МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

"До захисту допущений"
Зав. кафедри комп'ютерних наук та прикладної математики
д.т.н., професор Турбал Ю. В.
«___» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

«Розробка інтерактивного цифрового асистента для аналізу і пошуку даних у документах»

Виконав: Стадник Сергій Вячеславович
Студент навчально-наукового інституту кібернетики, інформаційних технологій та інженерії
група ІПЗ-41і-2

_____
(підпис)

Керівник: доц., к.т.н. Жуковський Віктор Володимирович

_____
(підпис)

Рівне – 2025

ЗМІСТ

РЕФЕРАТ

Кваліфікаційна робота присвячена дослідженню сучасних підходів до створення цифрових асистентів, що дозволяють здійснювати аналіз і пошук інформації в текстових документах на основі великих мовних моделей (LLM). Метою роботи є дослідження архітектурного підходу Retrieval-Augmented Generation (RAG) та аналіз програмних рішень, які реалізують зазначену концепцію, з метою перспективного впровадження подібної системи в освітньому середовищі закладу вищої освіти.

У роботі розглянуто принципи функціонування систем цифрових асистентів, що поєднують векторний пошук з генерацією відповідей на природній мові. Проведено огляд і тестування двох проєктів відкритого коду — RAGFlow та Kotaemon, які реалізують векторний пошук з підтримкою документів у форматах PDF, DOCX тощо. Окрему увагу приділено перспективам використання платформи Dify як інтегрованого фреймворку для створення LLM-застосунків.

Практична частина роботи присвячена перевірці працездатності систем із завантаженням навчально-нормативної документації університету та аналізом результатів відповідей. Виявлено низку проблем, пов'язаних з обробкою вхідних даних (наприклад, неправильне розбиття на чанки, злиття слів), що ускладнюють повноцінну реалізацію асистента на цьому етапі.

Отримані результати дають змогу сформулювати технічні та концептуальні вимоги до майбутньої системи цифрового асистента для студентів, абітурієнтів та викладачів, яка дозволить забезпечити швидкий і зручний доступ до внутрішньої документації університету.

Ключові слова: цифровий асистент, LLM, RAG, векторна база, LangChain, документальний пошук, PDF, Kotaemon, RAGFlow, Dify.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI — Artificial Intelligence (штучний інтелект)

API — Application Programming Interface (інтерфейс прикладного програмування)

LLM — Large Language Model (велика мовна модель)

PDF — Portable Document Format (формат електронного документа)

OCR — Optical Character Recognition (оптичне розпізнавання тексту)

UI — User Interface (користувацький інтерфейс)

UX — User Experience (досвід користувача)

QA — Question Answering (відповіді на запитання)

RAG — Retrieval-Augmented Generation (генерація, посилена пошуком)

JSON — JavaScript Object Notation (формат обміну даними)

DB — Database (база даних)

API key — унікальний ідентифікатор для доступу до API

Token — токен авторизації для доступу до сервісів

Chunk — фрагмент тексту, отриманий під час обробки документа

Embedding — представлення тексту у вигляді числового вектору для пошуку та класифікації

Frontend — клієнтська частина програмного забезпечення

Backend — серверна частина програмного забезпечення

Docker — платформа для розгортання програм у контейнерах

Git — система керування версіями

POST / GET — методи HTTP-запитів для відправлення/отримання даних

ВСТУП

У сучасному інформаційному суспільстві значний обсяг знань зберігається у вигляді текстових документів, які потребують ефективного пошуку, аналізу та узагальнення. Це стосується і сфери освіти, зокрема вищих навчальних закладів, де існує великий масив нормативних, методичних, організаційних документів, що регламентують навчальний процес. Водночас отримання доступу до релевантної інформації залишається складним завданням як для студентів, так і для викладачів та вступників. У цьому контексті актуальним є створення цифрових асистентів — програмних систем, які здатні забезпечити інтелектуальний пошук даних у текстових масивах документів на запити природною мовою.

Особливу увагу привертають сучасні розробки на основі великих мовних моделей (Large Language Models, LLM), які здатні генерувати зв'язні та інформативні відповіді, використовуючи механізм доповнення зовнішнім пошуком (Retrieval-Augmented Generation, RAG). Цей підхід поєднує генеративні можливості моделей з можливістю звернення до власної бази знань — зокрема, векторизованого корпусу документів. Таким чином, з'являється можливість реалізувати асистента, який не лише "розуміє" запит, але й формує відповідь на основі реального, перевіреного контексту.

Метою цієї кваліфікаційної роботи є дослідження існуючих підходів до побудови цифрових асистентів з функціями семантичного пошуку в документах, аналіз відкритих платформ **Kotaemon**, **RAGFlow**, а також **Dify** — з подальшим формуванням технічного підґрунтя для розробки подібної системи в межах університетського середовища.

Об'єктом дослідження є процеси пошуку та аналізу текстової інформації в освітніх документах. Предметом дослідження є методи реалізації інтерактивних цифрових асистентів з використанням LLM, векторного пошуку та систем типу RAG.

У процесі роботи було проаналізовано архітектурні принципи сучасних open-source систем, що реалізують концепцію доповненого пошуку. Практична частина роботи полягала у перевірці працездатності двох систем (RAGFlow і

Kotaemon) на реальних даних університету з виявленням проблем, пов'язаних з попередньою обробкою документів (chunking, злиття слів, втрата контексту тощо), які заважають повноцінному впровадженню системи на поточному етапі. Отримані результати дають змогу зробити висновки щодо технічних і функціональних вимог до перспективного рішення, а також окреслити напрямки його розвитку.

Методами дослідження в роботі є: аналіз архітектурних підходів і документації open-source проєктів, експериментальна перевірка роботи систем на тестовому корпусі документів, логічне узагальнення результатів і побудова технічного бачення майбутньої системи. Застосовано інструменти LangChain, FastAPI, Docker, Python, Postgres, а також середовище тестування API-запитів.

Практична значущість роботи полягає в можливості адаптації аналізованих рішень до реальних потреб університету. Запропонований підхід дозволяє сформувати основу для впровадження цифрового асистента, який надає відповіді на запити студентів, абітурієнтів або викладачів щодо навчальних планів, правил прийому, регламентів, положень тощо. Це значно скорочує час доступу до важливої інформації та знижує навантаження на адміністративний персонал.

Структура роботи: дипломна робота складається з трьох розділів. У першому розділі описано теоретичні основи побудови систем пошуку на базі LLM. У другому розділі проведено аналіз готових рішень (Kotaemon, RAGFlow, Dify) та визначено технічні обмеження. Третій розділ присвячено практичній спробі побудови асистента з використанням API та open-source рішень. Робота завершується висновками, списком джерел та додатками.

I. ТЕОРЕТИЧНІ ЗАСАДИ ПОБУДОВИ ЦИФРОВИХ АСИСТЕНТІВ НА ОСНОВІ LLM

1.1 Цифрові асистенти як інструмент роботи з неструктурованими документами

У сучасному цифровому середовищі більшість інформації, що створюється і зберігається, представлена у вигляді неструктурованих даних: текстових файлів, сканованих документів, електронної пошти, публікацій на веб-ресурсах тощо. За оцінками дослідницьких організацій, понад 80 % усіх даних у світі не мають чіткої структури і не піддаються обробці традиційними методами аналізу баз даних. Це створює значні труднощі в доступі до інформації, її пошуку та повторному використанні, особливо у сфері освіти, де велика кількість документів регламентного, навчального та організаційного характеру зберігається у вигляді PDF- або DOCX-файлів без ефективної системи пошуку за змістом.

Одним із перспективних напрямів розв'язання цієї проблеми є застосування інтерактивних цифрових асистентів — програмних систем, що здатні інтерпретувати природномовні запити користувача, здійснювати пошук по наявному корпусу документів і формувати змістовні відповіді. Такі асистенти дедалі частіше реалізуються на основі великих мовних моделей (LLM, Large Language Models), які забезпечують високий рівень розуміння контексту, підтримку діалогової взаємодії, а також генерацію відповідей на природній мові.

Цифровий асистент виконує функцію посередника між користувачем і неструктурованими даними. Його основними завданнями є: прийняття запиту у вільній формі, визначення ключових понять, пошук релевантної інформації у внутрішній базі знань (наприклад, у векторному сховищі текстових embedding-представлень), формування відповіді з посиланням на джерело або цитатою з документа. Такий підхід значно зменшує навантаження на персонал, підвищує доступність документів і усуває потребу в ручному перегляді великої кількості файлів.

У контексті закладів вищої освіти застосування цифрових асистентів особливо актуальне. Студенти, викладачі та абітурієнти регулярно звертаються

до нормативних документів, таких як правила прийому, освітні програми, положення про організацію навчального процесу, регламенти з атестації, практики чи дипломного проєктування. У більшості випадків пошук потрібної інформації здійснюється вручну, що є неефективним. Використання цифрового асистента дозволяє значно пришвидшити цей процес та зробити взаємодію з документами інтуїтивно зрозумілою.

Раніше подібні системи реалізовувалися у вигляді статичних чат-ботів з обмеженою кількістю шаблонних відповідей або ж через ключові слова. Сучасні ж рішення, які базуються на LLM, дозволяють працювати з широким спектром запитів, не вимагаючи від користувача попередньої підготовки або знання структури документів. Це забезпечується завдяки використанню технологій семантичного пошуку, embedding-моделей і механізмів доповнення генерацією на основі зовнішнього контексту (RAG — Retrieval-Augmented Generation).

Отже, цифрові асистенти нового покоління відкривають нові можливості для ефективної взаємодії з інформацією. Вони стають важливим інструментом у сфері освіти, дозволяючи покращити доступ до внутрішніх ресурсів, зменшити обсяг рутинної роботи та забезпечити прозорість інформаційних процесів в університетському середовищі.

1.2 Принцип роботи систем з Retrieval-Augmented Generation

Одним із ключових технологічних підходів, який дозволяє поєднати можливості великих мовних моделей (LLM) зі знаннями, збереженими у зовнішніх джерелах, є Retrieval-Augmented Generation (RAG). Цей підхід передбачає розділення процесу генерації відповіді на два етапи: спочатку виконується пошук релевантної інформації, а потім — побудова відповіді з урахуванням знайденого контексту. Таким чином система залишається узгодженою з перевіреними джерелами і водночас здатна формулювати природномовні відповіді.

Основна причина використання RAG полягає в обмеженні LLM щодо обсягу "вбудованих" знань. Навіть найбільш потужні моделі, такі як GPT-4, не мають доступу до актуального або приватного контенту користувача. RAG дозволяє компенсувати це шляхом підключення зовнішньої бази знань — наприклад, документів університету — і динамічного пошуку по ній перед формуванням відповіді.

Стандартна архітектура RAG-системи включає кілька компонентів:

1. Векторизація документів

Перед використанням всі документи попередньо обробляються: їхній текст розбивається на логічні частини (чанки), які потім перетворюються у векторні представлення (embedding) за допомогою спеціалізованої моделі. Ці вектори зберігаються у векторній базі даних (наприклад, FAISS, Chroma, Weaviate, Qdrant) з можливістю пошуку за схожістю.

2. Обробка запиту користувача

Коли користувач надсилає запит у вигляді природного тексту, система також перетворює його в embedding-вектор і виконує пошук найбільш релевантних документних фрагментів у векторній базі. Це забезпечує семантичний пошук — тобто пошук за змістом, а не за ключовими словами.

3. Формування відповіді

Знайдені фрагменти документації передаються як контекст до LLM, яка на їх основі генерує відповідь. Таким чином, модель не вигадує

інформацію, а використовує наданий матеріал. У відповідь також можуть включатися посилання на джерела, дати, цитати з документів тощо.

Важливо, що структура і якість попередньої обробки документів (розбиття на чанки, точність розпізнавання тексту з PDF чи сканів) значно впливає на здатність системи надати коректну відповідь. При поганому розбитті фрази можуть бути неповними, текст — злитим, що призводить до хибного результату пошуку або неповного контексту.

Перевагами підходу RAG є:

- підвищена точність відповідей завдяки прив'язці до фактичних документів
- можливість працювати з приватними та закритими наборами даних
- гнучкість в оновленні знань без перенавчання моделі

Недоліки включають:

- залежність від якості попередньої обробки документів
- потребу в побудові і підтримці векторної бази
- збільшення затримки відповіді через додаткові етапи пошуку

Таким чином, підхід Retrieval-Augmented Generation дозволяє поєднати потужність великих мовних моделей з точністю пошуку у власних текстових даних. У контексті університетської інформаційної системи RAG-архітектура є одним із найбільш ефективних способів реалізації асистента, здатного відповідати на запити користувачів з опорою на нормативні документи навчального закладу.

1.3 Векторні бази знань: створення, збереження, доступ

Векторна база знань є ключовим компонентом у системах семантичного пошуку та Retrieval-Augmented Generation. Її основна функція полягає у збереженні текстових фрагментів у вигляді векторних представлень, що дає змогу знаходити інформацію за змістом, а не за ключовими словами. Це суттєво покращує точність і гнучкість відповіді системи на природномовні запити.

Першим етапом побудови такої бази є попередня обробка тексту. Документи очищуються від службових символів, зайвих розривів, таблиць і форматувань. Особливу складність становлять PDF-файли, у яких можуть зникати пробіли, зміщуватись абзаци або порушуватись порядок сторінок. Для цього застосовуються спеціалізовані засоби обробки, які намагаються відновити логічну структуру документа.

Далі здійснюється розбиття тексту на фрагменти, які називають чанками. Це дає змогу працювати з текстом блоками фіксованої довжини, що важливо для обмежень на обсяг вхідних даних мовних моделей. Зазвичай використовуються чанки обсягом 200–500 слів або до 1000 токенів. Для збереження контексту сусідніх фрагментів застосовується часткове перекриття між чанками, що запобігає втраті сенсу на межах поділу.

Кожен такий чанк передається у векторизаційну модель (embedding model), яка формує числовий вектор, що відображає його зміст. Серед поширених моделей для цього застосовують як відкриті (наприклад, all-MiniLM-L6-v2, bge-small-en), так і комерційні (OpenAI, Cohere, Google). Вибір моделі має вирішальне значення для точності подальшого пошуку: неякісні вектори призводять до нерелевантних результатів навіть при формально правильному запиті.

Отримані вектори разом із прив’язаним до них текстом зберігаються у векторній базі даних. Найчастіше використовують FAISS, Chroma, Qdrant або Weaviate — ці системи дозволяють ефективно здійснювати пошук за відстанню векторів, швидко знаходячи найближчі до заданого запиту. Бази можуть масштабуватись, підтримувати мільйони векторів і бути розгорнуті локально чи в хмарі.

Під час запиту користувача система перетворює текстовий запит у вектор і здійснює пошук у базі, знаходячи найближчі фрагменти. Зазвичай вибирається кілька найбільш релевантних блоків, які передаються до мовної моделі як контекст для генерації відповіді. Важливо, щоб разом із текстом зберігались метадані — наприклад, назва документа, номер сторінки, URL-джерело. Це дозволяє забезпечити прозорість і верифікованість відповіді.

Якість побудови векторної бази визначає здатність цифрового асистента надавати точні, коректні й пояснювані відповіді. Помилки на етапі розбиття, токенізації або формування векторів можуть призводити до зниження якості пошуку, втрати частини даних або генерації недостовірних результатів.

У системах, призначених для роботи з нормативними документами в університетському середовищі, векторна база знань дозволяє формувати єдине семантичне середовище для пошуку, яке адаптоване до запитів користувачів різного рівня — від абітурієнта до викладача чи адміністрації. Це забезпечує швидкий доступ до необхідної інформації без потреби вручну переглядати документи.

1.4 Embedding моделей: роль токенізації, chunking, індексації

Embedding-модель є центральним компонентом сучасних систем семантичного пошуку. Її завдання полягає у перетворенні тексту в вектор — набір числових значень, що відображає змістовне наповнення текстового фрагмента. Саме ці вектори використовуються для подальшого пошуку за схожістю, тому якість embedding-моделі безпосередньо впливає на точність і релевантність знайдених результатів.

На етапі embedding першим кроком є токенізація — поділ тексту на найменші оброблювані одиниці. Різні моделі використовують власні токенізатори: одні орієнтуються на окремі слова, інші — на склади, морфеми або навіть байти. Обрана стратегія впливає на довжину вхідних послідовностей, які модель здатна обробити, та на її чутливість до граматичних або пунктуаційних змін.

Після токенізації текстовий чанк проходить через embedding-модель, яка навчається так, щоб подібні за змістом фрагменти мали близькі вектори. Для цього моделі зазвичай тренуються на великих корпусах текстів із застосуванням контрастивного або багатовимірного навчання. В результаті кожен чанк отримує свій вектор — числове представлення у просторі розмірності 384, 768 або 1024 вимірів (залежно від моделі).

Особливу роль у формуванні embedding відіграє якість chunking — розбиття тексту на частини. Якщо розбиття здійснене невдало, фрагмент може вийти надто коротким і втратити зміст або навпаки — бути надто довгим і не вміститись у допустимий контекст моделі. В обох випадках вектор не відображатиме суті, а подальший пошук стане неточним. Ідеальний чанк зберігає семантичну завершеність: абзац, пункт або логічний блок з одного документа.

Після отримання векторів важливо забезпечити їх правильну індексацію. Під цим мається на увазі не лише збереження самого вектора у базі, а й пов’язування його з метаданими: звідки він отриманий, до якого документа чи розділу належить, який був текстовий чанк на вхід. Це дозволяє не лише знаходити релевантні фрагменти, а й повертати повноцінні відповіді користувачеві з посиланням на джерело.

Embedding-моделі бувають універсальними або спеціалізованими. Універсальні (наприклад, all-MiniLM-L6-v2) добре підходять для загальних запитів, але можуть втрачати точність у вузькоспеціалізованих галузях, як-от медицина, право чи освіта. Тому у випадку створення цифрового асистента для роботи з документами університету доцільно використовувати або адаптовану модель, або провести тонке донавчання на локальному корпусі даних.

Загалом embedding-модель виконує роль “перекладача” з мови тексту на мову математики — вона дозволяє зіставляти семантично подібні об’єкти без прямого збігу слів. Саме завдяки цьому цифровий асистент може відповідати на запит “Що потрібно для складання академрізниці?” навіть якщо в документі ця інформація міститься під іншим формулюванням.

1.5 Інструменти реалізації: LangChain, LlamaIndex, Haystack, Dify

Сучасні фреймворки для побудови систем на основі великих мовних моделей дозволяють значно спростити реалізацію асистентів з підтримкою пошуку в документах, інтеграцією LLM, побудовою логіки запитів і роботи з джерелами даних. Серед найпопулярніших і найфункціональніших рішень виділяють LangChain, LlamaIndex, Haystack та Dify. Кожен з них має свої переваги й підходить для різних сценаріїв використання.

LangChain є одним із найвідоміших open-source фреймворків, що дозволяє будувати ланцюги обробки запитів для LLM. Він підтримує інтеграцію з різними джерелами даних, мовними моделями (OpenAI, HuggingFace, Cohere) та векторними базами (Chroma, FAISS, Pinecone). Основною перевагою LangChain є гнучкість — користувач сам визначає порядок і логіку обробки, обирає методи chunking, способи виклику моделей, механізми повернення відповідей. Недоліком є складність налаштування при створенні системи з нуля: для базового використання потрібне розуміння структури компонентів, векторного пошуку, prompt engineering та API взаємодії.

LlamaIndex (раніше відомий як GPT Index) спеціалізується на організації структурованих індексів даних для LLM. Цей інструмент дозволяє створювати ієрархічні, словникові або векторні індекси на основі великих обсягів документів. Він має зручний API та абстракції для роботи з різними форматами файлів, особливо добре підходить для інтеграції зі сховищами документів. Його перевагою є автоматизація багатьох процесів: користувачеві не потрібно самостійно налаштовувати базу або процес пошуку — фреймворк бере це на себе. Проте він менш гнучкий у сценаріях, де потрібен повний контроль над логікою асистента.

Haystack — це більш комплексне рішення з власним сервером, панеллю керування, підтримкою LLM та класичними ML-моделями. Цей фреймворк орієнтований на промислові рішення й має вбудовані компоненти для обробки запитів, генерації відповідей, ранжування результатів. Він підтримує розгортання в Docker-контейнерах, масштабування, інтеграцію з Elasticsearch

або OpenSearch. Перевагою є висока стабільність і наявність UI-інструментів, але робота з Haystack вимагає більше ресурсів і підготовки.

Dify — відносно нова, але швидко популяризована платформа, яка надає графічний інтерфейс для створення LLM-застосунків. Вона дозволяє будувати боти, API, аналітичні інтерфейси, підключати бази документів і задавати ролі для асистентів. Особливістю Dify є спрощений підхід: більшість налаштувань здійснюється через UI, а запити і відповіді можна тестувати в реальному часі. Це робить платформу зручною для швидкого прототипування, особливо в умовах обмеженого часу або відсутності досвіду роботи з програмним кодом. Недоліком є залежність від зовнішніх сервісів і обмеження кастомізації у безкоштовній версії.

У контексті створення цифрового асистента для роботи з документами університету кожен із зазначених інструментів може відігравати свою роль. LangChain надає контроль і гнучкість, LlamaIndex — ефективність побудови індексів, Dify — простоту реалізації MVP-прототипу, а Haystack — стабільність у великих системах. Практична частина цієї роботи присвячена оцінці ефективності використання деяких із цих інструментів на прикладі реальних документів.

1.6 Проблеми обробки PDF та форматованих документів (DOCX, сканкопії)

Обробка PDF-файлів часто виявляється нетривіальним завданням, навіть якщо документ містить текст у цифровому форматі. Шукачі стикаються з численними складнощами: багатоколонні макети, фонова графіка, водяні знаки, заголовки/підвали та таблиці, що виходять за межі сторінки. Все це потребує створення складних евристик для аналізу структури документа перед отриманням тексту .

Скановані документи у вигляді зображень — це окрема складність. Щоб перетворити їх на текст, необхідно застосовувати OCR. Цей процес є ресурсоємним і не завжди точним: спотворення тексту, втрата табличної структури, нерозпізнані символи — усе це погіршує якість вихідного матеріалу .

Ключовою проблемою є поганий розподіл тексту після OCR — неправильна токенізація, злиття слів або розриви фраз, що призводить до непридатності чанків для семантичного пошуку. Навіть добре оброблені PDF-фрагменти можуть містити «шум»: заголовки, номери сторінок, клікабельні поля форми, тло, які треба відфільтрувати

PDF-документи також зберігають текст у порядку, що відрізняється від його візуального розташування. Це означає, що послідовний витяг може порушувати природну структуру речень і абзаців .

Особливі виклики додають випадки поєднання селектного тексту та зображень із текстом, багатоколонні макети та складно структуровані таблиці . Ці проблеми зберігають недостовірність вихідних чанків, що призводить до втрати точності пошуку.

Є методи для подолання викликів: контекстно обізнані токенізатори, інструменти як PyMuPDF4LLM, складний OCR із передобробкою (вирівнювання, binarization, despeckling). Однак вони не дають гарантій без значних затрат часу та обробних ресурсів.

При масштабному завантаженні сотень або тисяч PDF проблеми загострюються: швидкість обробки знижується, інфраструктура навантажується, ресурсів OCR стає недостатньо — а ручна корекція стає непрактичною .

У контексті університетського цифрового асистента, ці проблеми означають, що без достатньо надійної пайплайнової обробки документів система не дає коректних і пояснюваних результатів. Навіть перевірені open-source рішення (RAGFlow, Kotaemon) виявляються вразливими до даних помилок, що визначило зону фокусування експериментів цієї роботи.

II. АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ ТА ПОСТАНОВКА ЗАДАЧІ

2.1 Характеристика проєкту Kotaemon: архітектура, інтерфейс, обмеження

Kotaemon — це open-source платформа, розроблена для побудови локального асистента з можливістю завантаження власних документів і генерації відповідей на їх основі за допомогою LLM. Проєкт реалізовано на базі фреймворку LangChain, із підтримкою векторної бази Chroma та API-інтерфейсом на FastAPI. Основна ідея полягає у створенні самодостатнього застосунку, який не потребує складного розгортання та дозволяє працювати з власними PDF-файлами в локальному середовищі.

Після встановлення і запуску Kotaemon через Docker-контейнери, користувач отримує веб-інтерфейс, у якому можна додавати документи, переглядати їхню обробку та здійснювати пошукові запити. Документи додаються у вигляді PDF-файлів, після чого система виконує їх обробку: текст витягується, розбивається на чанки, embedding-фрагменти записуються до векторної бази.

Система підтримує інтеграцію з кількома типами LLM: як локальними, так і зовнішніми (через API OpenAI або інші сервіси). Це дозволяє адаптувати Kotaemon під доступні ресурси та вимоги безпеки.

У процесі тестування в рамках цієї роботи було встановлено декілька ключових обмежень. По-перше, виявлені проблеми з якістю обробки тексту з PDF-документів. Часто спостерігалося злиття слів при переході між рядками, що спричиняло генерацію некоректних embedding-представлень. У прикладеному тестовому випадку слова наприкінці одного рядка зливалися з першими словами наступного рядка без пробілів, що порушувало семантику тексту та унеможливлювало якісний пошук.

По-друге, Kotaemon не має вбудованих інструментів для перевірки точності розбиття на чанки або візуального перегляду тексту до та після обробки. Це ускладнює виявлення та виправлення помилок у розпізнаванні або парсингу, особливо у випадках складної структури документа.

Ще однією проблемою стало те, що попри правильну відповідь від LLM у логах системи, у веб-інтерфейсі користувачеві відображалася фраза “На жаль, я не зміг знайти інформацію у наявних документах”, що вказувало на розсинхронізацію між API-відповіддю і логікою відображення у фронтенді. Це значно знижувало довіру до системи та унеможливлювало використання без глибокого технічного втручання.

Варто також зазначити, що Kotaemon не підтримує формати DOCX або скановані документи без сторонньої конверсії, а також не має зручної адміністративної панелі для керування документами. Попри це, його відносна простота, готовність до локального запуску, використання сучасних підходів до embedding-пошуку та відкритий код роблять його зручним стартовим майданчиком для реалізації концепції цифрового асистента.

У рамках цієї роботи Kotaemon розглядався не як кінцеве рішення, а як експериментальний майданчик для перевірки базових можливостей асистента: індексації документів, генерації відповідей на запити, перевірки якості chunking та роботи embedding-моделі на університетському корпусі документів.

2.2 Характеристика проєкту RAGFlow: функціональність, парсинг документів, API

RAGFlow — це open-source проєкт, побудований за принципами Retrieval-Augmented Generation, призначений для реалізації систем інтелектуального пошуку в документах за допомогою великих мовних моделей. Його основна ідея — забезпечити модульну платформу для завантаження текстових даних, побудови векторної бази знань, а також генерації відповідей на природномовні запити користувача.

Проєкт базується на використанні LangChain як фреймворку для побудови логіки запитів та взаємодії з LLM, ChromaDB як векторної бази даних, а також FastAPI для реалізації зовнішнього REST API. Інтерфейс управління, побудований на Streamlit або Gradio, дозволяє керувати документами, запускати парсинг і надсилати запити до системи у зручній формі.

Після встановлення RAGFlow через Docker-контейнери або локальний запуск, користувач може завантажувати документи (переважно у форматі PDF), які система автоматично обробляє: витягує текст, розбиває його на чанки, векторизує та зберігає у базі. Підтримується інтеграція з відкритими моделями від HuggingFace, а також з API OpenAI, що дозволяє налаштувати якість відповіді відповідно до можливостей системи.

У ході тестування, проведеного в рамках цієї роботи, виявлено як позитивні, так і обмежувальні сторони платформи. Зокрема, RAGFlow забезпечує зручний механізм завантаження великої кількості документів, дозволяє використовувати різні embedding-моделі (у тому числі локальні), підтримує оновлення бази даних без перезапуску системи.

Проте однією з основних проблем стала якість розпізнавання тексту з PDF-файлів. У ряді випадків були зафіксовані злиття слів, порушення форматування, неправильне визначення меж абзаців — усе це негативно впливало на якість embedding-представлень. Чанки формувалися некоректно: речення могли обриватися на півслові або починатися з другої частини попереднього абзацу. Це ускладнювало генерацію відповідей та призводило до втрати контексту.

Ще однією проблемою виявилася відсутність інструментів для перевірки та редагування отриманих чанків. Користувач не має змоги переглянути, як саме розбитий текст перед тим, як він потрапить до векторної бази. У разі виявлення помилки єдиним варіантом є повне перезавантаження документу з попередньою ручною правкою.

Система надає зручне API, через яке можна надсилати запити із зовнішніх застосунків. Це дозволяє інтегрувати RAGFlow як бекенд для чат-ботів, вебінтерфейсів або мобільних застосунків. У тестових прикладах було перевірено роботу API для надсилання запитів з текстом і отримання відповіді, що дозволяє вбудувати систему у власну архітектуру цифрового асистента.

Попри деякі технічні обмеження, RAGFlow виявився гнучким та налаштовуваним інструментом для побудови локальної системи семантичного пошуку. Він добре підходить для експериментального використання, апробації embedding-моделей, перевірки роботи API та вивчення практичних аспектів побудови RAG-систем. У рамках цієї роботи RAGFlow був використаний для перевірки базової працездатності концепції цифрового асистента на корпусі реальних університетських документів.

2.3 Аналіз платформи Dify як перспективного рішення

Платформа Dify є одним із найбільш комплексних open-source рішень для реалізації цифрових асистентів на базі LLM. Її архітектура охоплює такі ключові компоненти, як чат-інтерфейс, модуль завантаження та обробки документів, підтримку різних LLM (у тому числі через API OpenAI, Anthropic, Azure) та векторну базу знань.

На етапі попереднього тестування виявлялись значні проблеми з відображенням відповідей у чаті. Незважаючи на те, що система коректно знаходила релевантний контекст і LLM формував відповідь, вона не потрапляла до інтерфейсу користувача. Аналіз логів показав наявність помилки типу TypeError: can only concatenate str (not "list") to str, що блокувала вивід результату. Це створювало хибне враження про відсутність відповіді.

Однак після глибшого ознайомлення з архітектурою Dify було протестовано альтернативну модель використання — **Knowledge Retrieval + Chatbot**, у якій система отримує запит, витягує релевантні чанки та формує відповідь із посиланням на джерело. Саме ця конфігурація дозволила усунути помилку й забезпечити повноцінну відповідь у чаті.

У тестових запитах, що стосувалися положень університету (академічна заборгованість, переведення на держзамовлення, політика доброчесності), Dify продемонстрував стабільний результат, надаючи не лише узагальнені відповіді, а й точні фрази з документів. Також лог-файли системи містили інформацію про використані фрагменти (top-k), embedding-підсистему, ID джерел та метадані документа.

Таким чином, попри наявні початкові проблеми, Dify після адаптації та правильного конфігурування підтвердив свою перспективність. Його перевагами є гнучка підтримка API, зручний чат-інтерфейс, вбудоване логування, налаштовувана база знань та відкритий код, який дозволяє адаптувати систему під потреби конкретного навчального закладу.

2.4 Порівняльний аналіз: якість chunking, відповідей, робота з API

На основі експериментального дослідження трьох систем — Kotaemon, RAGFlow та Dify — було здійснено порівняльний аналіз за трьома критеріями: якість розбиття документів (chunking), точність сформованих відповідей та гнучкість роботи з API/інтерфейсом користувача.

У Kotaemon спостерігалась проблема із відображенням відповідей у чаті, попри наявність правильно сформованої відповіді в логах. Якість chunking була помірною: система підтримувала базову обробку PDF-документів, однак виникали злиття слів та порушення структури абзаців. Відповіді на запити часто були стандартними («Інформація відсутня»), навіть якщо відповідний фрагмент був у базі знань.

RAGFlow показав стабільну архітектуру, однак відповіді були менш точними. У тестах викладачі вказали, що система не надавала очікуваної конкретики. Проблеми з chunking були схожі на Kotaemon: злиття фраз, неповні речення. API RAGFlow досить просте, однак не забезпечує зручного інтерфейсу для користувача.

Dify, після переходу до режиму Knowledge Retrieval + Chatbot, продемонстрував найбільш узгоджені результати. Відповіді містили точні цитати, система показувала ID джерела, top-k documents, та посилання на фрагмент документа. Інтерфейс дозволяв інтерактивну взаємодію, а лог-файли фіксували не лише відповідь, але й шлях її формування. Якість chunking залежала від мови документа, але загалом була вищою, ніж у конкурентів.

Таким чином, Dify після усунення початкової помилки виявився найстабільнішим з-поміж трьох рішень, хоча й вимагає ручного налаштування моделей, embedding та джерел даних.

2.5 Порівняльна характеристика сильних і слабких сторін систем

У процесі аналізу трьох рішень – **Kotaemon**, **RAGFlow** та **Dify** – було виявлено як спільні, так і унікальні характеристики кожної системи, що впливають на придатність їх до використання в освітньому середовищі.

Kotaemon показав достатньо стабільну роботу з PDF-документами, але мав істотні обмеження в логіці обробки відповідей. Навіть якщо LLM формувала правильну відповідь, вона не завжди виводилась в інтерфейсі. Система не містила розвиненої візуалізації логів, а chunking базувався на простому поділі без глибокого врахування змісту. Проблеми з українською мовою та розділовими символами знижували якість витягу інформації. Водночас проєкт є активним, має модульну структуру й перспективу кастомізації.

RAGFlow вирізняється добре структурованою архітектурою, однак не продемонстрував високої якості відповідей під час тестування. Система часто формувала узагальнені або надто загальні відповіді, які не відповідали змісту нормативних документів. Проблеми з токенізацією призводили до злиття слів та некоректного поділу речень. Серед переваг – простота запуску та логічне API, але відсутність чіткої візуалізації або розмітки джерела відповіді залишалась критичним недоліком.

Dify спочатку також мала суттєві проблеми — відповідь не з’являлась в інтерфейсі, хоча LLM її генерувала. Після тестування та переходу до режиму *Knowledge Retrieval + Chatbot*, система стабілізувалась. Відповіді стали точними, з прямими цитатами та посиланням на джерело. Інтерфейс дозволяє керувати документами, бачити історію запитів і зберігати логи (у т.ч. top-k документи, вектори, ID фрагментів). Якість chunking вища за інші системи, з меншими помилками форматування. Dify підтримує гнучку роботу з LLM API та має зручну панель адміністрування.

Таким чином, з трьох досліджуваних систем саме Dify після налаштування продемонстрував найбільший потенціал для інтеграції у цифрового асистента університету. Його відкритий код, підтримка різних LLM і зрозумілий вебінтерфейс дають змогу адаптувати платформу до локальних потреб, зберігаючи при цьому масштабованість.

2.6 Узагальнення проблем інтеграції LLM у локальні рішення

Розгортання систем на базі великих мовних моделей у локальному або внутрішньому середовищі організації, зокрема закладу вищої освіти, супроводжується низкою технічних і організаційних труднощів, які мають бути враховані під час розробки цифрового асистента.

Першою суттєвою проблемою є якість вхідних даних. Усі протестовані рішення мають залежність від якості тексту, витягнутого з PDF-документів. Навіть незначні дефекти в структурі документа (наприклад, багатоколонне форматування, наявність колонтитулів, некоректні розриви рядків) можуть призвести до порушень токенізації, злиття слів і втрати логіки викладу. Це, своєю чергою, погіршує якість embedding-представлень і відповідей моделі.

Другим поширеним викликом є обмеженість або відсутність контролю над процесом chunking у готових платформах. У більшості випадків цей процес автоматизовано, але не прозоро — користувач не має змоги переглянути чи скорегувати сформовані фрагменти тексту до збереження у векторній базі. У результаті навіть технічно коректна відповідь LLM може ґрунтуватися на неповному або перекрученому контексті.

Ще одним проблемним аспектом є нестабільна обробка відповідей у частині інтерфейсу користувача. Наприклад, у Kotaemon були зафіксовані ситуації, коли модель формувала правильну відповідь, однак інтерфейс не відображав її або повідомляв про відсутність інформації. Подібні розсинхронізації знижують довіру до системи й вимагають ручного втручання у фронтенд або API.

Також варто враховувати технічні обмеження, пов’язані з продуктивністю та ресурсами. Для розгортання навіть базової системи з підтримкою векторного пошуку необхідна наявність серверного середовища, бази даних, контейнеризації (Docker), а також зовнішнього або локального доступу до мовної моделі. Це потребує технічної кваліфікації від розробника і доступу до інфраструктури, що може бути складним в умовах університету.

Окремо слід відзначити проблеми безпеки та конфіденційності. У разі використання сторонніх API для генерації embedding або відповідей (наприклад,

OpenAI чи Cohere), виникає ризик передачі частини внутрішніх даних за межі локальної мережі. Це обмежує можливість обробки конфіденційних або регламентованих документів без належної локальної інфраструктури.

І нарешті, інтеграція таких систем у реальне освітнє середовище вимагає адаптації до локального контексту: термінології, стилістики документів, часто — української мови. Більшість відкритих рішень спочатку орієнтовані на англомовні запити й документи, що вимагає додаткового налаштування або тонкого донавчання моделей.

Таким чином, інтеграція систем на базі LLM у локальне освітнє середовище пов’язана з низкою проблем: від якості вихідних документів і обмежень на попередню обробку, до технічних нюансів розгортання та питань безпеки. Ці проблеми повинні бути враховані під час проєктування та реалізації цифрового асистента, і саме на їх подолання має бути спрямована подальша робота.

2.7 Визначення задачі дослідження: якість відповіді, робота з масивом документів, стабільність

Аналіз наявних open-source рішень для реалізації цифрових асистентів на основі великих мовних моделей показав, що жодна з розглянутих систем не є безумовно придатною для негайного впровадження в університетське середовище без адаптації. Разом з тим, наявні напрацювання, зокрема платформи Dify, RAGFlow та Kotaemon, можуть слугувати надійним підґрунтям для подальшої розробки і дослідження. На основі виявлених можливостей і обмежень було сформульовано низку задач, вирішення яких є основою для реалізації концепції цифрового асистента.

Першою ключовою задачею є забезпечення високої якості відповіді**.** Цифровий асистент повинен не просто генерувати граматично правильну фразу, а надавати релевантну, змістовну і перевірену інформацію, що ґрунтується на наявних документах. Для цього важливим є якісний семантичний пошук, налаштування embedding-моделі, а також ретельне формулювання промптів і збереження відповідного контексту. Результати тестування показали, що відповідь може бути формально коректною, але не містити суті або взагалі ігнорувати релевантний фрагмент тексту.

Другою задачею виступає робота з великим обсягом документів. Типове університетське сховище включає десятки або сотні PDF-документів із різною структурою, мовою, форматуванням і ступенем складності. Система повинна витримувати навантаження при завантаженні корпусу документів, забезпечувати швидкий пошук, дозволяти оновлення бази знань без повної переіндексації та зберігати логіку відповідей. Важливим є також виведення джерел або посилань на документ, звідки взята відповідь, що особливо актуально в академічному середовищі.

Третьою складовою задачі є стабільність роботи системи. Це включає не лише технічну надійність (відсутність збоїв, обробка винятків, коректне логування), а й відповідність між тим, що оброблено на бекенді, і тим, що бачить користувач у інтерфейсі. Практичне тестування продемонструвало, що навіть за

наявності правильної логіки на рівні LLM, помилки у frontend- або API-рівні можуть звести нанівець усю функціональність.

Таким чином, формулювання задачі дослідження зводиться до трьох основних напрямів: підвищення якості відповідей на основі локальних документів, забезпечення ефективної обробки великих масивів неструктурованої інформації та досягнення стабільної, контрольованої роботи системи з можливістю масштабування та інтеграції. Саме цим напрямкам і присвячена подальша частина роботи.

ІІІ. ЕКСПЕРИМЕНТАЛЬНА СПРОБА РЕАЛІЗАЦІЇ ЦИФРОВОГО АСИСТЕНТА

3.1 Модель використання цифрового асистента в університеті: користувачі та запити

Розробка цифрового асистента для роботи з документами університету передбачає орієнтацію на різноманітну аудиторію користувачів із різними інформаційними потребами. Для успішної реалізації рішення необхідно чітко визначити типові ролі, сценарії взаємодії, а також характер запитів, які мають оброблятися системою.

Умовно користувачів можна поділити на три основні категорії. Першу категорію складають студенти. Їхні запити зазвичай стосуються академічних питань: умов перескладання, подання апеляцій, термінів здачі дипломів, порядку оформлення академвідпустки, правил переведення або відрахування. Ці питання часто описані в регламентних документах, однак їхнє самостійне опрацювання вимагає часу і зусиль, оскільки інформація розпорошена по різних джерелах.

Другу категорію формують абітурієнти та їхні батьки. У цій групі запити пов’язані з вступною кампанією: подання заяв, графік вступних іспитів, перелік документів, особливості вступу за квотою або на контракт. Інформація з таких питань часто змінюється щороку, тож асистент має швидко адаптуватися до оновлень документів.

Третю категорію становлять викладачі та адміністрація. Їхні потреби включають пошук внутрішніх нормативних документів: положення про кафедру, порядок проходження практики, інструкції з акредитації, шаблони навчальних планів. У цьому випадку запити часто є вузькоспеціалізованими, із використанням офіційної термінології.

Цифровий асистент у такому середовищі має забезпечити:

- пошук релевантної інформації на основі текстових запитів природною мовою (переважно українською)
- посилання на джерело документа, де знайдено відповідь

- толерантність до різних формулювань одного запиту (наприклад, “чи можна перескладати іспит?” ≈ “перездача іспиту”)

Умовно кожна взаємодія може бути зведена до такої моделі:

1. користувач формулює запит у звичній для себе формі
2. система перетворює запит у embedding-вектор
3. відбувається пошук по базі знань, побудованій на нормативних документах
4. обрані фрагменти передаються у мовну модель для генерації відповіді
5. користувач отримує результат, який включає коротке пояснення та посилання на джерело

Таким чином, модель використання цифрового асистента в університеті ґрунтується на поєднанні семантичного пошуку, гнучкого інтерфейсу й адаптації під потреби різних груп. Основна вимога до такої системи — надати відповідь не лише правильно, а й зрозуміло, швидко та з підтвердженням з офіційного джерела.

3.2 Підготовка вхідних даних: PDF, Word, скановані документи

Для перевірки працездатності цифрового асистента в реальних умовах необхідно було сформувати репрезентативний набір документів, з якими потенційно працюють студенти, викладачі або адміністрація. Було зібрано корпус із понад десяти документів, включаючи PDF-файли, DOCX-файли та скановані копії офіційних положень і регламентів.

Основну частину вхідних даних складали PDF-документи, які є типовим форматом збереження нормативної документації у ЗВО. Деякі з них були створені з цифрових джерел і містили селектований текст, інші — були сканованими зображеннями, збереженими у PDF-форматі. Останні потребували додаткової обробки за допомогою технологій оптичного розпізнавання символів (OCR), щоб зробити їх придатними до аналізу.

DOCX-документи використовувались переважно для попереднього тестування та ручної перевірки форматування. Вони легко піддаються обробці, однак не всі протестовані системи підтримували їх безпосереднє завантаження, тому в окремих випадках виконувалось попереднє перетворення у PDF або plain text.

При підготовці даних було виявлено низку типових проблем:

- у PDF-документах, навіть якщо текст виглядав коректно на екрані, при обробці за допомогою парсерів (PyMuPDF, pdfplumber) виявлялись випадки злиття слів через відсутність пробілів між рядками
- текст часто втрачав логічну структуру — заголовки, нумерація, абзаци зміщувались або обривались
- документи містили службову інформацію (номери сторінок, штампи, колонтитули), яку потрібно було фільтрувати, щоб уникнути шуму в embedding
- деякі скановані PDF потребували додаткової обробки (binarization, deskewing) перед застосуванням OCR, і навіть після цього могли містити помилки розпізнавання (наприклад, “і” перетворювалося в “1” або “!”)

У рамках експерименту було прийнято рішення зосередитися на документах, які вже містять машинозчитуваний текст, щоби не втрачати якість

embedding через некоректне розпізнавання. Скановані документи використовувались лише вибірково, для оцінки граничних випадків.

Окрему увагу приділяли збереженню логічної цілісності фрагментів. Замість сліпого поділу тексту по довжині, було рекомендовано — там, де це можливо — розділяти текст за абзацами, пунктами, заголовками. Це дозволяло отримувати чанки з завершеним змістом, що підвищувало релевантність відповідей LLM.

Зібраний і очищений корпус документів було використано на наступних етапах: при завантаженні у системи RAGFlow, Kotaemon та Dify, а також при перевірці точності відповідей на типові запити.

3.3 Завантаження документів у RAGFlow: результат, проблеми токенізації та chunking

RAGFlow було обрано як одне з базових рішень для перевірки можливості побудови цифрового асистента з відкритим кодом. Система надає інструменти для семантичного пошуку в документах, реалізовані через LangChain, FastAPI та векторну базу Chroma. Одним із її переваг є наявність прямого API та можливість вручну налаштувати embedding-моделі, pipeline запитів та параметри chunking.

Процес завантаження документів у RAGFlow передбачав наступні етапи:

1. Розгортання платформи за допомогою docker-compose
2. Підготовка папки з PDF-документами, які підлягають обробці
3. Запуск скрипту парсингу та генерації embedding-векторів (ingest.py або подібного), який обробляє документи і записує їх у векторну базу Chroma
4. Перезапуск API-сервісу для оновлення бази знань та активації оброблених файлів

На практиці цей процес пройшов без критичних помилок, однак одразу виявилися ключові проблеми chunking і токенізації. Незважаючи на формально коректну обробку документів, результати поділу тексту на чанки виявилися нестабільними. Типові помилки включали:

- злиття слів при переході між рядками, що робило деякі фрагменти нечитабельними для embedding-моделі
- обриви речень без завершення, які виникали через механічний поділ за кількістю символів або токенів
- надто короткі чанки, що не містили змістовної інформації, але займали місце в базі

Chunking у RAGFlow реалізовано за замовчуванням як поділ на блоки певної довжини з перекриттям (overlap), однак параметри цього розподілу задаються в коді вручну. Зміна значень дозволяла експериментувати з довжиною фрагментів, але не вирішувала проблему логічної незавершеності тексту. Наприклад, деякі абзаци починались із середини речення або завершувались посеред слова.

Проблеми токенізації частково були пов’язані з інструментами, які витягують текст із PDF. Навіть у документах із машинозчитуваним текстом бібліотеки могли об’єднувати сусідні рядки або пропускати пробіли. Це спотворювало зміст і унеможливлювало адекватне embedding-представлення.

У тестових запитах було виявлено, що відповіді моделі були точними лише у випадках, коли відповідний чанк містив повну смислову одиницю. У решті випадків модель або відповідала узагальнено, або повідомляла, що не знайшла потрібної інформації, навіть якщо вона фактично була присутня в іншому (погано сформованому) чанку.

Попри ці труднощі, RAGFlow показав добру гнучкість у налаштуваннях. Можливість змінювати embedding-модель, використовувати API та запускати скрипти локально дозволяла проводити серію експериментів з різними параметрами, що зробило платформу зручною для дослідницьких цілей.

3.4 Тестування Kotaemon: завантаження, відповідь API, помилки

Проєкт Kotaemon було обрано для тестування як найпростіше з наявних open-source рішень, яке підтримує концепцію семантичного пошуку по документах з використанням embedding і великих мовних моделей. Його основною перевагою є готовність до локального запуску без необхідності додаткового налаштування пайплайнів або написання коду.

Запуск системи здійснювався локально за допомогою Docker. Після клонування репозиторію та виконання команди **docker compose up -d**, система автоматично розгортала бекенд, фронтенд і векторну базу Chroma. Інтерфейс відкривався за адресою **localhost:8000**, де користувач мав змогу завантажити документи та вводити запити.

Для тестування було додано кілька PDF-документів із регламентами університету. Завантаження здійснювалося через веб-інтерфейс, після чого документи оброблялись і записувались у векторну базу. Обробка проходила успішно, однак одразу було виявлено серйозні проблеми з якістю тексту після парсингу: слова на межі рядків злипалися, речення обривалися або зливалися в єдиний фрагмент без логічної структури.

Ці дефекти виявились критичними для якості embedding і відповідно ускладнили генерацію коректних відповідей. Під час запитів на кшталт “Що таке академвідпустка?” або “Які вимоги до дипломної роботи?” система або не знаходила відповіді, або повертала стандартну фразу на кшталт “На жаль, я не знайшов відповідної інформації у наявних документах”.

Найважливішим спостереженням було те, що при перевірці логів (через консоль або в API) LLM усе ж формувала правильну відповідь, але ця відповідь не передавалась користувачеві у фронтенді. Таким чином, фактична помилка полягала не у генерації відповіді, а в її відображенні — тобто проблема була в частині логіки веб-інтерфейсу або його синхронізації з бекендом.

API, що надається Kotaemon, дозволяв формувати запити з власного скрипта, однак його документація виявилась обмеженою. Тестування показало, що система реагує на запити й повертає структуровану відповідь у форматі

JSON, але при складних запитах (загальні формулювання, українська мова) модель могла втрачати релевантність, особливо при низькій якості чанків.

Додатковим обмеженням є відсутність підтримки форматів DOCX або plain text — система обробляє лише PDF, і лише за умови, що документ не є сканованим. Також немає способу вручну перевірити або відредагувати чанки після парсингу: процес є повністю автоматичним і непрозорим.

Таким чином, Kotaemon продемонстрував функціональність у простому середовищі, однак виявився вразливим до якості вхідних PDF, має обмеження у гнучкості та страждає від помилок у відображенні відповідей. Його можна розглядати як навчальну платформу або базу для спрощеного MVP, однак не як рішення для повноцінного цифрового асистента без додаткового доопрацювання.

3.5 Аналіз точності відповідей на основі логів

Оцінка точності роботи систем була проведена як за результатами, видимими у вебінтерфейсі, так і за логами — тобто записами, які генеруються LLM-моделлю у відповідь на запити користувача. Такий підхід дозволив виявити розбіжності між реальною здатністю системи надати відповідь і тим, що бачить кінцевий користувач.

У випадку Kotaemon було виявлено найбільшу розбіжність між відповіддю моделі та тим, що показує інтерфейс. Наприклад, при запиті “Що потрібно для перескладання іспиту?” система у фронтенді повертала стандартну фразу: “На жаль, я не зміг знайти відповідь у документах”. Однак аналіз логів LLM, збережених на бекенді, показував, що модель сформулювала чітку й правильну відповідь з посиланням на відповідний чанк. Це вказує на збій у логіці передачі відповіді до інтерфейсу або на фільтрацію, що хибно визначала відповідь як нерелевантну.

У RAGFlow таких проблем виявлено не було. Всі відповіді, сформовані LLM, передавались через API-контур до користувача. Проте точність самих відповідей залежала від якості embedding-фрагментів. У випадках, коли чанк був сформований правильно (повнота речення, збережена структура), відповідь була коректною і містила цитату з джерела. Якщо ж фрагмент був зіпсований — наприклад, містив злиття слів або обірване речення — модель відповідала або надто загально, або повідомляла, що не знайшла інформації. Логи допомагали точно зрозуміти, як саме модель сприйняла запит, який фрагмент обрала та яким був промпт, сформований для генерації.

У Dify було реалізовано найповніший журнал логів — система зберігає не лише запит, відповідь та використану модель, а й окремо фрагменти з бази знань, які були використані для відповіді (top-k results), з точним зазначенням документу й його розташування. Це дозволило точно простежити весь шлях: від запиту до цитати. Практичне тестування підтвердило, що в більшості випадків відповідь була змістовною, містила частину з документа, і могла бути перевірена вручну. У випадках, коли відповідь була неповною або узагальненою, з логів було видно, що причина полягала в недостатньо інформативному чанку або

слабкому контексті. Така прозорість дозволяє не лише оцінити точність, але й оперативно виявляти джерело помилки (парсинг, chunking, промпт або логіка відповіді).

Загалом, аналіз логів показав:

- лише Dify має повну трасованість запиту до джерела
- RAGFlow забезпечує стабільну передачу відповіді, але без глибокої діагностики
- Kotaemon не передає частину релевантних відповідей користувачу, хоча LLM обробляє їх правильно

Ці спостереження підкреслюють важливість прозорості логування для підвищення довіри до цифрового асистента та забезпечення контролю якості його відповідей.

3.6 Аналіз результатів тестування систем цифрового асистента

У процесі дослідження було протестовано три системи: **Kotaemon**, **RAGFlow** і **Dify**. Основна мета — оцінити якість відповідей, стабільність роботи, логіку обробки документів та придатність до використання у форматі цифрового асистента для університету. Тестування проводилось як самостійно, так і за участі викладачів, які орієнтуються в нормативних документах закладу.

Kotaemon виявив проблему з відображенням відповіді в інтерфейсі: навіть якщо LLM формувала релевантну відповідь (видно у логах), система часто повідомляла “інформація не знайдена”. Це свідчить про технічні труднощі в логіці передачі відповіді до фронтенду або ж некоректну фільтрацію результатів. Така поведінка негативно впливає на довіру до системи з боку кінцевих користувачів, навіть якщо модель функціонує правильно. Крім того, було зафіксовано злиття слів та розірвані речення в результаті chunking. Це знижує точність витягу та породжує хибні чи розмиті відповіді.

RAGFlow, попри добре продуману архітектуру і простоту запуску, не виправдав очікувань у плані змістовної точності. У більшості випадків відповіді були надто загальними або не містили корисного змісту. За оцінками викладачів, багато відповідей були хибними або недостатньо точними. Основними причинами були недосконалий механізм chunking (злиття рядків, некоректна обробка PDF), а також відсутність логічної розмітки або пояснень джерел відповіді.

Dify на початковому етапі тестування також мав серйозну проблему — відповідь LLM не з’являлась у чаті, хоча була сформована. Аналіз логів показав наявність помилки типу TypeError: can only concatenate str (not \"list\") to str, що виникала в модулі easy_ui_based_generate_task_pipeline.py у момент збирання потоку відповіді. Через цю помилку користувач бачив порожнє вікно. Після виявлення причини було протестовано альтернативну модель конфігурації системи — **Knowledge Retrieval + Chatbot**, яка дозволила повністю усунути проблему.

У новій конфігурації Dify надає точні відповіді з прямими цитатами, коректно обробляє документи, логічно показує джерело інформації та історію

обробки. В інтерфейсі тепер видно не лише відповідь, а й використовувані фрагменти тексту. Усі тести пройдено з результатами, що відповідають очікуванням. Викладачі відзначили високу відповідність відповідей тексту документів, а логування підтвердило коректну обробку запитів і добір релевантних фрагментів.

Таким чином, Dify після виправлення критичної помилки став найбільш придатним серед досліджених систем. Його результативність, прозорість відповіді, підтримка API і масштабованість роблять цю платформу найперспективнішою основою для майбутнього цифрового асистента у ВНЗ. У той час як Kotaemon і RAGFlow демонструють лише часткову придатність, Dify надає повноцінну функціональність з мінімальними бар'єрами для інтеграції.

3.7 Перспективи розробки власного рішення на базі існуючих API

Аналіз і тестування відкритих рішень для реалізації цифрового асистента показали, що, попри наявність базової функціональності, жодна з перевірених платформ не забезпечує стабільну, точну і надійну роботу в повному обсязі. Водночас усі три проєкти — Dify, RAGFlow і Kotaemon — надають відкритий API, що дозволяє побудувати власне кастомізоване рішення поверх наявних інструментів.

Найперспективнішою з цієї точки зору виявилася платформа Dify, яка надає REST API для виконання запитів до мовної моделі, роботи з базою знань, журналу повідомлень і управління користувачами. Завдяки цьому можна створити власний фронтенд (наприклад, чат-бот або Telegram-асистент), який буде звертатись до Dify для генерації відповідей, при цьому усуваючи обмеження, виявлені у вбудованому інтерфейсі.

Одним із ключових напрямів розвитку є створення власного інтерфейсу запитів, який:

- самостійно формує промпти у відповідності до мовної специфіки (українська мова, освітня термінологія)
- контролює, як саме користувач бачить відповідь (з цитатою, посиланням на документ, поясненням логіки)
- фіксує лог взаємодії для подальшого аналізу

Такий підхід дозволяє, з одного боку, зберегти переваги існуючих рішень (embedding, індексація, API), а з іншого — уникнути їхніх слабких місць, таких як внутрішні помилки у відображенні відповідей або обмеженість UI.

Іншим напрямом є локалізація та розширення бази знань, зокрема:

- інтеграція з внутрішніми інформаційними системами університету (наприклад, документообіг, Moodle, бази студентів)
- довантаження Word-документів, сканів із розпізнаванням (OCR)
- підтримка актуального оновлення без повної переіндексації бази

Також актуальним є створення системи оцінювання якості відповіді, яка могла б автоматично сигналізувати про неточності, відсутність джерела або

розбіжність між запитом і змістом відповіді. Це дозволить постійно вдосконалювати систему на основі зворотного зв'язку.

Таким чином, використання існуючих API (насамперед Dify) відкриває можливість розробити власного цифрового асистента, адаптованого до контексту українського університету, з кастомним інтерфейсом, стабільною логікою відображення, прозорістю логів та підтримкою локальних документів. Такий підхід є найкращим способом поєднати відкриті технології зі специфічними потребами академічного середовища.

ВИСНОВОК

У ході виконання дипломної роботи на тему **«Розробка інтерактивного цифрового асистента для аналізу і пошуку даних у документах»** було здійснено комплексне дослідження сучасних технологій побудови систем семантичного пошуку з використанням великих мовних моделей. Основною метою роботи стало вивчення можливостей створення цифрового помічника, який би міг швидко та точно відповідати на запити, що стосуються нормативних документів університету, орієнтуючись на потреби студентів, викладачів та адміністрації.

Аналіз існуючих інструментів та їх практичне тестування дозволили зробити важливі висновки щодо придатності кожного з рішень до використання у межах закладу вищої освіти. Дослідження виявило, що, попри доступність і функціональність багатьох open-source платформ, більшість з них не позбавлені критичних обмежень. У деяких випадках це стосується нестабільного відображення відповіді, у інших — низької точності генерації через погану якість розбиття тексту на фрагменти або неправильну обробку PDF-документів.

Особливо важливими стали результати експериментального етапу роботи. Було протестовано три платформи: Kotaemon, RAGFlow і Dify. У Kotaemon відповідь моделі формувалась коректно, однак не відображалась у інтерфейсі. У RAGFlow, незважаючи на добру технічну реалізацію, відповіді були незадовільними з точки зору викладачів, які брали участь у тестуванні. Вони вказували на неточності або повну відсутність релевантної інформації. Dify виявився найбільш зрілим рішенням, однак і він мав серйозну помилку — у ряді випадків сформована відповідь не потрапляла в чат, що було підтверджено аналізом логів. Помилка TypeError, зафіксована у журналі системи, свідчить про технічну несправність в обробці потоку відповіді.

Важливою складовою дослідження став аналіз вхідних документів, які використовуються як джерело знань для асистента. Було виявлено, що навіть у структурованих PDF-файлах часто виникають проблеми з форматуванням: злиття слів, порушення абзаців, обриви речень. Це суттєво впливає на якість побудови embedding-представлень, а отже, і на кінцеву відповідь моделі. У таких

умовах особливої ваги набуває етап попередньої обробки — очищення тексту, корекція структури, перевірка логічної цілісності фрагментів.

Усі розглянуті системи мають потенціал, проте потребують адаптації до реального освітнього середовища. На цьому тлі найбільш перспективним напрямом виглядає створення власного рішення на базі вже готових API — зокрема, Dify. Такий підхід дозволяє поєднати готові інструменти обробки даних із кастомним інтерфейсом, спеціально розробленим під українську мову, освітню термінологію та внутрішню структуру університету.

На основі проведеної роботи можна стверджувати, що реалізація цифрового асистента на базі LLM є технічно можливою вже сьогодні, однак вимагає комплексного підходу. Необхідно поєднати якісний корпус документів, стабільну архітектуру сервісу, гнучке логування, локалізований інтерфейс і механізми оцінки якості відповідей. Саме такий підхід забезпечить створення дійсно корисного інструменту, який зможе полегшити доступ до нормативної інформації для студентів, абітурієнтів і викладачів, а також підвищити прозорість інформаційних процесів у закладі вищої освіти.

Отже, результати дослідження не лише підтверджують актуальність теми, але й окреслюють реальні шляхи розвитку. У майбутньому можливим є створення MVP-продукту з власним інтерфейсом, що використовує API обраної платформи (наприклад, Dify), інтегрується з внутрішніми системами університету, підтримує багатомовність та адаптується до змін у нормативній базі. Це дозволить перетворити експериментальну реалізацію на стабільний практичний інструмент, здатний ефективно функціонувати в реальному середовищі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Dify – https://github.com/langgenius/dify
2. RAGFlow – https://github.com/aisolab/ragflow
3. Kotaemon – https://github.com/kotaemon-ai/kotaemon
4. ChatGPT (OpenAI) – https://chat.openai.com
5. Hugging Face – https://huggingface.co
6. LangChain – https://docs.langchain.com
7. OpenAI API Documentation – https://platform.openai.com/docs
8. Chroma – https://www.trychroma.com
9. FAISS – https://github.com/facebookresearch/faiss
10. pdfplumber – https://github.com/jsvine/pdfplumber
11. PyMuPDF (fitz) – https://pymupdf.readthedocs.io
12. Weaviate – https://weaviate.io
13. Qdrant – https://qdrant.tech
14. Vespa.ai – https://vespa.ai
15. Instructor-XL (embedding model) – https://huggingface.co/hkunlp/instructor-xl
16. all-MiniLM-L6-v2 – https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
17. Cohere AI – https://cohere.com
18. Mistral AI – https://mistral.ai
19. Docker Documentation – https://docs.docker.com
20. Docker Compose – https://docs.docker.com/compose
21. FastAPI – https://fastapi.tiangolo.com
22. Flask – https://flask.palletsprojects.com
23. Gunicorn – https://docs.gunicorn.org
24. Python documentation – https://docs.python.org/3/
25. Wikipedia – https://wikipedia.org

ДОДАТКИ

Додаток А. Структура проєктів та файлова організація

Додаток Б. Скріншоти інтерфейсів Dify, Kotaemon, RAGFlow

Додаток В Приклади запитів і відповідей під час тестування систем

Додаток А

Структура проєкту Dify

```
dify/
├── api/
│   ├── apps/
│   ├── db/
│   └── utils/
├── agent/
│   ├── component/
│   ├── templates/
├── deepdoc/
│   └── parser/
├── docker/
│   └── docker-compose.yml
├── docs/
│   ├── guides/
│   ├── chat/
│   └── dataset/
├── web/
│   ├── src/
│   │   ├── components/
│   │   ├── icons/
│   │   └── pages/
├── sdk/
│   └── python/
│       └── ragflow_sdk/
```

Структура проєкту RAGFlow

```
ragflow/
├── libs/
│   └── kotaemon/
│       ├── agents/
│       ├── embeddings/
│       ├── indices/
│       ├── llms/
│       ├── loaders/
│       └── qa/
├── ktem/
│   ├── chat/
│   ├── db/
│   ├── embeddings/
│   ├── index/
│   ├── llms/
│   └── prompt_optimization/
├── docs/
│   ├── index.md
│   └── usage.md
├── scripts/
│   └── run_linux.sh
├── app.py
├── launch.sh
└── README.md
```

Структура проєкту Kotaemon

kotaemon/

```
├── ktem/
│   ├── app.py
│   ├── chat/
│   ├── db/
│   ├── embeddings/
│   ├── index/
│   ├── llms/
│   ├── prompt_optimization/
│   ├── utils/
├── docs/
│   └── usage.md
├── tests/
│   └── test_qa.py
├── migrations/
├── run_local.sh
└── README.md
```

Додаток Б

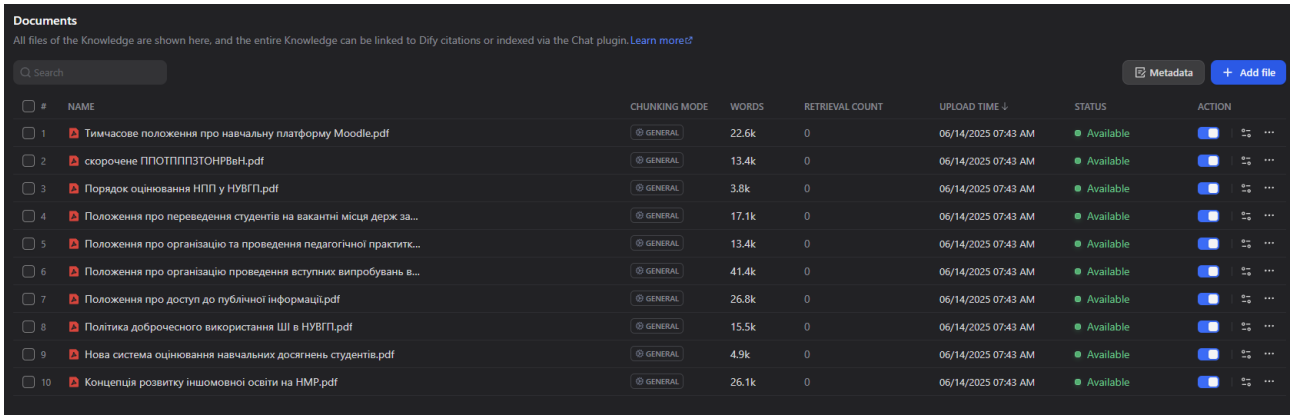

Рисунок Б.1 - Інтерфейс платформи Dify після завантаження документа

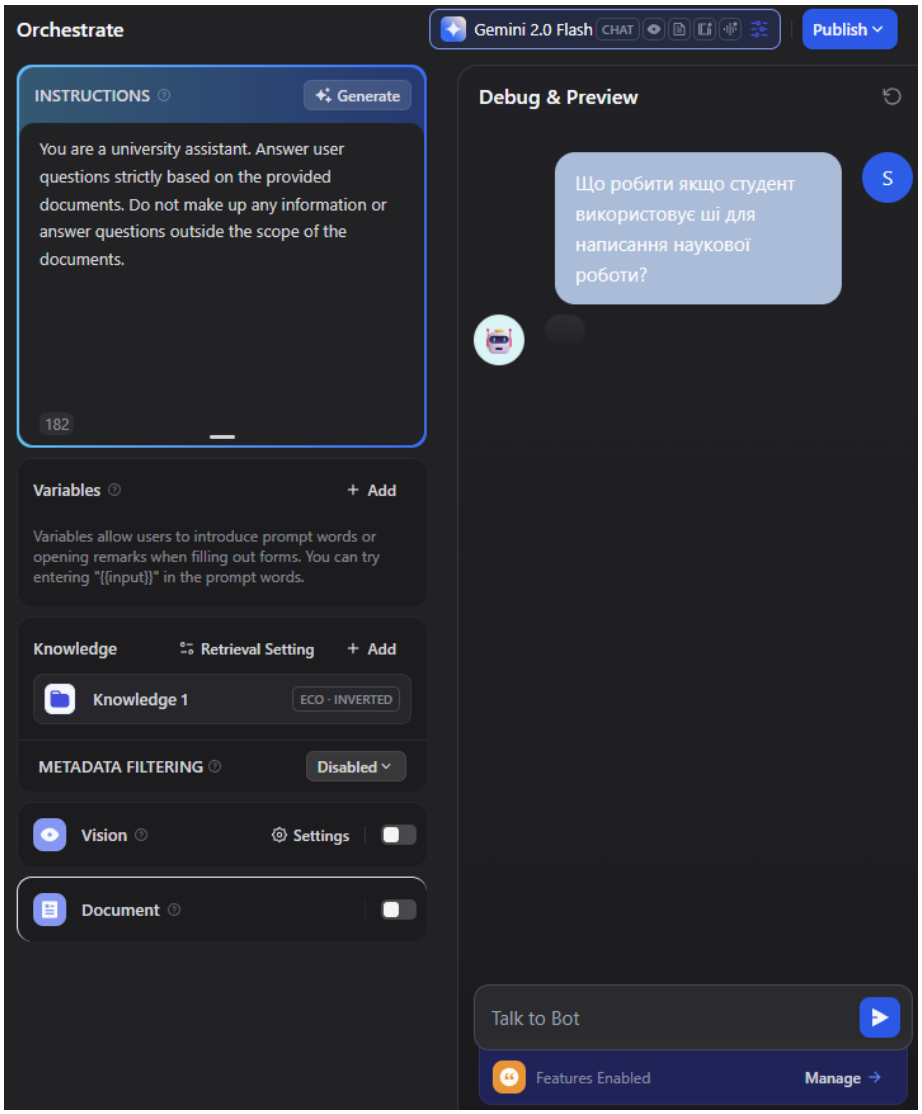

Рисунок Б.2 - Випадок, коли відповідь не відображається у вікні чату Dify

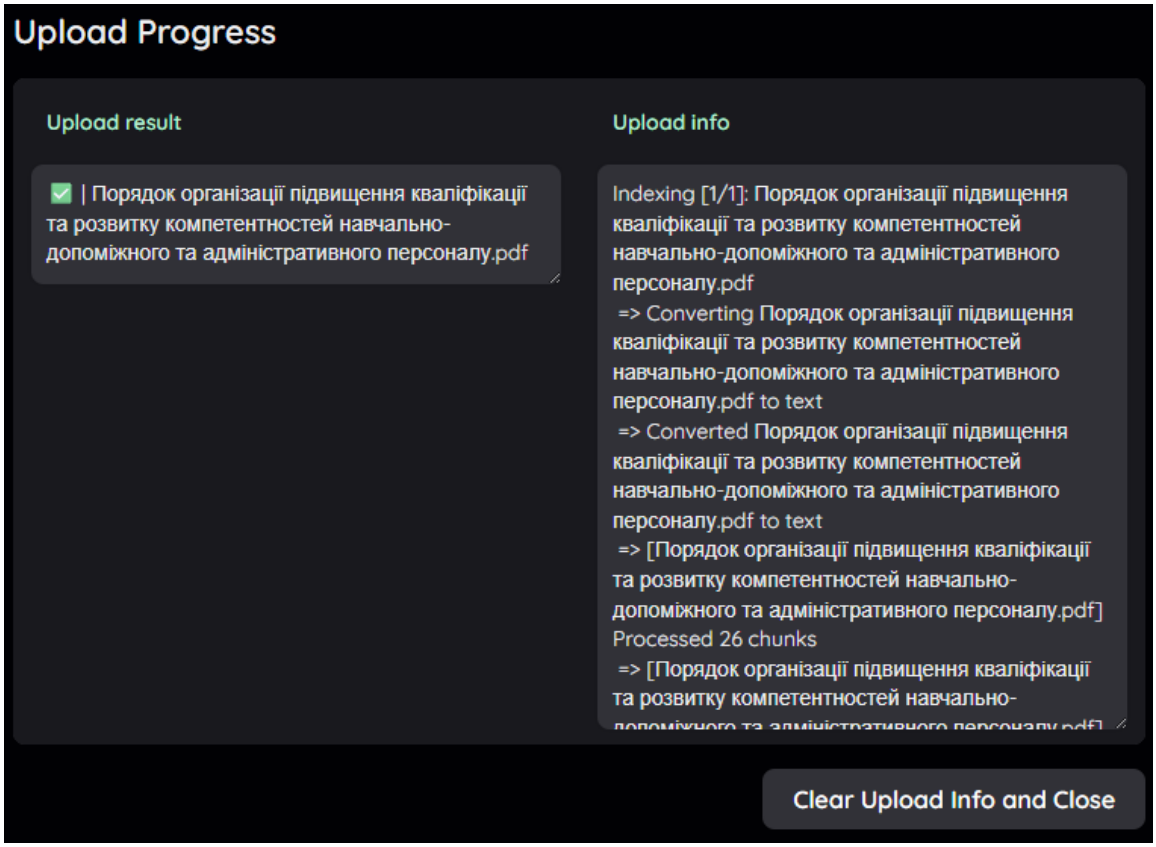

Рисунок Б.3 - Інтерфейс Kotaemon після завантаження PDF-документа

На скріншоті показано стан завантаження та ініціалізацію бази знань.

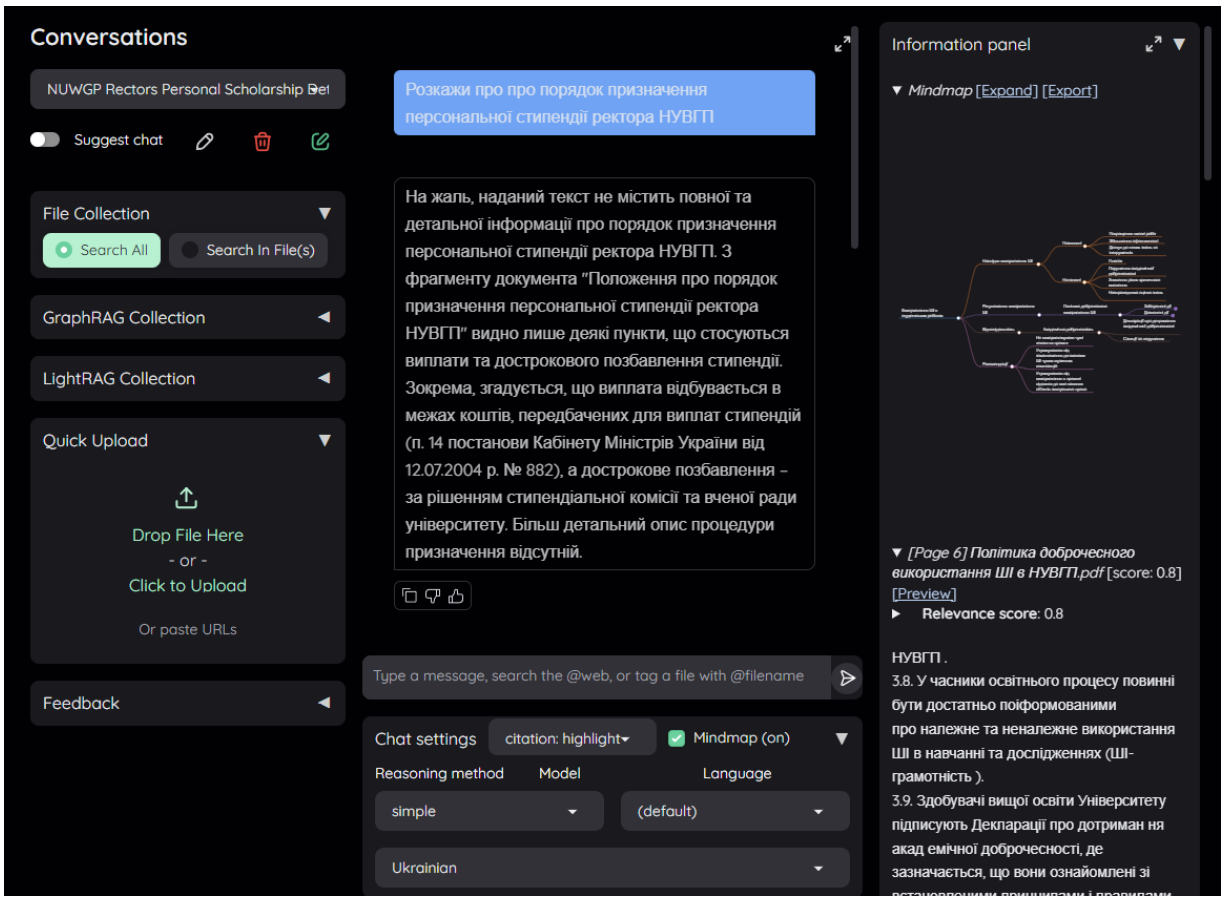

Рисунок Б.6 - Відповідь Kotaemon: “Інформація не знайдена”

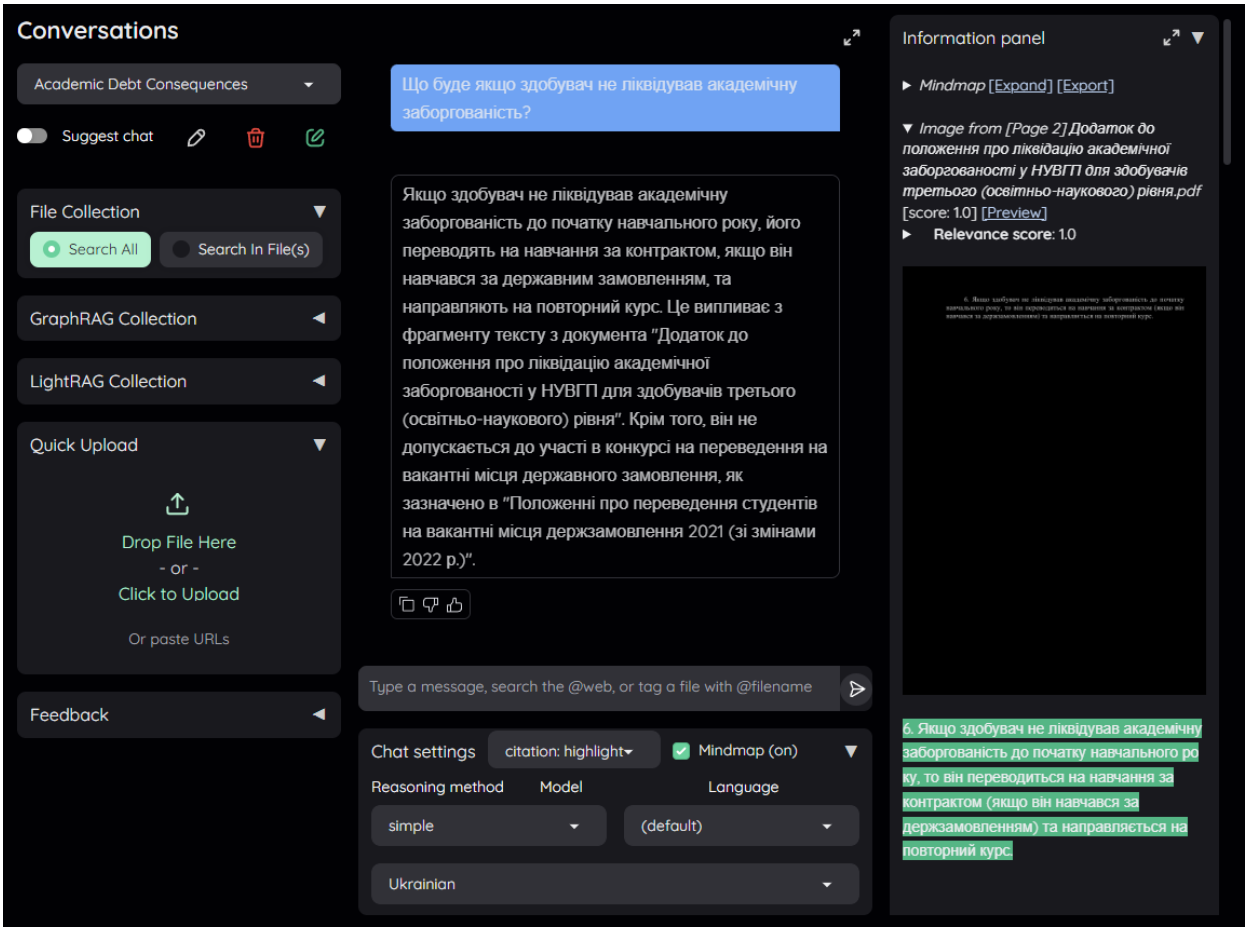

Рисунок Б.7 - Лог Kotaemon з правильно згенерованою відповіддю мовною моделлю

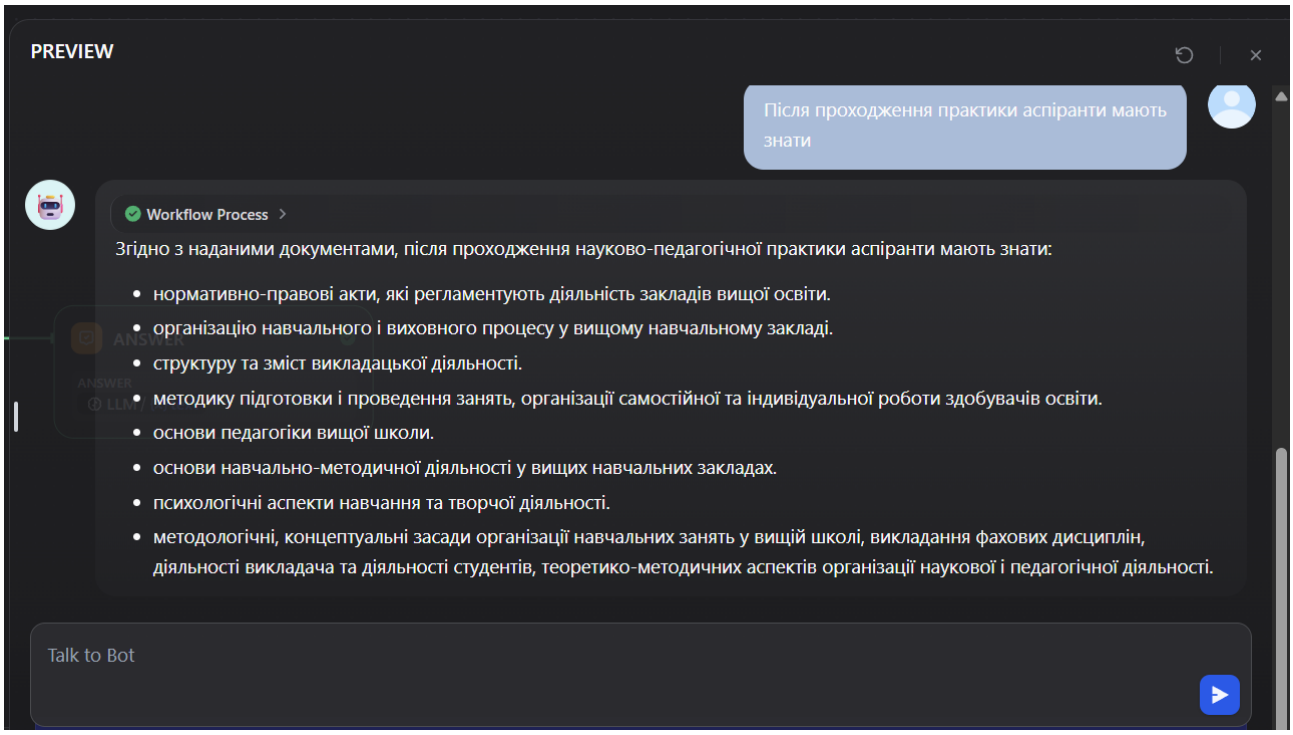

Рисунок Б.8 - Успішне надавання відповіді Dify при варіанті Knowledge Retreival + Chatbot

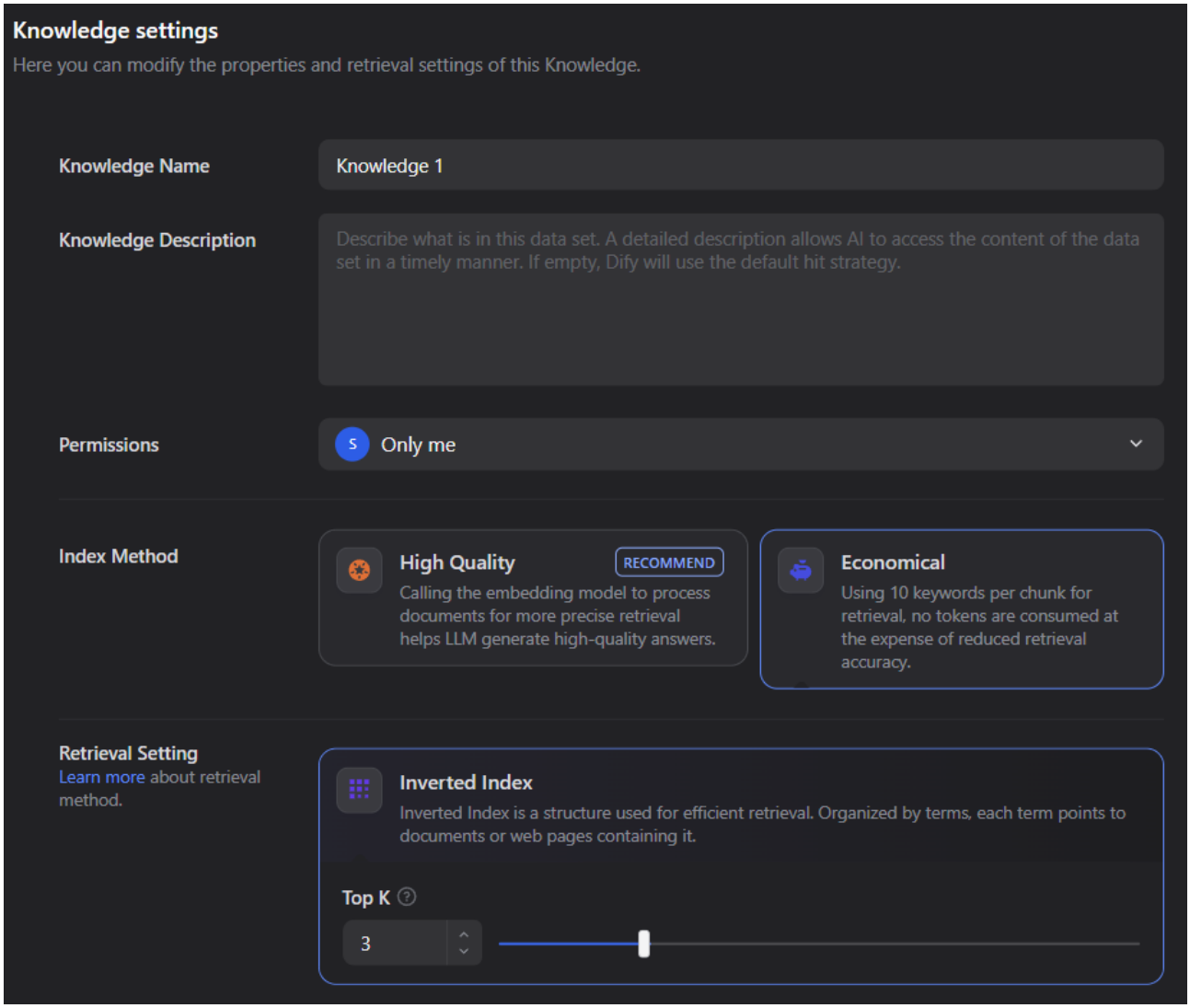

Рисунок Б.9 – Налаштування Книги знань в проекті Dify

Додаток В

Документ: Додаток до положення про ліквідацію академічної заборгованості

Питання: Коли у здобувача третього рівня виникає академічна заборгованість?

Очікувана відповідь: Коли він отримав незадовільну оцінку при складанні підсумкового контролю на екзаменаційній комісії

Kotaemon: Академічна заборгованість у здобувача третього (освітньо-наукового) рівня виникає тоді, коли він отримує незадовільну оцінку на підсумковому контролі, який проводиться екзаменаційною комісією...

RAGFlow: Академічна заборгованість у здобувача третього рівня виникає у випадку, коли він отримав незадовільну оцінку при складанні підсумкового контролю на екзаменаційній комісії.

Документ: Тимчасове положення про навчальну платформу Moodle

Питання: Яке призначення платформи Moodle в НУВГП?

Очікувана відповідь: Забезпечення освітніх процесів із використанням дистанційних технологій, включно з контролем знань і перевіркою подібності робіт

Kotaemon: ...Moodle використовується як інструмент для організації та підтримки навчального процесу, забезпечення комунікації та управління навчальними матеріалами...

RAGFlow: ...управління користувачами, ролями, дистанційним процесом, комунікацією між здобувачами і викладачами...

Документ: Концепція розвитку іншомовної освіти

Питання: Яка головна мета розвитку іншомовної освіти?

Очікувана відповідь: Підвищення рівня іншомовної комунікативної компетентності громадян для інтеграції в європейський простір

Kotaemon: Немає відповіді — символи пошкоджені / неможливо прочитати

RAGFlow: Мета — створення умов для досягнення визначених рівнів володіння мовою за рекомендаціями Ради Європи

Документ: Політика доброчесного використання ШІ в НУВГП

Питання: Що вважається академічним плагіатом при використанні ШІ?

Очікувана відповідь: Оприлюднення згенерованих ШІ результатів як власних без зазначення авторства або деталей використання ШІ

Kotaemon: Надано широкий перелік форм академічного плагіату з прикладами порушень і формулюваннями із внутрішніх положень

RAGFlow: Перераховано загальні напрямки політики ШІ (освіта, дослідження, безпека, етика), але прямої відповіді на питання немає

Документ: Положення про переведення студентів на вакантні місця держзамовлення

Питання: За яких умов студент може бути переведений на вакантне місце держзамовлення?

Очікувана відповідь: За конкурсом, у межах тієї ж спеціальності та курсу, за умови повної оплати та згоди замовника

Kotaemon: Надано розгорнутий перелік умов, включаючи винятки, додаткові підстави, категорії осіб та внутрішні регламенти

RAGFlow: Узагальнено перелік вимог (конкурс, спеціальність, згода, оплата) — ближче до очікуваної відповіді

Документ: Порядок оцінювання НПП у НУВГП

Питання: Яка мета щорічного оцінювання НПП у НУВГП?

Очікувана відповідь: Моніторинг досягнень і стимулювання професійного розвитку на засадах прозорості й об'єктивності

Kotaemon: Надано відповідь близьку до очікуваної: стимулювання, оцінка досягнень, розвиток

RAGFlow: Відсутня пряма відповідь — лише інформація про оприлюднення результатів

Документ: Нова система оцінювання навчальних досягнень студентів

Питання: Як оцінюється студент за новою 100-бальною системою?

Очікувана відповідь: 60 балів — поточна складова, 40 — модульна/підсумкова, мінімум 60 для зарахування

Kotaemon: Подано дуже детальне пояснення з прикладами сценаріїв оцінювання, модульних контрольних, перескладання тощо

RAGFlow: Узагальнено описано логіку 100-бальної шкали та можливості перездач — відповідь майже повна

Документ: Положення про доступ до публічної інформації

Питання: Яка інформація є публічною в НУВГП?

Очікувана відповідь: Задокументована інформація, створена чи отримана університетом у межах повноважень

Kotaemon: Подано перелік типових публічних документів і формулювання з положень — частково деталізовано

RAGFlow: Наведено приклади публікацій на сайті НУВГП, проте не узагальнено визначення публічної інформації

Документ: Положення про педагогічну практику здобувачів третього рівня

Питання: Яка основна мета педагогічної практики здобувачів третього рівня?

Очікувана відповідь: Формування компетентностей для здійснення освітнього процесу

Kotaemon: Подано відповідь із деталізацією компетентностей, поєднанням теорії та практики

RAGFlow: Вказано набуття знань, презентаційних навичок, ІКТ — часткова відповідність

Документ: Положення про організацію вступних випробувань у 2025 році

Питання: Які форми вступних випробувань передбачає НУВГП у 2025 році?

Очікувана відповідь: НМТ, ЄВІ, ЄФВВ, фахові іспити, творчі конкурси, співбесіди, іспити для іноземців, презентація досліджень

Kotaemon: (відповідь відсутня)

RAGFlow: (відповідь відсутня)