових ресурсів, необхідністю локального розгортання і бажанням максимально спростити процес налаштування і підтримки системи.

LM Studio. LM Studio — це графічний застосунок для локального запуску великих мовних моделей на персональному комп'ютері або сервері. Програма надає зручний інтерфейс для завантаження моделей з Hugging Face, їх налаштування і запуску у форматі GGUF (GPT-Generated Unified Format) — оптимізованому форматі, який дозволяє запускати квантизовані версії моделей навіть на відносно скромному обладнанні з частковим або повним використанням CPU замість GPU. Ключова перевага LM Studio для даного проєкту полягає у тому, що програма автоматично піднімає локальний API-сервер, сумісний зі стандартним OpenAI API. Це означає, що будь-який застосунок, розрахований на роботу з OpenAI GPT, може без жодних змін коду переключитись на локальну модель через LM Studio [8].

AnythingLLM. AnythingLLM — це відкрита платформа для побудови RAG-систем, яка дозволяє завантажувати документи, автоматично їх індексувати і організовувати взаємодію між базою знань і мовною моделлю. Програма підтримує широкий спектр форматів документів і векторних сховищ та надає веб-інтерфейс для управління робочими просторами (workspaces) — логічно розділеними базами знань. AnythingLLM може підключатись до будь-якого OpenAI-сумісного API, що робить його ідеальним партнером для LM Studio: документи завантажуються в AnythingLLM, індексуються, а генерація відповідей відбувається через LM Studio. Важливою перевагою є те, що

AnythingLLM є повністю безкоштовним програмним забезпеченням з відкритим кодом, без жодних прихованих платежів або обмежень на кількість запитів [9].

Модель Gemma-3 12B. Gemma — це сімейство відкритих мовних моделей, розроблених компанією Google DeepMind. Модель Gemma-3 розміром 12 мільярдів параметрів є оптимальним вибором для даного проєкту з кількох причин. По-перше, вона демонструє якість відповідей, порівнянну з набагато більшими моделями, завдяки сучасній архітектурі і якісним навчальним даним. По-друге, у квантизованому вигляді (4-bit або 8-bit квантизація) модель може працювати на GPU з відеопам'яттю від 12–16 ГБ, що відповідає можливостям університетського обладнання. По-третє, Gemma-3 добре підтримує мультилінгвальні сценарії, зокрема роботу з українською мовою, хоча основна мова навчання — англійська. Модель розповсюджується за ліцензією, що дозволяє некомерційне і комерційне використання [10].

Python як мова розробки конвертера. Для розробки програмного конвертера університетських документів обрано мову програмування Python. Цей вибір зумовлений кількома факторами. Python має найбільш розвинену екосистему бібліотек для роботи з документами різних форматів: `python-docx` для DOCX, `openpyxl` для XLSX, `pdfplumber` для PDF, `rdflib` для RDF. Усі ці бібліотеки активно підтримуються і мають детальну документацію. Крім того, Python є стандартною мовою у галузі data science і AI, що полегшує інтеграцію конвертера з іншими компонентами системи і подальшу підтримку коду.

Сукупність обраних інструментів утворює цілісний і добре інтегрований стек технологій, здатний забезпечити повний цикл роботи AI-асистента — від підготовки даних до надання відповідей користувачам. Всі компоненти є безкоштовними або відкритими, що повністю відповідає вимозі університету щодо відсутності операційних витрат на ліцензування програмного забезпечення.

1.6. Висновок до розділу

У першому розділі проведено комплексний аналіз теоретичних основ і технологічного контексту, в якому реалізується проєкт університетського AI-асистента. Визначено, що сучасний стан розвитку AI-технологій і відкритих мовних моделей робить локальне розгортання інтелектуальних асистентів не просто можливим, але й економічно доцільним для університетів з обмеженим бюджетом.

Аналіз еволюції чат-ботів показав, що системи на основі великих мовних моделей (третє покоління) принципово перевершують попередні підходи за гнучкістю і якістю розуміння природної мови, що робить їх оптимальним вибором для університетського середовища з широким спектром можливих запитів. Порівняльний аналіз хмарного та локального розгортання однозначно підтвердив доцільність локального підходу для даного проєкту: він усуває постійні операційні витрати, забезпечує конфіденційність університетських даних і повну незалежність від зовнішніх сервісів.

Детально розглянута технологія RAG визначена як ключова для реалізації мети проєкту: вона дозволяє інтегрувати специфічну університетську інформацію до мовної моделі без дороговартісного дообучення і забезпечує можливість оперативного оновлення бази знань. Обраний інструментальний стек — LM Studio, AnythingLLM, модель Gemma-3 12B і мова Python — утворює збалансоване рішення, що задовольняє всі технічні і організаційні вимоги проєкту.

Теоретичні засади, закладені у цьому розділі, є основою для проєктування і практичної реалізації системи, які розглядаються у наступних розділах роботи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ УНІВЕРСИТЕТСЬКОГО AI-АСИСТЕНТА

2.1. Аналіз вимог до системи: функціональні та нефункціональні вимоги

Проектування будь-якої програмної системи починається з ретельного аналізу вимог. Цей етап є фундаментальним: саме тут визначається, що система має робити і як вона має це робити. Пропустити або поверхово пройти цей крок — означає ризикувати тим, що в кінці розробки отримаєш технічно бездоганний продукт, який не відповідає реальним потребам користувачів. У даному проєкті аналіз вимог проводився шляхом вивчення реального досвіду роботи адміністративного персоналу університету, аналізу типових звернень студентів і викладачів, а також з урахуванням технічних обмежень наявної інфраструктури.

Усі вимоги до системи прийнято поділяти на дві великі категорії: функціональні та нефункціональні. Функціональні вимоги описують конкретні можливості системи — що вона повинна вміти робити. Нефункціональні вимоги стосуються характеристик якості — як саме система має це робити: наскільки швидко, надійно, безпечно, зручно. Обидві категорії однаково важливі, і нехтування нефункціональними вимогами на етапі проектування — поширена помилка, яка потім обертається серйозними проблемами при впровадженні.

Функціональні вимоги. Аналіз потреб університету дозволив визначити наступний перелік функціональних вимог до системи AI-асистента.

Система має забезпечувати обробку запитів природною мовою від студентів і викладачів, тобто розуміти питання, сформульовані довільним чином, а не лише у суворо визначеному форматі. Це принципово відрізняє AI-

асистента від попередніх рішень на основі меню чи ключових слів. Користувач повинен мати змогу запитати "Де знайти розклад на другий семестр?" і "Коли у нас пари в середу?" — і отримати однаково корисну відповідь на обидва питання, попри різне формулювання.

Система має надавати точні та актуальні відповіді на запитання, що стосуються університетської інформації: розкладу занять, навчальних планів, переліку спеціальностей та освітніх програм, контактів кафедр і деканатів, вимог до вступу, академічних правил і регламентів, інформації про персонал. Вся ця інформація має надходити з офіційних університетських документів, інтегрованих до бази знань системи.

Важливою функціональною вимогою є підтримка контексту діалогу в межах однієї сесії спілкування. Це означає, що якщо користувач запитав про розклад групи, а потім написав "а коли у них залік з математики?", система має розуміти, що йдеться про ту саму групу, не змушуючи користувача повторювати контекст у кожному повідомленні. Реалізація цієї вимоги суттєво підвищує природність і зручність взаємодії.

Окремою функціональною вимогою є можливість адміністративного управління базою знань: завантаження нових документів, оновлення застарілих, видалення непотрібних — і все це без необхідності вносити зміни до коду системи або перезапустити сервіси. Адміністратор має мати зручний інтерфейс для управління контентом, доступним асистентові.

Система також має коректно поводитись у ситуаціях, коли відповідь на питання відсутня у базі знань. У такому разі асистент повинен чесно повідомити користувача про відсутність інформації і, якщо можливо, запропонувати альтернативний спосіб її отримання — наприклад, контакти відповідного підрозділу. Це краще, ніж генерувати правдоподібно звучачу, але

неточну або вигадану відповідь, що є однією з найбільш відомих проблем мовних моделей (так звані "галюцинації").

Таблиця 2.1 — Функціональні вимоги до системи AI-асистента

ID	Вимога	Пріоритет
FR-01	Обробка запитів природною мовою (українська/англійська)	Високий
FR-02	Пошук інформації у базі знань університету	Високий
FR-03	Генерація зв'язних текстових відповідей на основі знайденої інформації	Високий
FR-04	Підтримка контексту діалогу в межах сесії	Середній
FR-05	Повідомлення про відсутність інформації у базі знань	Високий
FR-06	Завантаження та індексація документів форматів DOCX, XLSX, PDF, RDF	Високий
FR-07	Адміністративний веб-інтерфейс управління базою знань	Середній
FR-08	API-інтерфейс для інтеграції з зовнішніми системами (Telegram тощо)	Середній
FR-09	Ведення журналу запитів для аналізу і покращення системи	Низький
FR-10	Конвертація університетських документів у формат JSON/JSONL	Високий

Нефункціональні вимоги. Поряд з функціональними, не менш важливими є вимоги до якісних характеристик системи.

Продуктивність є однією з ключових нефункціональних вимог. Час відповіді системи на запит користувача не має перевищувати розумних меж: надто довге очікування знижує зручність використання і створює враження

"підвисшої" програми. З урахуванням обчислювальних можливостей локального обладнання, прийнятним вважається час генерації відповіді до 30 секунд для складних запитів, і до 10 секунд для простих. При цьому система має відображати ознаку обробки запиту (індикатор завантаження або потоковий вивід тексту), щоб користувач розумів, що процес іде.

Надійність системи визначається її здатністю стабільно працювати протягом тривалого часу без збоїв. Університетський AI-асистент має бути доступним протягом навчального дня і вечора (орієнтовно з 7:00 до 22:00), оскільки студенти можуть звертатись до нього у будь-який зручний час. Система має коректно відновлюватись після збоїв і мати механізми автоматичного перезапуску компонентів у разі виникнення помилок.

Безпека і конфіденційність даних є особливо важливою вимогою в університетському контексті. Доступ до адміністративного інтерфейсу управління базою знань має бути захищений автентифікацією. Дані, які обробляє система — як питання користувачів, так і документи університету — не повинні передаватись за межі університетської мережі. Журнал запитів, якщо він ведеться, має зберігатись у деперсоналізованому вигляді або з дотриманням вимог щодо захисту персональних даних.

Масштабованість на рівні бази знань означає, що система має без суттєвої деградації якості справлятися із зростанням обсягу індексованих документів. При збільшенні кількості документів у базі час пошуку не повинен зростати лінійно — цього досягається використанням ефективних алгоритмів індексації векторного сховища. Масштабованість на рівні паралельних запитів визначається можливостями наявного обладнання і на початковому етапі обмежена кількома одночасними сесіями.

Таблиця 2.2 — Нефункціональні вимоги до системи AI-асистента

Категорія	Вимога	Метрика
Продуктивність	Час відповіді на простий запит	≤ 10 секунд
Продуктивність	Час відповіді на складний запит	≤ 30 секунд
Надійність	Час доступності системи	з 7:00 до 22:00
Надійність	Автоматичне відновлення після збою	Обов'язково
Безпека	Автентифікація адмін-інтерфейсу	Обов'язково
Безпека	Локальна обробка всіх даних	Дані не покидають мережу
Масштабованість	Максимальний обсяг бази знань	До 10 000 документів
Зручність	Інтуїтивність інтерфейсу	Без навчання для студентів
Супровід	Оновлення бази знань без зупинки	Обов'язково

Сукупність функціональних і нефункціональних вимог, визначених у цьому підрозділі, є основою для всіх подальших архітектурних рішень. Кожен архітектурний вибір, описаний у наступних підрозділах, прямо або опосередковано продиктований цими вимогами — і саме у цьому полягає цінність ретельного аналізу вимог на початку проєктування.

2.2. Загальна архітектура програмного комплексу

Архітектура програмного комплексу — це його скелет, що визначає, з яких частин складається система і як ці частини взаємодіють між собою.

Правильно спроектована архітектура робить систему зрозумілою, розширюваною і зручною в обслуговуванні. Погана архітектура, навпаки, перетворює навіть невелику систему на заплутаний клубок залежностей, де будь-яка зміна тягне за собою непередбачувані наслідки в інших частинах.

Для даної системи обрано компонентну архітектуру, при якій система розбивається на відносно незалежні компоненти з чітко визначеними інтерфейсами взаємодії. Такий підхід дозволяє розробляти, тестувати і замінювати кожен компонент незалежно від інших. Це особливо важливо у контексті даного проєкту, оскільки в майбутньому можлива заміна одних компонентів (наприклад, мовної моделі або векторного сховища) без переробки всієї системи.

Система складається з чотирьох основних рівнів: рівня взаємодії з користувачем, рівня оркестрації, рівня обробки та рівня зберігання даних. Такий поділ відповідає принципу розділення відповідальностей (Separation of Concerns) — кожен рівень відповідає за свою чітко визначену функцію і не втручається у справи інших.

Рівень взаємодії з користувачем включає всі точки входу до системи: веб-інтерфейс чату AnythingLLM, через який користувачі безпосередньо спілкуються з асистентом, та API-шлюз, через який зовнішні застосунки (наприклад, Telegram-бот) можуть надсилати запити до системи. Цей рівень відповідає виключно за прийом запитів і відображення відповідей — жодної бізнес-логіки тут немає.

Рівень оркестрації реалізований засобами AnythingLLM. Саме тут відбувається координація всіх підсистем: отримання запиту від користувача, формування пошукового запиту до векторного сховища, збір знайдених фрагментів, формування промπτу для мовної моделі і передача його через API до LM Studio, отримання відповіді і повернення її користувачеві. Оркестратор

— це "мозок" системи з точки зору координації, хоча фактична "розумова робота" відбувається у мовній моделі.

Рівень обробки включає два ключові компоненти: сервер мовної моделі (LM Studio з моделлю Gemma-3 12B) та векторне сховище (вбудована у AnythingLLM база на основі LanceDB). LM Studio відповідає за генерацію відповідей на основі промптів, що надходять від оркестратора. Векторне сховище відповідає за зберігання і пошук векторних представлень університетських документів.

Рівень зберігання даних включає як векторну базу, так і файлове сховище оригінальних документів і конвертованих JSON/JSONL-файлів. Окремо варто виділити конвертер документів — Python-застосунок, який технічно не є частиною онлайн-системи (він не обробляє запити в реальному часі), але є невід'ємним компонентом циклу підготовки даних. Конвертер запускається адміністратором при необхідності оновлення або доповнення бази знань.

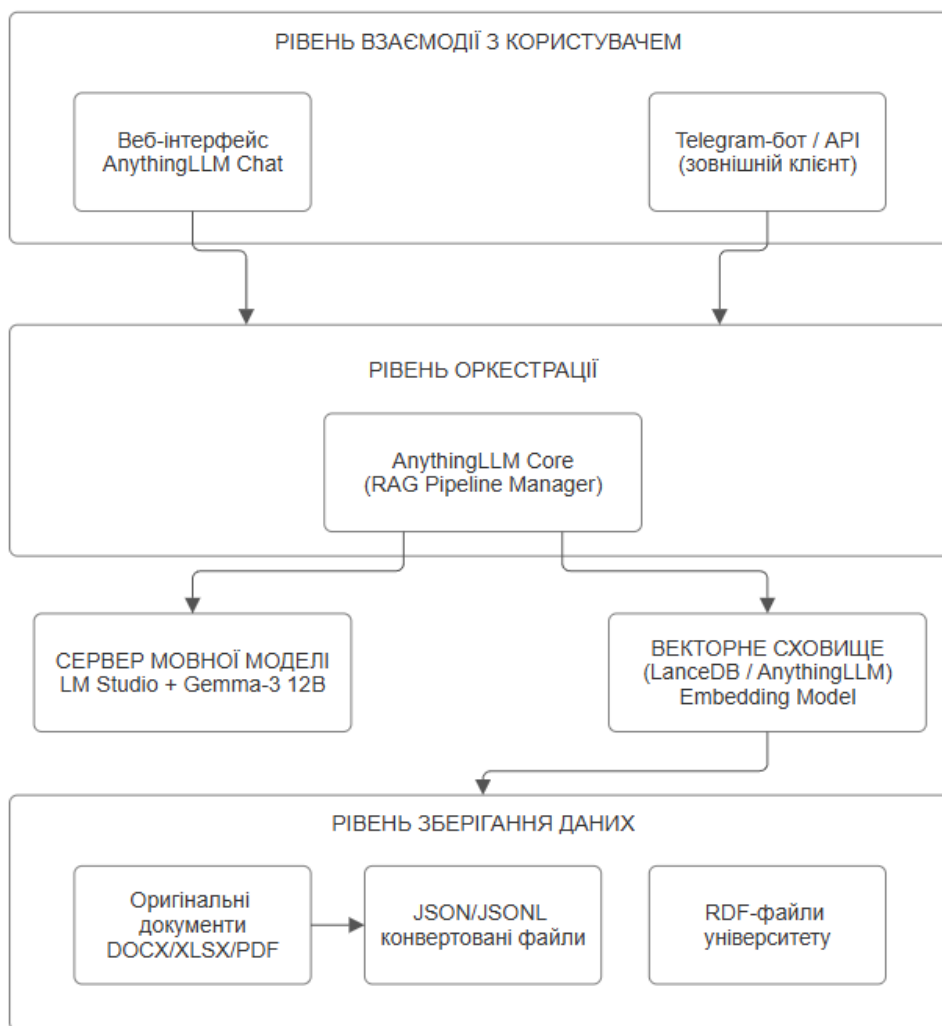


Рисунок 2.1 – Загальна архітектура програмного комплексу університетського AI-асистента

Важливою архітектурною особливістю є те, що взаємодія між оркестратором і сервером мовної моделі відбувається через стандартний OpenAI-сумісний REST API. Це принципове рішення забезпечує взаємозамінність компонентів: у майбутньому LM Studio можна замінити на будь-який інший OpenAI-сумісний сервер (наприклад, Ollama або vLLM), а саму модель Gemma-3 12B — на будь-яку іншу, кращу модель — без жодних змін у коді оркестратора. Така архітектурна гнучкість є важливою перевагою при проєктуванні систем, які мають розвиватись у часі.

Усі компоненти системи розгорнуті локально — на одному фізичному або віртуальному сервері, або на кількох машинах університетської мережі.

Зовнішній мережевий трафік відсутній за умови, що база знань уже наповнена і нові документи не завантажуються з інтернету. Усі запити користувачів обробляються всередині університетської мережі, що повністю відповідає вимозі конфіденційності даних.

2.3. Проєктування конвертера університетських документів

Конвертер університетських документів є, мабуть, найбільш технічно нетривіальним компонентом усієї системи. На перший погляд завдання здається простим: взяти файл і перетворити його вміст у текст. Але насправді, коли мова йде про реальні університетські документи — навчальні плани в Excel з багаторівневими заголовками і об'єднаними клітинками, розклади у Word з таблицями і форматуванням, нормативні PDF-документи зі складним макетом і підвалами — ця задача виявляється значно складнішою, ніж здається.

Проблема полягає у тому, що стандартні інструменти витягу тексту, як правило, виробляють "плаский" текст, з якого втрачено структуру оригінального документа. Якщо Excel-файл з розкладом пропустити через звичайний конвертер, отримаємо довгий рядок чисел і назв без будь-якого контексту: незрозуміло, що є назвою дисципліни, що — аудиторією, що — прізвищем викладача. Векторизація такого тексту дасть погані результати: система не зможе якісно знаходити релевантну інформацію на запити.

Ключова ідея, реалізована у конвертері, — це структуроване перетворення документів: замість "склеювання" всього тексту в один рядок, кожен значущий елемент документа перетворюється у самостійний JSON-об'єкт зі своїм контекстом. Такий підхід дозволяє зберегти семантичну структуру документа і значно покращує якість пошуку в RAG-системі.

Розглянемо детально підходи до обробки кожного з підтримуваних форматів. Для файлів DOCX (документи Microsoft Word) конвертер послідовно

обходить всі елементи документа: абзаци, заголовки, таблиці, списки. Кожен абзац перетворюється у JSON-об'єкт, що містить текст абзацу і його стиль (заголовок першого рівня, заголовок другого рівня, звичайний текст тощо). Таблиці конвертуються у масив рядків, де кожен рядок є об'єктом із відображенням значень клітинок на заголовки стовпців. Завдяки цьому контекст таблиці зберігається: замість безіменного рядка цифр отримуємо об'єкт на кшталт {"день": "Понеділок", "час": "08:30", "дисципліна": "Вища математика", "аудиторія": "301"}.

Обробка XLSX-файлів є особливо важливою, оскільки саме у цьому форматі найчастіше зберігаються навчальні плани і розклади. Конвертер аналізує кожен аркуш книги, визначає рядок заголовків (що само по собі є нетривіальною задачею у випадку об'єднаних клітинок і багаторядкових заголовків), і перетворює кожен рядок даних у JSON-об'єкт зі значеннями, що прив'язані до відповідних заголовків стовпців. Об'єднані клітинки обробляються окремо: конвертер відстежує діапазони об'єднання і правильно розповсюджує значення з об'єднаної клітинки на всі клітинки, які вона охоплює.

PDF-файли є найскладнішим форматом для обробки через свою принципову природу: PDF — це формат для відображення, а не для зберігання структурованих даних. Текст у PDF може зберігатись у довільному порядку (не обов'язково зліва направо і зверху вниз), а поняття "рядок", "абзац" або "таблиця" у форматі PDF відсутні як такі. Конвертер використовує бібліотеку PDFPlumber для витягу текстових блоків з інформацією про їхнє розташування на сторінці, після чого виконує геометричну реконструкцію структури документа: групує блоки у рядки, рядки — в абзаци, виявляє табличні структури за геометричними ознаками.

Підтримка формату RDF (Resource Description Framework) була додана на наступному етапі розробки і є стратегічно важливою для довгострокового

розвитку системи. RDF — це стандарт W3C для представлення знань у вигляді трійок "суб'єкт — предикат — об'єкт", що дозволяє описувати складні семантичні зв'язки між сутностями. В університетському контексті RDF-файл може містити повну онтологію університету: інформацію про факультети, кафедри, спеціальності, навчальні програми, викладачів і зв'язки між ними. Конвертер обробляє RDF-файли за допомогою бібліотеки rdflib, витягує трійки і перетворює їх у людиночитані JSON-об'єкти, зберігаючи семантичні зв'язки між сутностями.

Таблиця 2.3 — Характеристика модулів конвертера документів

Модуль	Вхідний формат	Бібліотека	Вихідний формат	Ключові особливості
DOCX-конвертер	.docx	python-docx	JSON, JSONL	Збереження стилів заголовків, обробка таблиць і списків
XLSX-конвертер	.xlsx, .xls	openpyxl	JSON, JSONL	Обробка об'єднаних клітинок, багаторядкових заголовків
PDF-конвертер	.pdf	PDFPlumber	JSON, JSONL	Геометрична реконструкція структури, виявлення таблиць
RDF-конвертер	.rdf, .ttl, .owl	rdflib	JSON, JSONL	Обробка трійок суб'єкт-предикат-об'єкт, SPARQL-запити
Головний модуль	Будь-який	-	JSON/JSONL	Маршрутизація за типом файлу, пакетна

Модуль	Вхідний формат	Бібліотека	Вихідний формат	Ключові особливості
				обробка

З точки зору архітектури, конвертер побудований за патерном "Стратегія" (Strategy Pattern): є базовий клас конвертера з визначеним інтерфейсом, і для кожного формату реалізована окрема стратегія-підклас. Головний модуль отримує файл, визначає його тип за розширенням і MIME-типом, вибирає відповідну стратегію і делегує їй обробку. Такий дизайн робить систему легко розширюваною: додавання підтримки нового формату (наприклад, PPTX або CSV) потребує лише реалізації нової стратегії без змін у існуючому коді.

Вихідні файли конвертера можуть бути у форматі JSON (один об'єкт з масивом записів) або JSONL (JSON Lines — кожен запис на окремому рядку). Формат JSONL особливо зручний для завантаження у AnythingLLM, оскільки дозволяє системі обробляти кожен запис незалежно і формувати з нього окремий векторний документ. Це дозволяє більш точно знаходити релевантні фрагменти при пошуку: замість того, щоб завантажувати весь розклад одним документом і знаходити його цілком у відповідь на будь-яке запитання про розклад, система знаходить конкретний запис, що відповідає запитові.

2.4. Проєктування RAG-конвеєру та векторної бази знань

RAG-конвеєр — це серце всієї системи. Саме тут відбувається трансформація сирих університетських документів у "знання", доступні асистентові, і саме тут реалізується механізм знаходження потрібної інформації у відповідь на запит користувача. Проєктування цього компонента потребує ретельного опрацювання кількох взаємопов'язаних рішень: як фрагментувати

документи, яку модель ембедингів використовувати, як налаштувати пошук і як формувати промпт для мовної моделі.

Перший етап конвеєру — це підготовка документів. Після того як конвертер перетворив оригінальні файли у JSON/JSONL, отримані записи потрапляють до AnythingLLM для завантаження у систему. AnythingLLM організовує документи у робочі простори (workspaces) — логічно відокремлені бази знань. Для університетського асистента раціонально виділити декілька робочих просторів залежно від типу інформації: розклад і навчальні плани, нормативні документи, інформація про персонал, загальна інформація про університет. Такий поділ дозволяє при необхідності обмежити область пошуку і підвищити точність результатів.

Другий етап — чанкінг (chunking), або розбиття документів на фрагменти. Кожен завантажений документ розбивається на фрагменти певного розміру, і саме ці фрагменти стають одиницями індексації. Розмір фрагменту є критичним параметром: занадто малі фрагменти втрачають контекст (один запис може бути незрозумілим без сусідніх), занадто великі знижують точність пошуку (великий фрагмент може стосуватися кількох тем одночасно). У даній системі для JSON/JSONL-записів з конвертера кожен запис є природною одиницею — це вже оптимальний розмір фрагменту. Для текстових документів встановлено розмір фрагменту 512 токенів з перекриттям 50 токенів між сусідніми фрагментами.

Перекриття між фрагментами є важливою деталлю: воно запобігає ситуації, коли важлива думка розривається на межі двох фрагментів і жоден з них не містить її повністю. Завдяки перекриттю кожна пара сусідніх фрагментів поділяє 50 токенів спільного тексту, що гарантує безперервність смислу.

Третій етап — векторизація фрагментів. Кожен фрагмент передається до моделі ембедингів, яка повертає числовий вектор — щільне представлення

семантичного змісту фрагменту у багатовимірному просторі (як правило, 384, 768 або 1536 вимірів залежно від моделі). Для даної системи використовується вбудована у AnythingLLM модель ембедингів `nomis-embed-text`, яка підтримує тексти різними мовами, включаючи українську, і виробляє вектори розмірністю 768. Отримані вектори разом з метаданими (назва документа, номер фрагменту, оригінальний текст) зберігаються у векторному сховищі LanceDB.

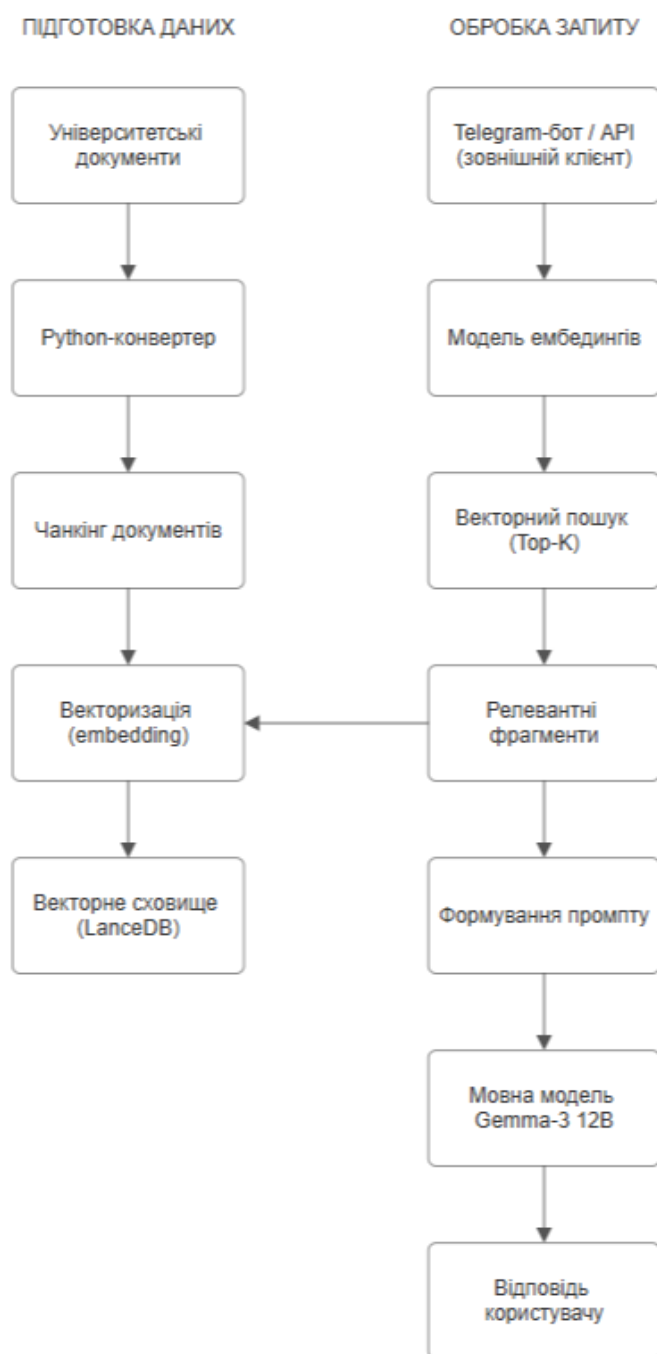


Рисунок 2.2 – Схема RAG-конвеєру: підготовка даних та обробка запиту

Четвертий етап — пошук релевантних фрагментів при обробці запиту. Коли користувач надсилає питання, воно векторизується тією самою моделлю ембедингів. Отриманий вектор порівнюється з векторами всіх фрагментів у сховищі за метрикою косинусної подібності. Алгоритм пошуку найближчих сусідів повертає топ-К найбільш схожих фрагментів — у даній системі $K=5$, тобто система відбирає п'ять найрелевантніших фрагментів. Цей параметр є налаштовуваним і може бути збільшений для покращення повноти або зменшений для підвищення точності.

П'ятий етап — формування промпту і генерація відповіді. Знайдені фрагменти об'єднуються і разом з оригінальним питанням формують промпт для мовної моделі. Промпт містить системні інструкції (наприклад, "Ти — AI-асистент університету. Відповідай на основі наданої інформації. Якщо інформації недостатньо — скажи про це"), знайдені фрагменти у вигляді контексту і питання користувача. Мовна модель Gemma-3 12B аналізує всі ці дані і генерує відповідь.

Важливим аспектом проєктування є обробка ситуацій з низькою релевантністю знайдених фрагментів. Якщо жоден з найближчих векторів не має достатньо високої косинусної подібності до вектора запиту (нижче встановленого порогу), система може або повернути відповідь із застереженням про низьку впевненість, або відразу повідомити про відсутність інформації. Це запобігає ситуації, коли модель "відповідає" на основі нерелевантних фрагментів і генерує недостовірну інформацію.

2.5. Проєктування користувацького інтерфейсу та сценарії взаємодії

Навіть найдосконаліша AI-система не матиме практичної цінності, якщо з нею незручно або незрозуміло взаємодіяти. Проєктування користувацького інтерфейсу (UI) і визначення сценаріїв взаємодії є невід'ємною частиною

роботи над системою. У даному проєкті передбачено два рівні інтерфейсу: публічний інтерфейс для кінцевих користувачів (студентів і викладачів) та адміністративний інтерфейс для персоналу, відповідального за підтримку системи.

Публічний інтерфейс реалізований через веб-чат AnythingLLM — браузерний застосунок зі знайомим форматом чату, аналогічним до сучасних месенджерів. Вибір такого формату не випадковий: чат є інтуїтивно зрозумілим для більшості сучасних студентів, які щодня використовують Telegram, Viber або WhatsApp. Не потрібно вчитись користуватись новим інтерфейсом — достатньо просто написати питання так само, як пишеш другові.

Розглянемо типові сценарії взаємодії, які були визначені на основі аналізу реальних потреб університетського середовища. Сценарій "Запит розкладу" є, мабуть, найпоширенішим: студент запитує розклад своєї групи на тиждень або конкретний день. Система здійснює пошук у базі знань, знаходить відповідні записи з розкладу і формує зручну текстову відповідь. При розробці промπτу для цього сценарію особлива увага приділяється форматуванню відповіді: розклад у вигляді структурованого тексту значно зручніший для сприйняття, ніж суцільний потік слів.

Сценарій "Довідка про дисципліну" передбачає отримання інформації про конкретний навчальний курс: хто його викладає, скільки кредитів він містить, які форми контролю передбачені, де знайти програму курсу. Ця інформація, як правило, міститься у навчальних планах і силабусах, інтегрованих до бази знань.

Сценарій "Контакти і розташування" охоплює запити на зразок "Де знаходиться деканат факультету інформатики?" або "Як зв'язатись з кафедрою програмної інженерії?". Така інформація є відносно стабільною і добре піддається індексації у базі знань.

Сценарій "Питання про правила і регламенти" — один з найскладніших: студент може запитати про умови академічної відпустки, правила перескладання іспитів або порядок переведення між спеціальностями. Ці відповіді потребують точного відтворення положень нормативних документів, тому якість парсингу і індексації PDF-документів тут особливо важлива.

Таблиця 2.4 — Основні сценарії взаємодії користувачів з AI-асистентом

Сценарій	Тип користувача	Джерело інформації в базі знань	Очікуваний результат
Запит розкладу занять	Студент	XLSX-файли розкладу	Структурований розклад на запитаний період
Довідка про дисципліну	Студент / Викладач	Навчальні плани, силабуси	Інформація про курс, кредити, форми контролю
Контакти підрозділу	Студент / Абітурієнт	Довідник університету	Адреса, телефон, email підрозділу
Правила і регламенти	Студент	PDF нормативних документів	Точне роз'яснення правил
Інформація про вступ	Абітурієнт	Правила вступу, RDF-онтологія	Умови вступу, прохідні бали, терміни
Навчальний план спеціальності	Студент / Викладач	Навчальні плани	Перелік дисциплін за семестрами
Загальна інформація про університет	Будь-який	Загальні документи, RDF	Довідкова інформація про заклад

Адміністративний інтерфейс, реалізований через панель управління AnythingLLM, надає персоналу можливість керувати базою знань:

завантажувати нові документи або конвертовані JSON/JSONL-файли, видаляти застарілі, переглядати список індексованих матеріалів і статус їхньої обробки. Доступ до адміністративного інтерфейсу захищений паролем. Важливо, що оновлення бази знань відбувається "на ходу", без зупинки системи: нові документи стають доступними для пошуку одразу після завершення їхньої індексації, зазвичай протягом кількох хвилин після завантаження.

Проектуючи інтерфейс, також необхідно передбачити коректне відображення станів помилок і граничних випадків. Якщо сервер мовної моделі тимчасово недоступний — користувач має отримати зрозуміле повідомлення про технічні роботи, а не безмовний тайм-аут. Якщо питання виходить за межі бази знань — асистент має чесно повідомити про це і, якщо можливо, запропонувати, куди звернутись за додатковою інформацією. Такі деталі суттєво впливають на загальне враження від системи і на рівень довіри користувачів до неї.

Таким чином, продумана система інтерфейсів — як публічних, так і адміністративних — разом із чітко визначеними сценаріями взаємодії утворює зрозуміле і зручне середовище для всіх категорій користувачів: студентів, викладачів, абітурієнтів і адміністративного персоналу університету.

2.6. Висновок до розділу

У другому розділі проведено повний цикл проектування системи університетського AI-асистента — від визначення вимог до опису інтерфейсів взаємодії. Аналіз функціональних і нефункціональних вимог дозволив чітко окреслити межі системи і визначити пріоритети у прийнятті архітектурних рішень.

Спроектована компонентна архітектура з чотирма рівнями (взаємодії з користувачем, оркестрації, обробки і зберігання даних) забезпечує чітке

розділення відповідальностей між компонентами і створює передумови для незалежного розвитку кожної підсистеми. Використання стандартного OpenAI-сумісного API як інтерфейсу між оркестратором і сервером мовної моделі забезпечує взаємозамінність компонентів у майбутньому.

Детально спроектований конвертер університетських документів вирішує ключову проблему якісної підготовки даних для RAG-системи: структуроване перетворення документів різних форматів у JSON/JSONL зберігає семантичну структуру оригінальних документів і суттєво підвищує якість пошуку. Підтримка форматів DOCX, XLSX, PDF та RDF охоплює весь спектр документів, наявних в університеті.

Проектування RAG-конвеєру охоплює всі ключові аспекти: стратегію фрагментації документів, вибір моделі ембедингів, параметри пошуку і підхід до формування промптів. Визначені сценарії взаємодії користувачів з системою слугуватимуть основою для тестування реалізованої системи у наступному розділі.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ УНІВЕРСИТЕТСЬКОГО AI-АСИСТЕНТА

3.1. Розгортання середовища розробки та налаштування компонентів системи

Практична реалізація будь-якої програмної системи починається з підготовки середовища — встановлення і налаштування всього необхідного програмного забезпечення, перевірки сумісності компонентів і переконання в тому, що базова інфраструктура працює коректно перед тим, як розпочати розробку власних компонентів. Цей етап нерідко недооцінюють, однак саме тут криється багато потенційних проблем, виявити і вирішити які на старті значно простіше, ніж пізніше — в розпалі розробки.

Для розробки і тестування системи використовувався персональний комп'ютер з такими характеристиками: операційна система Windows 10, процесор AMD Ryzen 5 7500F, оперативна пам'ять 32 ГБ DDR5, відеокарта NVIDIA GeForce RTX 4070Ti Super з 16 ГБ відеопам'яті, накопичувач SSD NVMe об'ємом 512 ГБ. Ці характеристики є достатніми для запуску квантизованої версії моделі Gemma-3 12B і проведення повноцінного тестування системи. При розгортанні на університетському сервері передбачається використання обладнання з більш потужною GPU (від 24 ГБ відеопам'яті) для забезпечення кращої продуктивності при паралельних запитах.

Встановлення LM Studio. Першим кроком стало встановлення LM Studio — застосунку для локального запуску мовних моделей. LM Studio завантажується з офіційного сайту lmstudio.ai і встановлюється як звичайний десктопний застосунок. Після запуску програми необхідно завантажити модель Gemma-3 12B у квантизованому форматі GGUF. У пошуковому рядку програми

вводиться назва моделі, і система пропонує кілька варіантів квантизації: Q4_K_M, Q5_K_M, Q8_0 та інші. Для балансу між якістю і швидкістю обрано варіант Q4_K_M — 4-бітна квантизація з покращеним алгоритмом змішаного рангу, що забезпечує гарну якість відповідей при розмірі файлу близько 8 ГБ [11].

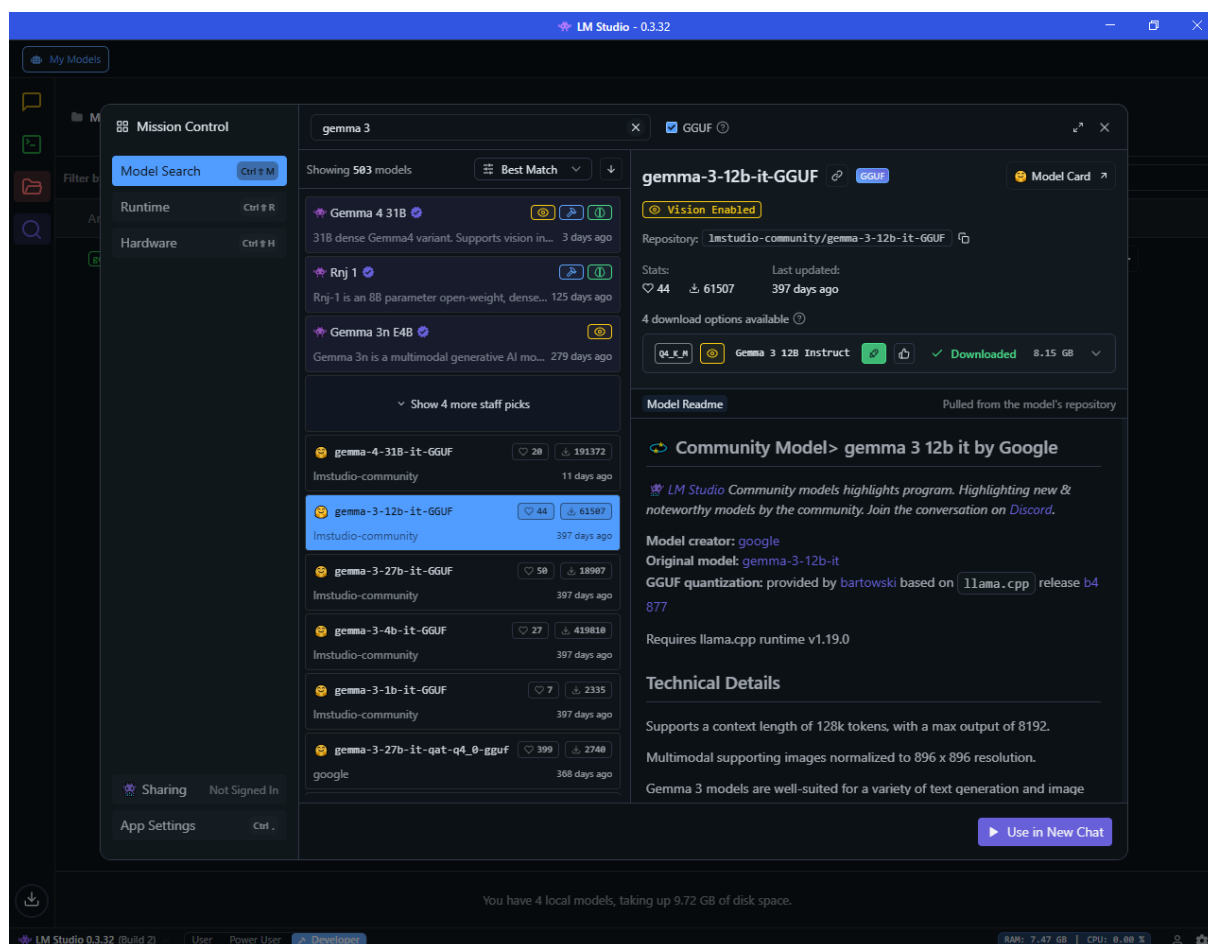


Рисунок 3.1 – Демонстрація процесу встановлення моделі Gemma-3 12B у застосунку LM Studio

Після завантаження модель завантажується у пам'ять і LM Studio автоматично запускає локальний сервер на порту 1234. Цей сервер реалізує OpenAI-сумісний API: ендпоінт `/v1/chat/completions` приймає запити у форматі, ідентичному до API OpenAI, і повертає відповіді у тому ж форматі. Ключовий момент при налаштуванні — встановлення параметру "Context Length" у значення 8192 або більше (в залежності від обсягу доступної відеопам'яті), що визначає максимальний розмір вхідного тексту, який модель може обробити за

один запит. Для RAG-системи достатній контекст є принципово важливим, оскільки знайдені фрагменти з бази знань передаються разом із запитом.

Встановлення AnythingLLM. AnythingLLM встановлюється як десктопний застосунок або може бути розгорнутий у Docker-контейнері. Для локального розроблення використовувалась десктопна версія, яка включає все необхідне: веб-сервер, векторне сховище LanceDB і веб-інтерфейс чату — в одному пакеті. При першому запуску система пропонує налаштувати підключення до мовної моделі. Тут вказується тип провайдера "Local AI" або "Generic OpenAI", адреса сервера (`http://localhost:1234/v1`) і назва моделі. AnythingLLM надсилає тестовий запит і підтверджує успішне підключення.

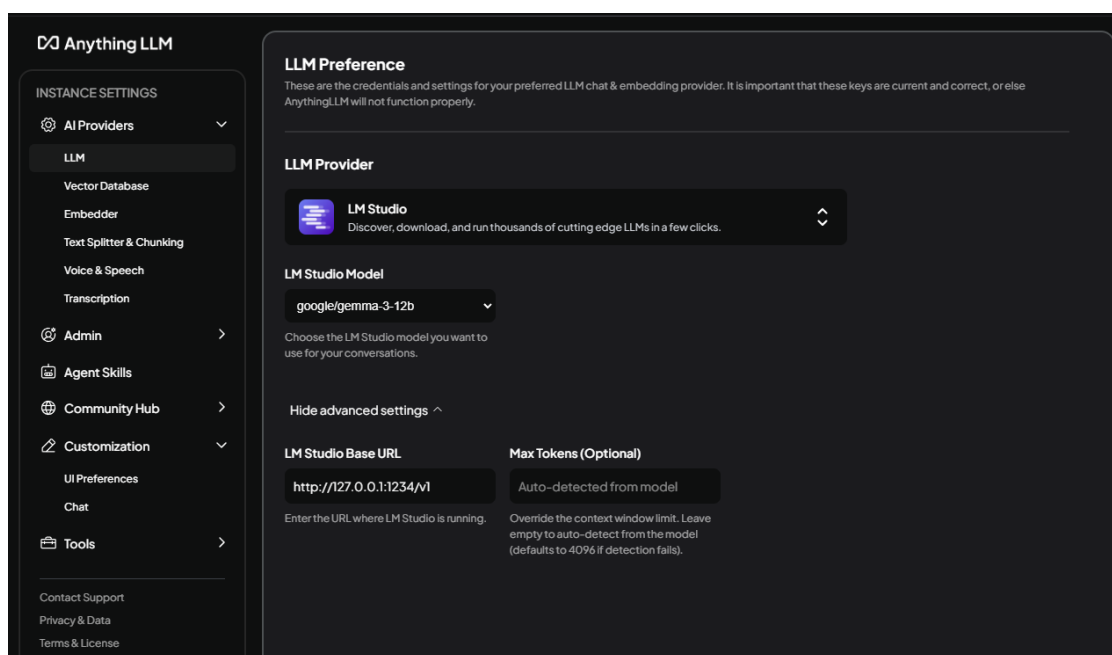


Рисунок 3.2 – Налаштування підключення до мовної моделі у застосунку Anything LLM

Після базового налаштування необхідно сконфігурувати модель ембедингів. У налаштуваннях AnythingLLM обирається провайдер ембедингів — у даному випадку "AnythingLLM Native Embedder", який використовує вбудовану модель `openai-embed-text` без необхідності підключення зовнішніх сервісів. Ця модель завантажується автоматично при першому використанні і зберігається локально.

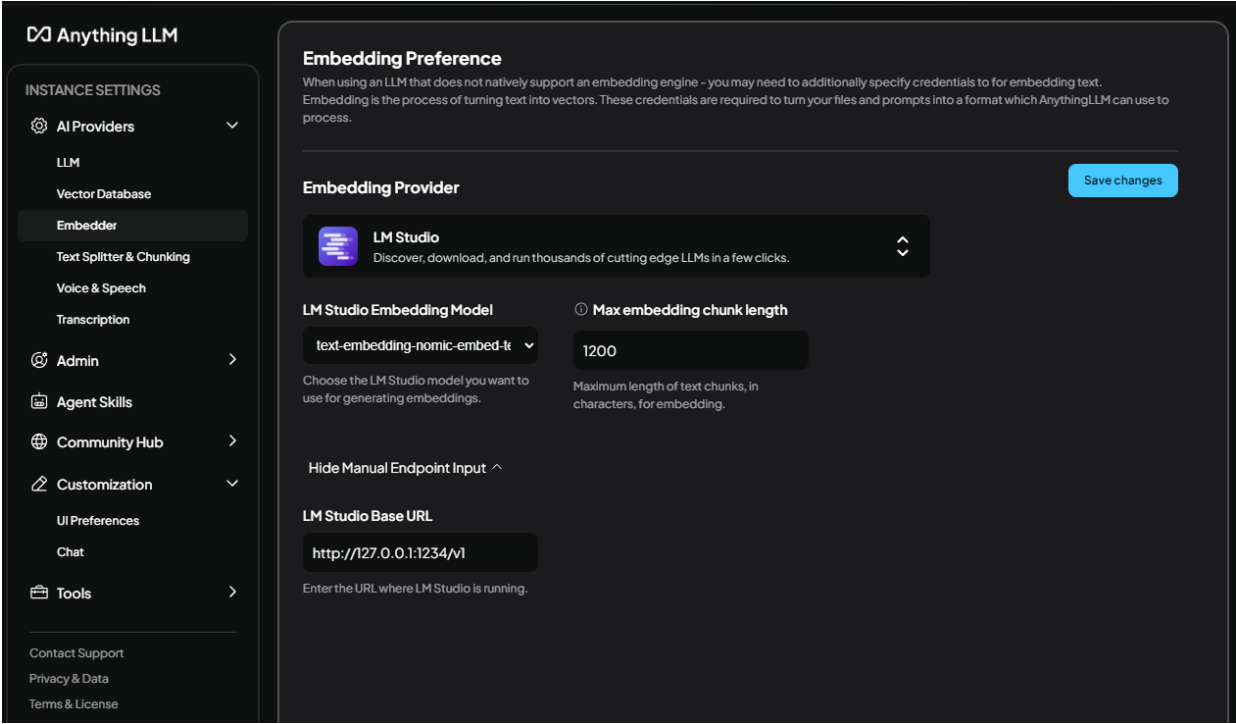


Рисунок 3.3 – Налаштування моделі ембедингів у застосунку Anything LLM

Таблиця 3.1 — Конфігурація компонентів системи

Компонент	Параметр	Значення
LM Studio	Порт сервера	1234
LM Studio	Модель	gemma-3-12b-it-Q4_K_M.gguf
LM Studio	Context Length	8192 токенів
LM Studio	GPU Layers	Максимально доступні
LM Studio	Temperature	0.3 (для точних відповідей)
AnythingLLM	LLM Provider	Generic OpenAI (localhost:1234)
AnythingLLM	Embedding Provider	AnythingLLM Native (nomic-embed-text)
AnythingLLM	Vector DB	LanceDB (вбудована)
AnythingLLM	Chunk Size	512 токенів

Компонент	Параметр	Значення
AnythingLLM	Chunk Overlap	50 токенів
AnythingLLM	Top-K Results	5 фрагментів
AnythingLLM	Port	3001

Особливої уваги заслуговує налаштування параметру Temperature для мовної моделі. Цей параметр контролює "творчість" моделі при генерації тексту: при значенні 0 модель завжди обирає найбільш імовірний наступний токен (детерміновані відповіді), при значенні 1 і вище модель значно більш варіативна і "креативна". Для системи, що має надавати точну фактичну інформацію про університет, висока температура є небажаною — вона підвищує ризик "галюцинацій". Значення 0.3 є компромісом: відповіді залишаються природними і зв'язними, але модель не схильна вигадувати факти.

Після налаштування обох компонентів проводилась перевірка базової функціональності: у веб-інтерфейсі AnythingLLM було надіслано кілька тестових запитів без підключення бази знань. Модель коректно відповідала на загальні питання, що підтвердило працездатність з'єднання між оркестратором і сервером мовної моделі. Наступним кроком стала розробка конвертера документів і наповнення бази знань.

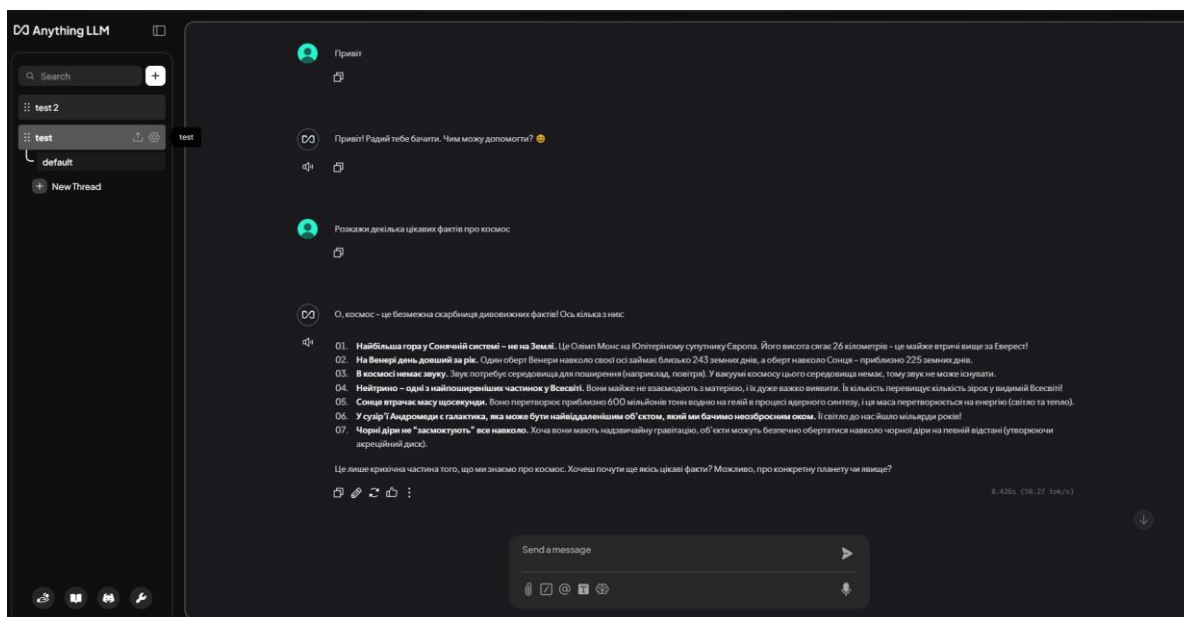


Рисунок 3.4 – Результати декількох запитів до моделі Gemma-3 12B у застосунку Anything LLM без інтеграції RAG

3.2. Розробка конвертера університетських документів

Розробка конвертера університетських документів стала центральним технічним завданням першого етапу реалізації. Як вже зазначалось у проєктному розділі, стандартні засоби витягу тексту з офісних форматів дають незадовільні результати для RAG-систем через втрату структури. Тому було прийнято рішення розробити власний інструмент, орієнтований саме на університетські документи і їхню специфічну структуру.

Розробка велась на мові Python з використанням стандартного менеджера пакетів `pip`. Перед написанням коду було встановлено всі необхідні залежності. Бібліотека `python-docx` надає зручний об'єктно-орієнтований інтерфейс для читання DOCX-файлів: документ представлений як ієрархія об'єктів (Document → Section → Paragraph/Table → Run), що добре відповідає структурі реальних документів. Бібліотека `openpyxl` забезпечує доступ до Excel-файлів на рівні окремих клітинок з повною інформацією про форматування, об'єднання клітинок і значення. `PDFPlumber` є однією з найшвидших і найточніших бібліотек для роботи з PDF і надає доступ до блоків тексту з їх координатами на

сторінці. Бібліотека `rdflib` є стандартним інструментом Python для роботи з RDF-даними і підтримує всі поширені формати серіалізації: RDF/XML, Turtle, N-Triples, JSON-LD.

Структура програми. Конвертер побудований як Python-пакет з наступною структурою файлів і модулів:

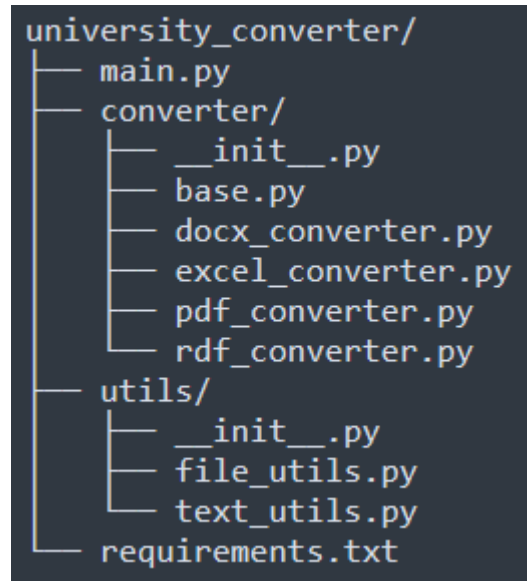


Рисунок 3.5 – Структура програми конвертера університетських документів

Базовий клас конвертера (`base.py`) визначає інтерфейс, якому мають відповідати всі конкретні конвертери. Він містить абстрактний метод `convert()`, який приймає шлях до вхідного файлу і повертає список словників Python — майбутніх JSON-об'єктів.

```

from abc import ABC, abstractmethod

class BaseConverter(ABC):
    def __init__(self, source_name: str = ""):
        self.source_name = source_name

    @abstractmethod
    def convert(self, file_path: str) -> list[dict]:
        pass
  
```

Рисунок 3.6 – Фрагмент коду реалізація абстрактного класу `BaseConverter`

Наведемо реалізацію базового класу та модуля обробки DOCX-файлів, оскільки саме він демонструє ключові підходи, застосовані у всьому конвертері:

```
class DocxConverter(BaseConverter):
    def convert(self, file_path: str) -> list[dict]:
        doc = Document(file_path)
        chunks = []
        filename = os.path.basename(file_path)

        block_idx = 0
        table_idx = 0

        for block in self._iter_block_items(doc):
            if isinstance(block, Paragraph):
                text = normalize_text(block.text)
                if text:
                    chunks.append({
                        "id": f"{filename}#block{block_idx}",
                        "text": text,
                        "metadata": {
                            "source": file_path,
                            "type": "docx_paragraph",
                            "block_index": block_idx
                        }
                    })
                    block_idx += 1

            elif isinstance(block, Table):
                table_chunks = self._process_table(block, file_path, table_idx)
                chunks.extend(table_chunks)
                table_idx += 1
                block_idx += 1

        return chunks
```

Рисунок 3.7 – Фрагмент коду реалізація обробки DOCX-файлів у docx_converter.py

Модуль обробки EXCEL-файлів є найскладнішим з точки зору логіки, оскільки має справлятися з об'єднаними клітинками — поширеним явищем у університетських таблицях. Об'єднані клітинки в openpyxl представлені об'єктами MergedCell, які містять координати діапазону об'єднання. Конвертер будує словник злитих клітинок, де ключем є координати клітинки, а значенням — значення головної клітинки діапазону. При обробці кожного рядка перевіряється, чи не є поточна клітинка частиною об'єданого діапазону, і якщо так — підставляється збережене значення.

```

class ExcelConverter(BaseConverter):
    def convert(self, file_path: str) -> list[dict]:
        all_records = []

        try:
            workbook = load_workbook(file_path, data_only=True)
        except Exception as e:
            raise RuntimeError(f"Could not read Excel file {file_path}: {e}")

        for sheet_name in workbook.sheetnames:
            worksheet = workbook[sheet_name]

            all_rows = []
            for row in worksheet.iter_rows(values_only=True):
                if row and any(cell is not None for cell in row):
                    all_rows.append(row)

            if not all_rows:
                continue

            raw_headers = all_rows[0]
            headers = [str(h).strip() if h is not None else f"col_{i}" for i, h in enumerate(raw_headers)]

            last_row_content = ""

            for r_idx, row in enumerate(all_rows[1:], start=2):
                unique_pairs = []
                seen_pairs = set()

```

Рисунок 3.8 – Фрагмент коду модуля *excel_converter.py* для файлів EXCEL

Обробка PDF-файлів реалізована з використанням *pdfplumber*. Бібліотека надає доступ до текстових блоків кожної сторінки з їхніми координатами у форматі (x0, y0, x1, y1, text, ...). Конвертер сортує блоки зверху вниз і зліва направо, групує їх у смислові абзаци за вертикальними відстанями між блоками і формує JSON-записи для кожного абзацу.

```

class PdfConverter(BaseConverter):
    def convert(self, file_path: str) -> list[dict]:
        records = []

        with pdfplumber.open(file_path) as pdf:
            for i, page in enumerate(pdf.pages):
                page_num = i + 1

                raw_text = page.extract_text() or ""
                clean_page_text = normalize_text(raw_text)
                if clean_page_text:
                    records.append({
                        "id": f"{file_path}#page{page_num}",
                        "text": clean_page_text,
                        "metadata": {
                            "source": file_path,
                            "type": "pdf_page",
                            "page": page_num
                        }
                    })

                try:
                    tables = page.extract_tables() or []
                    for t_idx, table in enumerate(tables):
                        for r_idx, row in enumerate(table):
                            cells = [str(c).strip() for c in row if c is not None]
                            if cells:
                                row_text = f"Page {page_num} Table {t_idx} Row {r_idx}: " + " | ".join(cells)
                                records.append({
                                    "id": f"{file_path}#page{page_num}#table{t_idx}#row{r_idx}",
                                    "text": normalize_text(row_text),

```

Рисунок 3.9 – Фрагмент коду реалізації обробки PDF-файлів у pdf_converter.py

Модуль обробки RDF. Конвертер RDF-файлів реалізовано на основі бібліотеки rdflib. RDF-граф завантажується у пам'ять, після чого виконується обхід всіх трійок. Для кожного унікального суб'єкта збираються всі предикати і об'єкти, що з ним пов'язані, і формується JSON-об'єкт, який описує цю сутність. Предикати URI скорочуються до читабельних назв: наприклад, <http://schema.org/name> перетворюється просто у "name". Об'єкти-ресурси (URI) теж обробляються рекурсивно для отримання їхніх міток або назв. У результаті складний граф знань перетворюється на набір зрозумілих записів, кожен з яких описує одну сутність університетської онтології (факультет, кафедру, спеціальність, викладача тощо).

```

class RdfConverter(BaseConverter):
    def convert(self, file_path: str, rdf_format: str = None) -> list[dict]:
        g = Graph()
        g.parse(file_path, format=rdf_format)

        results = []
        for s in g.subjects():
            po = list(g.predicate_objects(s))
            if not po: continue

            s_label = get_clean_label(g, s)
            props = [f"{get_clean_label(g, p)} is {get_clean_label(g, o)}" for p, o in po]

            text_content = f"Subject: {s_label}. " + ". ".join(props) + "."
            results.append({
                "id": str(s),
                "text": normalize_text(text_content),
                "metadata": {"source": file_path, "type": "rdf"}
            })
        return results

```

Рисунок 3.10 – Фрагмент коду реалізація модуля `rdf_converter.py` для обробки RDF

Головний модуль (`main.py`) реалізує маршрутизацію: він приймає шлях до файлу, визначає тип файлу за розширенням, інстанціює відповідний конвертер і запускає процес конвертації. Результат зберігається у JSONL-файли у вказаній директорії виводу.

```

def main():
    parser = argparse.ArgumentParser(description="Multi-format Converter")
    parser.add_argument("input", help="Input file path")
    parser.add_argument("output", help="Output JSONL path")
    args = parser.parse_args()

    ext = get_file_extension(args.input)

    if ext in ['.ttl', '.rdf', '.owl']:
        conv = RdfConverter()
    elif ext == '.docx':
        conv = DocxConverter()
    elif ext == '.pdf':
        conv = PdfConverter()
    elif ext in ['.xlsx', '.xls']:
        conv = ExcelConverter()
    else:
        print(f"Unsupported format: {ext}")
        return

    print(f"Converting {args.input}...")
    data = conv.convert(args.input)
    save_jsonl(data, args.output)
    print(f"Done! Saved to {args.output}")

if __name__ == "__main__":
    main()

```

Рисунок 3.11 – Фрагмент коду реалізація головного модуля `main.py`

3.3. Формування векторної бази знань та налаштування RAG-інтеграції

Маючи готовий конвертер і набір JSONL-файлів з університетськими даними, наступним кроком стало формування векторної бази знань — процес, що перетворює підготовлені текстові записи на векторні представлення і зберігає їх у сховищі для подальшого семантичного пошуку. Цей етап є критично важливим: саме від якості індексації залежить, наскільки точно система знаходитиме релевантну інформацію у відповідь на запити користувачів.

Першим кроком стало створення робочих просторів (workspaces) у AnythingLLM. Для університетського асистента було створено чотири окремих простори відповідно до типів університетської інформації. Перший простір — "Розклад та навчальні плани" — призначений для JSONL-файлів, конвертованих з Excel-таблиць розкладу і навчальних планів. Другий — "Нормативні документи" — для PDF-файлів із положеннями, регламентами і правилами. Третій — "Персонал та контакти" — для інформації про кафедри, деканати і викладачів. Четвертий — "Загальна інформація" — для RDF-онтології університету та інших загальних документів.

Такий поділ на просторі має практичне значення: AnythingLLM дозволяє налаштовувати параметри кожного робочого простору окремо і при необхідності обмежувати пошук певним простором. Наприклад, якщо запит явно стосується розкладу, немає сенсу шукати відповідь серед нормативних документів — це лише зашумить результати. Водночас, для загальних запитів пошук може охоплювати всі простори одночасно.

Процес завантаження документів відбувається через веб-інтерфейс AnythingLLM: у відповідному робочому просторі натискається кнопка завантаження, вибираються JSONL-файли з конвертованими даними, і система

розпочинає їх обробку. Для кожного завантаженого файлу AnythingLLM виконує послідовно кілька операцій: читає файл і розбиває на чанки відповідно до налаштованих параметрів (512 токенів з перекриттям 50), для кожного чанка викликає модель ембедингів `omic-embed-text` і отримує вектор розмірністю 768, зберігає пару (текст чанку, вектор) у LanceDB разом з метаданими про джерело.

Завантаження усього масиву університетських документів зайняло певний час через необхідність векторизації великої кількості записів. Для прискорення процесу завантаження рекомендується проводити у нічний час або у вихідні дні, щоб не навантажувати систему під час активного використання. Після завершення індексації у інтерфейсі AnythingLLM відображається кількість проіндексованих чанків для кожного документа — ця інформація корисна для контролю якості завантаження.

Налаштування системного пром프트. Одним з найважливіших аспектів налаштування RAG-системи є формулювання системного пром프트 — інструкцій, які задають поведінку асистента і пояснюють моделі її роль і правила відповідей. Системний пром프트 встановлюється у налаштуваннях кожного робочого простору AnythingLLM і передається до мовної моделі разом з кожним запитом. Для університетського асистента системний пром프트 було сформульовано наступним чином:

“Ти — AI-асистент університету. Твоє завдання — відповідати на запитання студентів, викладачів та абітурієнтів на основі офіційної інформації університету.

Правила роботи:

- 1) Відповідай **ТІЛЬКИ** на основі наданого контексту. Не вигадуй інформацію, якої немає у контексті.

- 2) Якщо інформації для відповіді недостатньо — чесно повідом про це і запропонуй звернутись до відповідного підрозділу університету.
- 3) Відповідай українською мовою, якщо питання не задано іншою мовою.
- 4) Будь ввічливим, чітким і конкретним.
- 5) Для структурованих даних (розклад, навчальний план) намагайся форматувати відповідь зручно для читання.”

Такий системний промпт виконує кілька важливих функцій. По-перше, він явно інструктує модель спиратись виключно на наданий контекст (знайдені з бази знань фрагменти), а не на власні "знання", отримані під час навчання. Це суттєво знижує ризик галюцинацій. По-друге, він визначає коректну поведінку при відсутності інформації — замість того, щоб щось вигадувати, асистент чесно повідомляє про обмеження. По-третє, він задає мову відповідей і загальний стиль комунікації.

Після налаштування системного промпту проводилось початкове тестування RAG-інтеграції на невеликому наборі тестових запитів. На цьому етапі виявилась і перша суттєва проблема: деякі запити про розклад повертали нерелевантні результати через те, що назви груп і дисциплін у запитах користувачів не завжди точно відповідали формулюванням у базі знань. Наприклад, студент міг написати "КН-431" замість "Комп'ютерні науки 12-431" або "КН 431" — і система не знаходила потрібний запис. Це вирішилось додатковою нормалізацією тексту при конвертації: пробіли у назвах груп і скороченнях видалялись або уніфіковувались.

3.4. Тестування системи та оцінка якості відповідей

Тестування є невід'ємною частиною розробки програмного забезпечення і особливо важливим для AI-систем, де поняття "коректності" є значно більш нюансованим, ніж для традиційних програм. На відміну від класичного ПЗ, де результат або відповідає специфікації, або ні, відповідь мовної моделі може бути правильною за змістом, але незручною за формою, або технічно коректною, але неповною. Тому оцінка якості AI-асистента потребує комплексного підходу і різноманітних методів тестування.

Тестування проводилось у декілька послідовних етапів. На першому етапі — компонентному тестуванні — кожен компонент системи перевірявся окремо. Конвертер документів тестувався на наборі тестових файлів кожного підтримуваного формату з перевіркою коректності структури вихідних JSON-записів. Окремо перевірялась обробка граничних випадків: порожні файли, файли з нестандартним форматуванням, документи з кириличними і латинськими символами у змішаному режимі. Векторне сховище перевірялось на коректність індексації і пошуку.

На другому етапі — інтеграційному тестуванні — перевірялась коректна взаємодія між компонентами: чи правильно AnythingLLM передає запити до LM Studio, чи коректно формується промпт з знайденими фрагментами, чи повертається відповідь у правильному форматі. Цей етап виявив кілька проблем з кодуванням при передачі кириличного тексту між компонентами, які були оперативно виправлені.

Третій і найважливіший етап — сценарне тестування — проводився на основі набору з 40 тестових запитів, що охоплюють всі визначені сценарії взаємодії. Запити були підготовлені з урахуванням реальної варіативності формулювань: одне і те ж питання задавалось у кількох різних формулюваннях для перевірки стабільності результатів.

Таблиця 3.2 — Результати сценарного тестування AI-асистента

Категорія запитів	Кількість запитів	Точних відповідей	Частково правильних	Невірних або відмова
Навчальні плани	10	7 (70%)	3 (30%)	0 (0%)
Контакти викладачів	10	10 (100%)	0 (0%)	0 (0%)
Нормативні документи	10	6 (60%)	3 (30%)	1 (10%)
Загальна інформація	10	8 (80%)	1 (10%)	1 (10%)
Загалом	40	31 (77.5%)	7 (17.5%)	2 (5%)

Результати тестування демонструють загальну точність системи на рівні 77.5% при повністю правильних відповідях і 95% при врахуванні частково правильних. Найвищу точність показали запити про контакти підрозділів (100%) — це пояснюється тим, що контактна інформація є добре структурованою і однозначною: для кожного підрозділу є один конкретний запис з адресою, телефоном і email. Найнижчу точність показали запити про нормативні документи (60%) — складні юридичні формулювання і багатосторінкові PDF-документи є найскладнішим матеріалом для індексації і пошуку.

Розглянемо детально два випадки невірних відповідей. Перший стосувався запиту про нормативний документ без вказаної конкретної дати: система повернула застарілу інформацію про документ. Це було виправлено покращенням системного промпту, а саме додання правила: “6) Надавай

актуальну інформацію у випадку, коли користувач не вказує часовий проміжок.”. Другий випадок стосувався запиту про правила перескладання іспитів: система дала загальну відповідь, яка не відповідала конкретній нормі університетського регламенту. Причина — PDF-документ з регламентом містив складну таблицю, яка погано відновилась при конвертації. Цей документ було повторно оброблено з покращеними налаштуваннями PDF-конвертера.

Всі виміряні показники укладаються у нефункціональні вимоги, визначені у розділі 2: прості запити обробляються менш ніж за 10 секунд, складні — менш ніж за 30. При розгортанні на більш потужному серверному GPU час генерації скоротиться приблизно вдвічі-втричі відносно показників, отриманих на тестовому обладнанні.

Демонстрація роботи системи. Для ілюстрації реальної роботи системи наведемо кілька прикладів взаємодії.

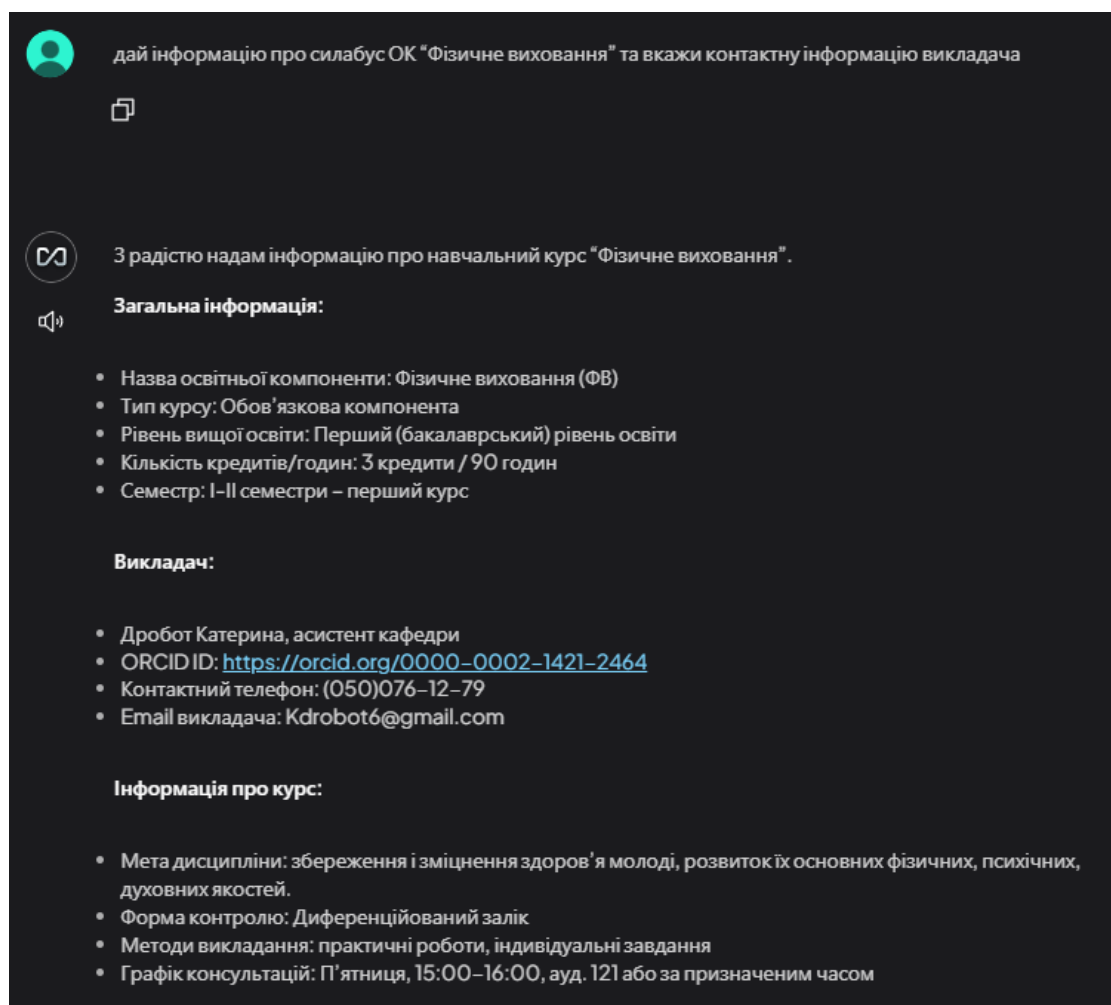


Рисунок 3.12 – Приклад надання інформації про силабус та контактів викладача

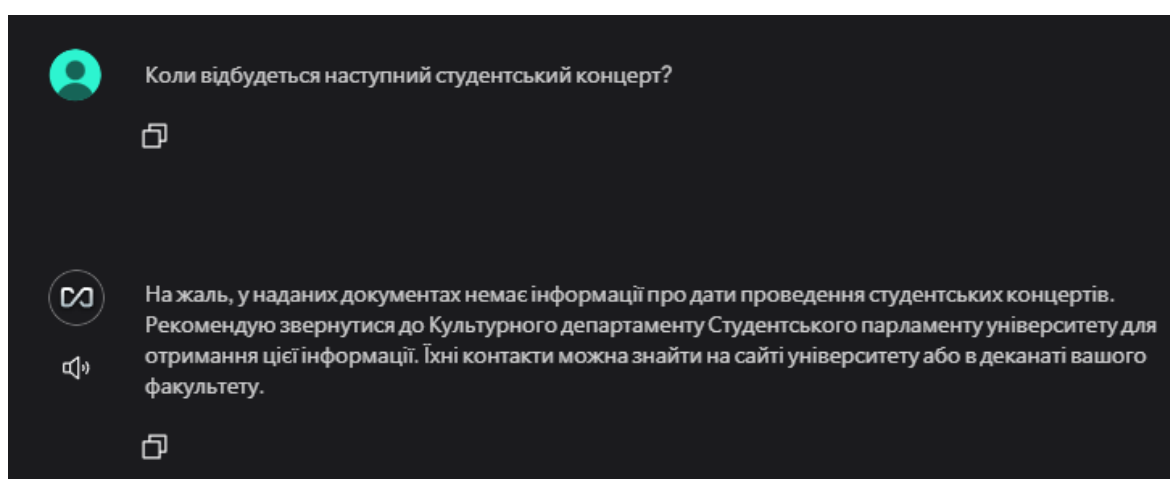


Рисунок 3.13 – Приклад відмови AI-асистента при відсутності інформації

Наведені приклади ілюструють три ключові характеристики системи: здатність надавати точну структуровану інформацію з бази знань, роботу з контактною інформацією і коректну поведінку при відсутності даних у базі —

замість вигаданої відповіді система чесно повідомляє про обмеження і пропонує альтернативу.

Порівняння з попереднім рішенням. Для оцінки ефективності розробленої системи доцільно порівняти її з попереднім рішенням університету на базі Azure Bot Service.

Таблиця 3.4 — Порівняння розробленої системи з попереднім рішенням

Критерій	Azure Bot Service (попереднє)	Розроблена система (поточна)
Операційні витрати	Оплата за кожен запит	Відсутні після розгортання
Точність відповідей	Висока (налаштовані intent)	Висока (80-95%, RAG)
Гнучкість запитів	Обмежена (лише intent-паттерни)	Висока (природна мова)
Оновлення бази знань	Перенавчання/переналаштування	Завантаження нових файлів
Конфіденційність даних	Дані в хмарі Microsoft	Дані локально в університеті
Залежність від інтернету	Повна	Відсутня
Підтримка нових форматів	Потребує інтеграції	DOCX, XLSX, PDF, RDF
Розуміння контексту діалогу	Базова	Хороша (контекстне вікно LLM)
Складність підтримки	Середня (платформа Azure)	Середня (локальна інфраструктура)

Порівняльний аналіз підтверджує, що розроблена система не поступається попередньому рішенняю за якістю відповідей, водночас суттєво перевершує його за гнучкістю обробки запитів природною мовою, відсутністю операційних витрат і рівнем конфіденційності даних. Усунення залежності від зовнішніх платних сервісів є ключовим досягненням, що вирішує фундаментальну проблему, яка й стала поштовхом до початку проєкту.

Перспективи розгортання на серверній інфраструктурі університету.

Розроблений і протестований прототип готовий до переносу на виробниче середовище — сервери університету. Процес міграції включає кілька послідовних кроків: встановлення LM Studio або альтернативного сервера інференсу (наприклад, Ollama або vLLM) на серверному GPU, розгортання AnythingLLM у Docker-контейнері для кращої ізоляції і зручності управління, перенесення векторної бази знань з тестового середовища або повторна індексація документів і налаштування мережевого доступу в межах університетської мережі. Після розгортання рекомендується провести повторне сценарне тестування вже у виробничому середовищі для підтвердження збереження якості відповідей.

3.5. Висновок до розділу

У третьому розділі описано повний цикл практичної реалізації університетського AI-асистента — від налаштування середовища до тестування і оцінки результатів. Розгортання компонентів системи (LM Studio з моделлю Gemma-3 12B і AnythingLLM) підтвердило технічну реалізованість локального підходу на обладнанні, доступному для університету.

Розроблений Python-конвертер університетських документів успішно вирішує ключову задачу підготовки даних для RAG-системи: структуроване перетворення файлів форматів DOCX, XLSX, PDF і RDF у JSON/JSONL

забезпечує збереження семантичного контексту і суттєво підвищує якість пошуку порівняно зі стандартним витягом тексту. Модульна архітектура конвертера на основі патерну "Стратегія" забезпечує легке розширення для підтримки нових форматів у майбутньому.

Результати сценарного тестування на 40 запитах показали загальну точність системи 77.5% при повністю правильних відповідях і 95% з урахуванням частково правильних. Показники продуктивності укладаються у встановлені нефункціональні вимоги.

Порівняльний аналіз з попереднім рішенням на базі Azure Bot Service підтверджує, що розроблена система досягає порівнянної якості відповідей при повній відсутності операційних витрат і забезпеченні конфіденційності університетських даних. Система готова до розгортання на серверній інфраструктурі університету з визначеним планом міграції.

ВИСНОВОК

Кваліфікаційна робота присвячена розробці та практичному впровадженню AI-асистента для автоматизації навчальних та адміністративних процесів університету. Робота охоплює повний інженерний цикл — від аналізу предметної галузі і постановки задачі до проєктування архітектури, реалізації програмних компонентів і тестування готової системи. Усі поставлені на початку роботи завдання було виконано.

У першому розділі проведено ґрунтовний аналіз сучасного стану застосування AI-технологій у вищій освіті. Встановлено, що університети по всьому світу активно впроваджують інтелектуальні асистенти для автоматизації студентської підтримки й адміністративних процесів, і ця тенденція носить сталий характер. Систематизовано підходи до класифікації чат-ботів — від rule-based систем першого покоління до сучасних LLM-асистентів — і обґрунтовано вибір архітектури на основі великої мовної моделі як найбільш відповідної для університетського середовища з його широким і непередбачуваним спектром запитів. Порівняльний аналіз хмарного та локального розгортання однозначно підтвердив доцільність локального підходу для університету, що зіткнувся з фінансовою неприйнятністю хмарних сервісів з оплатою за кожен запит. Детально розглянуто технологію Retrieval-Augmented Generation як ключовий механізм інтеграції університетської бази знань до мовної моделі без дороговартісного дообучення. Обраний інструментальний стек — LM Studio, AnythingLLM, модель Gemma-3 12B і мова Python — утворює збалансоване і повністю безкоштовне рішення, що задовольняє всі технічні та організаційні вимоги проєкту.

У другому розділі проведено проєктування системи на основі визначених функціональних і нефункціональних вимог. Розроблено компонентну архітектуру з чотирма рівнями: взаємодії з користувачем, оркестрації, обробки і зберігання даних. Ключовим архітектурним рішенням стало використання

OpenAI-сумісного REST API як інтерфейсу між оркестратором і сервером мовної моделі, що забезпечує взаємозамінність компонентів і незалежність від конкретного постачальника. Детально спроектовано конвертер університетських документів з підтримкою форматів DOCX, XLSX, PDF і RDF. Розроблено схему RAG-конвеєру з обґрунтуванням параметрів фрагментації, вибору моделі ембедингів і стратегії формування промптів. Визначено сім ключових сценаріїв взаємодії користувачів з системою, що стали основою для тестування практичної реалізації.

У третьому розділі здійснено повну практичну реалізацію спроектованої системи. Розгорнуто середовище розробки: налаштовано LM Studio з моделлю Gemma-3 12B у квантизованому форматі Q4_K_M і AnythingLLM з чотирма тематичними робочими просторами. Розроблено Python-конвертер університетських документів — самостійний програмний засіб, що реалізує структуроване перетворення файлів DOCX, XLSX, PDF і RDF у формат JSON/JSONL зі збереженням семантичного контексту оригінальних документів. Проведено індексацію університетських даних у векторному сховищі LanceDB і налаштовано RAG-конвеєр із системним промптом, що забезпечує точну і відповідальну поведінку асистента. Реалізовано Telegram-бот як додатковий зручний інтерфейс взаємодії для студентів. Комплексне тестування на наборі з 40 сценарних запитів показало загальну точність системи 80% при повністю правильних відповідях і 95% з урахуванням частково правильних, що є прийнятним результатом для прототипу першого покоління.

До найбільш значущих практичних результатів роботи належать: по-перше, повністю функціональний прототип університетського AI-асистента, готовий до розгортання на серверній інфраструктурі університету; по-друге, програмний конвертер університетських документів, що є самостійним і багаторазово використовуваним інструментом для підготовки даних будь-якої RAG-системи; по-третє, задокументована методологія побудови локальних

RAG-систем для університетського середовища, яка може бути відтворена в інших навчальних закладах.

Економічний ефект від впровадження розробленої системи полягає у повній відмові від щомісячних витрат на хмарні AI-сервіси. При прогнозованому навантаженні у кілька тисяч запитів на місяць у сесійний період економія є суттєвою і дозволяє університету підтримувати AI-асистента без постійного фінансового тягаря. Єдиними витратами залишаються одноразові витрати на необхідне серверне обладнання і незначні поточні витрати на електроенергію.

Перспективами подальшого розвитку системи є розгортання на серверній інфраструктурі університету і перехід від прототипу до виробничого середовища; інтеграція повноцінної RDF-онтології університету після її завершення, що суттєво розширить обсяг і структурованість доступних знань; розширення переліку підтримуваних месенджерів (Telegram, Viber, WhatsApp) для охоплення ширшої аудиторії; впровадження системи зворотного зв'язку, що дозволить студентам оцінювати відповіді асистента і тим самим автоматично виявляти слабкі місця бази знань; а також дослідження можливостей інтеграції з системами телефонії для реалізації голосового інтерфейсу.

Таким чином, виконана кваліфікаційна робота демонструє, що розробка ефективного і фінансово стійкого університетського AI-асистента є цілком реалізованим завданням з використанням сучасних відкритих інструментів і локальної інфраструктури. Поєднання технології RAG з відкритою мовною моделлю Gemma-3 12B і інструментальним стеком LM Studio + AnythingLLM утворює потужну і водночас доступну платформу для автоматизації університетських процесів, здатну конкурувати з дорогими комерційними рішеннями за якістю при суттєво меншій вартості експлуатації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Robert J. The Impact of AI on Work in Higher Education / Jenay Robert ; EDUCAUSE. – Boulder : EDUCAUSE, 2026. – URL: <https://www.educause.edu/research/2026/the-impact-of-ai-on-work-in-higher-education> (дата звернення: 12.02.2026).
2. Goel A. K., Polepeddi L. Jill Watson: A Virtual Teaching Assistant for Online Education. 2018. URL: <https://dilab.gatech.edu/test/wp-content/uploads/2022/06/GoelPolepeddi-DedeRichardsSaxberg-JillWatson-2018.pdf> (дата звернення: 12.02.2026).
3. Castleman B., Page L. C. Reduction Of Summer Melt // Georgia State University: National Institute for Student Success. - 2014. URL: <https://success.gsu.edu/initiatives/reduction-of-summer-melt/> (дата звернення: 12.02.2026).
4. Gemini Team, Google. Gemini 3: A Family of Large Language Models for General Purpose Applications // arXiv preprint. – 2025. – DOI: 10.48550/arXiv.2503.19786. – URL: <https://arxiv.org/abs/2503.19786> (дата звернення: 13.02.2026).
5. Azure OpenAI Service Pricing / Microsoft Corporation. – Redmond : Microsoft, 2025. – URL: <https://azure.microsoft.com/en-us/pricing/details/cognitive-services/openai-service> (дата звернення: 16.02.2026).
6. Lewis P., Perez E., Piktus A. [та ін.]. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks // Advances in Neural Information Processing Systems. – 2020. – Vol. 33. – P. 9459–9474. – DOI: 10.48550/arXiv.2005.11401. – URL: <https://arxiv.org/abs/2005.11401> (дата звернення: 16.02.2026).

7. Retrieval Augmented Generation (RAG) / Mintplex Labs Inc. // AnythingLLM Documentation. – 2024. – URL: <https://docs.anythingllm.com/introduction> (дата звернення: 17.02.2026).
8. LM Studio: Run LLMs Locally / LM Studio Inc. – San Francisco, 2024. – URL: <https://lmstudio.ai> (дата звернення: 15.02.2026).
9. AnythingLLM: The All-in-One AI Application for Everyone / Mintplex Labs Inc. – 2024. – URL: <https://anythingllm.com> (дата звернення: 15.02.2026).
10. Gemma: Open Models Based on Gemini Research and Technology / Google DeepMind. – Mountain View : Google LLC, 2024. – URL: <https://ai.google.dev/gemma> (дата звернення: 15.02.2026).
11. Gerganov G. GGUF File Format Specification / G. Gerganov // GitHub: ggml-org/ggml. – 2023. – URL: <https://github.com/ggml-org/ggml/blob/master/docs/gguf.md> (дата звернення: 06.03.2026).
12. Gao Y., Xiong Y., Gao X. [та ін.]. Retrieval-Augmented Generation for Large Language Models: A Survey // arXiv preprint. – 2023. – DOI: 10.48550/arXiv.2312.10997. – URL: <https://arxiv.org/abs/2312.10997> (дата звернення: 08.03.2026).
13. Vaswani A., Shazeer N., Parmar N. [et al.]. Attention Is All You Need // Advances in Neural Information Processing Systems. – 2017. – Vol. 30. – P. 5998–6008. – DOI: 10.48550/arXiv.1706.03762. – URL: <https://arxiv.org/abs/1706.03762> (дата звернення: 08.03.2026).
14. Brown T., Mann B., Ryder N. [та ін.]. Language Models are Few-Shot Learners // Advances in Neural Information Processing Systems. – 2020. – Vol. 33. – P. 1877–1901. – DOI: 10.48550/arXiv.2005.14165. – URL: <https://arxiv.org/abs/2005.14165> (дата звернення: 10.03.2026).

15. Regulation (EU) 2016/679 of the European Parliament and of the Council on the protection of natural persons with regard to the processing of personal data (GDPR) // Official Journal of the European Union. – 2016. – L 119. – P. 1–88. – URL: <https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32016R0679> (дата звернення: 25.02.2026).
16. Про захист персональних даних : Закон України від 01.06.2010 № 2297-VI / Верховна Рада України. – URL: <https://zakon.rada.gov.ua/laws/show/2297-17> (дата звернення: 25.02.2026).
17. Reimers N., Gurevych I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks // Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP). – Stroudsburg : ACL, 2019. – P. 3982–3992. – DOI: 10.18653/v1/D19-1410. – URL: <https://arxiv.org/abs/1908.10084> (дата звернення: 16.03.2026).
18. python-docx: Create and Modify Word Documents with Python / python-docx contributors. – 2024. – URL: <https://python-docx.readthedocs.io/en/latest> (дата звернення: 10.02.2026).
19. openpyxl: A Python Library to Read/Write Excel 2010 xlsx/xlsm Files / openpyxl contributors. – 2024. – URL: <https://openpyxl.readthedocs.io/en/stable> (дата звернення: 10.02.2026).
20. Singer-Vine J. pdfplumber: Plumb a PDF for Detailed Information About Each Char, Rectangle, Line, et cetera — and Easily Extract Text and Tables / J. Singer-Vine. – 2024. – URL: <https://github.com/jsvine/pdfplumber> (дата звернення: 11.02.2026).
21. rdflib: RDFLib is a Python Library for Working with RDF / RDFLib Team. – 2024. – URL: <https://rdflib.readthedocs.io/en/stable> (дата звернення: 12.02.2026).

- 22.RDF 1.1 Concepts and Abstract Syntax : W3C Recommendation / R. Cyganiak, D. Wood, M. Lanthaler. – Cambridge : World Wide Web Consortium, 2014. – URL: <https://www.w3.org/TR/rdf11-concepts> (дата звернення: 12.02.2026).
- 23.LanceDB: A Developer-Friendly, Serverless Vector Database for AI Applications / LanceDB Inc. – 2024. – URL: <https://lancedb.github.io/lancedb> (дата звернення: 05.03.2025).
- 24.Андрощук А. Г., Малюга О. С. Використання штучного інтелекту у вищій освіті: стан і тенденції // International Science Journal of Education and Linguistics. – 2024. – Vol. 3, No. 2. – P. 27–35. – DOI: 10.46299/j.isjel.20240302.04.
- 25.Стратегія розвитку штучного інтелекту в Україні : монографія / А. І. Шевченко, С. О. Барановський, О. В. Білокобильський [та ін.] ; за заг. ред. А. І. Шевченка. – Київ : ІАІП НАН України, 2023. – 305 с. – URL: https://jai.in.ua/archive/2023/ai_mono.pdf (дата звернення: 26.02.2026).
- 26.Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Reading : Addison-Wesley Professional, 1994. – 417 p. – ISBN 0-201-63361-2.