МАГІСТЕРСЬКА РОБОТА

МР.ІПм – 31.00.00.000 ПЗ

Група ІПм-24-2

Іванов Владислав

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Іванов Владислав Володимирович

(прізвище, ім'я, по батькові)

УДК 004.8

(індекс)

МАГІСТЕРСЬКА РОБОТА

Застосування засобів штучного інтелекту в рамках соціального нетворкінгу для підвищення ефективності навчальної діяльності студентів

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 – Інженерія програмного забезпечення

(шифр і назва спеціальності)

Іванов В. В.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Григорчук Любомир Іванович, к. п. н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

доц. В. В. Бандура

(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Р. Б. Вовк

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітньо-кваліфікаційний рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою ІПЗ
доц. В. В. Бандура
“___” _____ 2025 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Іванову Владиславу Володимировичу
(прізвище, ім’я, по батькові)

1. Тема магістерської роботи: «Застосування засобів штучного інтелекту в рамках соціального нетворкінгу для підвищення ефективності навчальної діяльності студентів» керівник проекту (роботи) Григорчук Любомир Іванович, к. т. н., доцент
затверджено наказом вищого навчального закладу від «05» листопада 2025 р. № 695/7

2. Строк виконання студентом проекту (роботи): 12 грудня 2025 року

3. Вихідні дані до проекту (роботи): архітектура розподілених систем, результати і матеріально-технічна база, отримана під час проходження науково-дослідної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):

1. Аналіз сучасного стану цифровізації освіти та впливу систем штучного інтелекту
2. Аналіз перспективних напрямів ведення дослідження
3. Вибір технологій та обґрунтування архітектури системи
4. Реалізація програмних модулів демонстраційного проєкту

5. Перелік графічного матеріалу (з точним забезпеченням обов’язкових креслень)

1. Вигляд ETL-пайплайну індексації файлів у векторну БД (рис. 3.1, ст. 59)
2. Архітектура системи та підключення до клієнтської частини (рис. 4.3, ст. 71)
3. Діаграма отримання через ШІ доступу до кешованих даних (рис. 4.4, ст. 72)
4. Діаграма комунікації мікросервісів через відповідні API та порти (рис. А.1, ст. 105)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Перевірка на плагіат	доц. к. т. н. Вовк Р. Б.		

7. Дата видачі завдання: _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області та постановка задачі	24.10.25	виконано
2	Проєктування архітектури та вибір технологій	04.11.25	виконано
3	Підготовка даних та розгортання інфраструктури	07.11.25	виконано
4	Розробка підсистеми семантичного пошуку	20.11.25	виконано
5	Реалізація соціального модуля та інтерфейсу	01.12.25	виконано
6	Завершення реалізації основного функціоналу, тестування та валідація MVP	03.12.25	виконано
7	Оформлення роботи та підготовка до захисту	12.12.25	виконано

Студент-магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 109 с., 23 рис., 6 табл., 46 джерел, 3 додатки.

Тема: Застосування засобів штучного інтелекту в рамках соціального нетворкінгу для підвищення ефективності навчальної діяльності студентів.

Об’єкт дослідження: процеси інформаційної підтримки та соціальної взаємодії учасників освітнього процесу у вищих навчальних закладах.

Мета роботи: підвищення ефективності навчальної діяльності студентів шляхом розробки гібридної інтелектуальної системи, яка забезпечує уніфікований доступ до навчальних матеріалів та автоматизований пошук експертної підтримки в академічній спільноті.

Предмет дослідження: методи та засоби підвищення ефективності навчальної діяльності шляхом інтеграції технологій генеративного штучного інтелекту, семантичного пошуку та алгоритмів соціальних зв’язків.

Результати дослідження:

Проаналізовано проблему фрагментації сучасних освітніх інструментів. Обґрунтовано архітектуру типу «єдине вікно». Розроблено систему семантичного пошуку на базі векторної БД із застосуванням ембеддингів із вбудованими метаданими для підвищення релевантності видачі навчальних матеріалів шляхом збагачення контекстом. Реалізовано соціальний модуль, що використовує «академічний цифровий слід» студента з Moodle для автоматичного підбору партнерів для навчання. Створено повнофункціональний прототип системи.

Висновок:

Створена гібридна система продемонструвала ефективність у вирішенні задачі інформаційної студентів. Поєднання генеративного ШІ для роботи з документами та соціальної взаємодії дозволяє створити освітнє середовище, що сприяє покращенню академічних результатів та навчання.

ГЕНЕРАТИВНИЙ ШТУЧНИЙ ІНТЕЛЕКТ, СОЦІАЛЬНИЙ НЕТВОРКІНГ, ВЕКТОРНИЙ ПОШУК, ПІРИНГОВЕ НАВЧАННЯ.

ANNOTATION

Master's thesis: 109 pages, 23 figures, 6 tables, 46 sources, 3 appendices.

Topic: The use of artificial intelligence in social networking to improve the effectiveness of student learning.

Object of research: processes of information support and social interaction of participants in the educational process in higher education institutions.

Purpose of the work: to improve the effectiveness of students' learning activities by developing a hybrid intelligent system that provides unified access to learning materials and automated search for expert support in the academic community.

Subject of research: methods and means of improving the effectiveness of educational activities through the integration of generative artificial intelligence (GenAI) technologies, semantic search, and social connection algorithms.

Research results:

The paper analyzes the problem of fragmentation of modern educational tools. The architecture of the “single window” type justified. A semantic search subsystem based on a vector database with embedded metadata has been developed, which has led to an increase in the relevance of educational materials by context enriching them. A social module has been implemented that uses the student's “academic digital footprint” from LMS Moodle to automatically select competent learning partners, considering their academic performance. A fully functional prototype of the system has been created.

Conclusion:

The created hybrid system has demonstrated effectiveness in solving the problem of student information. The combination of generative AI for working with documents and social interaction allows the creation of an educational environment that contributes to the improvement of academic performance and learning.

GENERATIVE ARTIFICIAL INTELLIGENCE, SOCIAL NETWORKING, VECTOR SEARCH, PEER LEARNING.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ІТ – інформаційні технології (information technology)

ВНЗ – вищий навчальний заклад

LMS – система керування навчанням/навчальним процесом (learning management system)

FAQ – часті питання (frequently asked questions)

HTML – мова розмітки гіпертексту (hypertext markup language)

JSON – запис об’єктів JavaScript (JavaScript object notation)

БД (DB) – база даних (database)

MVP – мінімально життєздатний продукт (minimum viable product)

SDK – набір інструментів розробки (software development kit)

SPA – застосунок однієї сторінки (single page application)

API – інтерфейс програмування застосунків (application programming interface)

REST – передавання репрезентативного стану (representational state transfer)

HTTP – протокол передачі гіпертексту (hypertext transfer protocol)

HTTPS – безпечний протокол передачі гіпертексту (hypertext transfer protocol secure)

ШІ (AI) – штучний інтелект (artificial intelligence)

ML – машинне навчання (machine learning)

GenAI – генеративний штучний інтелект (generative artificial intelligence)

LLM – велика мовна модель (large language model)

RAG – генерування з розширеним/доповненим пошуком (retrieval-augmented generation)

ETL – витягання, перетворення, завантаження (extract, transform, load)

ВСТУП

Актуальність теми. Сучасний етап розвитку вищої освіти характеризується стрімким процесом цифровізації навчального середовища. Вимушений перехід до дистанційних форм навчання, спричинений пандемією COVID-19 та подальшими кризовими явищами, прискорив впровадження систем управління навчанням (LMS) і онлайн-комунікацій. Водночас аналіз існуючого IT-ландшафту університетів свідчить про проблему фрагментації інструментарію: навчальні матеріали, адміністративні дані та канали взаємодії часто розміщені на ізольованих платформах, що створює додаткове когнітивне навантаження на здобувачів освіти.

Паралельно з цим, поява великих мовних моделей і генеративного штучного інтелекту відкрила нові можливості для персоналізації навчання. Однак використання базових моделей в академічному середовищі обмежене проблемою галюцинацій, відсутністю доступу до локальних даних університету та ізольованістю студента від академічної спільноти. Усе це зумовлює необхідність створення інтелектуальних рішень, які поєднують доступ до навчального контенту та соціальної підтримки, забезпечуючи підвищення ефективності навчальної діяльності.

Метою роботи є підвищення ефективності навчальної діяльності студентів шляхом розробки гібридної інтелектуальної системи, яка забезпечує уніфікований доступ до навчальних матеріалів та автоматизований пошук експертної підтримки в академічній спільноті для вирішення вище зазначених проблем.

Досягнення мети включало розв’язання таких задач:

1. Провести аналіз існуючих підходів до цифровізації освіти та виявити обмеження сучасних LMS і систем генеративного ШІ.
2. Обґрунтувати архітектуру гібридної системи, що поєднує семантичний пошук по документах та соціальний нетворкінг.

3. Розробити методику індексації навчальних матеріалів для векторного пошуку з урахуванням специфіки академічного контенту.

4. Розробити алгоритм підбору партнерів/менторів для навчання на основі аналізу цифрового сліду успішності студента.

5. Здійснити програмну реалізацію та тестування прототипу системи у вигляді мікросервісного додатку.

Об’єктом дослідження є процеси інформаційної підтримки та соціальної взаємодії учасників освітнього процесу у вищих навчальних закладах.

Предметом дослідження є методи та засоби підвищення ефективності навчальної діяльності шляхом інтеграції технологій на основі генеративного штучного інтелекту, семантичного пошуку та алгоритмічної оркестрації соціальних зв’язків.

Методи дослідження. У роботі використано методи системного аналізу для визначення вимог до системи; методи обробки природної мови (NLP) та векторизації тексту для реалізації семантичного пошуку; теорію графів та множин для побудови соціального модуля; методи об'єктно-орієнтованого програмування та мікросервісної архітектури для реалізації програмного комплексу.

Наукова новизна одержаних результатів:

1. Удосконалено метод Retrieval-Augmented Generation (RAG) для генеративного штучного інтелекту в освітніх системах шляхом попереднього збагачення фрагментів навчальних матеріалів контекстними метаданими, що забезпечує підвищення релевантності сформованих відповідей.

2. Запропоновано підхід до виявлення носіїв неформалізованих («мовчазних») знань у студентських спільнотах на основі аналізу академічного

цифрового сліду, доступного в LMS Moodle, що створює передумови для автоматизації менторської підтримки.

3. Розроблено інтегровану архітектуру інтелектуального навчального помічника, яка поєднує семантичний пошук у знаннях університету та сервіс соціальної взаємодії студентів на основі мікросервісного підходу та єдиного інтерфейсу.

4. Сформовано модель соціального графа знань студентів із використанням вагових коефіцієнтів академічної активності та перетину навчальних курсів, що дозволяє підвищити обґрунтованість рекомендацій щодо навчальних партнерів.

Практичне значення одержаних результатів. Розроблено повнофункціональний прототип системи на базі мікросервісної архітектури (Python, FastAPI, Milvus, Redis, React), який забезпечує інтеграцію з LMS Moodle. Система готова до розгортання і дозволяє студентам отримувати релевантні відповіді на запитання по матеріалах курсів університету та знаходити менторів для вирішення складних завдань.

Структура та обсяг роботи. Магістерська робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Повний обсяг роботи становить 109 сторінок.

РОЗДІЛ 1.
АНАЛІЗ СУЧАСНОГО СТАНУ ЦИФРОВІЗАЦІЇ ОСВІТИ ТА ВПЛИВУ СИСТЕМ НА ОСНОВІ ШТУЧНОГО ІНТЕЛЕКТУ

1.1. Аналіз комунікації та обміну знаннями у навчальних закладах

Інформаційна екосистема сучасного закладу вищої освіти є складною, багаторівневою структурою, де процеси передачі даних виходять далеко за межі традиційного аудиторного навчання. Досліджуючи природу інформаційних потоків, доцільно розділити їх на два фундаментальні типи: вертикальні та горизонтальні. Вертикальна комунікація, що відбувається за вектором «адміністрація-студент» або «викладач-студент», зазвичай регламентована, формалізована та спрямована на доставку навчального контенту чи адміністративної інформації. Натомість горизонтальна комунікація або ж пірингова взаємодія (англ. peer-to-peer interaction) відбувається безпосередньо між здобувачами освіти та базується на неформальних зв'язках.

Численні дослідження вказують на те, що саме горизонтальний рівень комунікації часто стає вирішальним фактором у засвоєнні матеріалу та загальній академічній успішності. Як зазначається у систематичних оглядах, соціальні мережі та підтримка від однолітків є критичними чинниками, що корелюють із високими показниками успішності студентів, особливо серед тих груп, що потребують додаткової адаптації [28], що підтверджує тезу, що освітній процес – це не лише індивідуальне споживання інформації, а й колективна діяльність, де обмін досвідом відіграє роль каталізатора знань.

Для глибшого розуміння цього феномену необхідно ввести поняття соціального капіталу (англ. social capital). У науковій літературі соціальний капітал визначається як сукупність ресурсів, що стають доступними індивіду або групі через володіння мережею відносин взаємного визнання та довіри. Його структурними компонентами виступають соціальні мережі, довіра, а

також спільні норми та цінності [6]. У контексті університету соціальний капітал проявляється у здатності студентів ефективно взаємодіяти для вирішення навчальних проблем, ділитися нотатками, пояснювати одне одному складні теми та надавати підтримку. Дослідження підтверджують, що наявність розвиненого соціального капіталу в навчальних мережах (англ. learning networks) значно покращує результати обміну знаннями [27]. До цього також додається, що університет як інституція відіграє ключову роль у формуванні цього капіталу, створюючи середовище для виникнення стійких зв'язків.

Однак сучасний стан цифровізації освіти створює парадоксальну ситуацію. З одного боку, студенти активно використовують онлайн-соціальні мережі, які стали для них природним простором для нетворкінгу та обміну ідеями [7]. Неформальні практики використання соціальних медіа мають потенціал суттєво оптимізувати робочі та навчальні процеси, заповнюючи прогалини, залишені офіційними каналами.

З іншого боку, спостерігається критична розрізненість цифрових платформ. Офіційні системи управління навчанням, фокусуються на доставці контенту та оцінюванні, фактично ігноруючи соціальний аспект. Водночас реальна комунікація та обмін досвідом, який важко формалізувати, мігрують у приватні месенджери (як-от: Telegram, WhatsApp), що призводить до хаотичності: цінна інформація губиться у чатах, а доступ до експертизи обмежується колами найближчих друзів.

Варто також зауважити, що неформальний обмін знаннями не є панацеєю і має свої обмеження. Дослідження феномену «слабких зв'язків» у студентських групах показують, що соціальна взаємодія може бути неефективною без належної організації. Зокрема, існує тенденція до кластеризації: успішні студенти спілкуються переважно з іншими успішними студентами, тоді як ті, хто потребує допомоги, можуть опинитися в ізоляції або в групах з низьким рівнем академічної експертизи, що не сприяє їхньому прогресу [37]. Крім того, наукова спільнота відзначає фрагментарність

досліджень щодо механізмів обміну знаннями серед студентів, що ускладнює створення універсальних методик покращення цього процесу [19].

Підсумовуючи вище викладене, можна стверджувати, що сучасна цифрова екосистема вищої освіти подекуди страждає від розриву між інструментами доступу до матеріалів та інструментами соціальної взаємодії. Хоч соціалізація і є потужним чинником формування соціального капіталу, відсутність деякої долі керованості цим процес, навіть незначної, призводить до втрати значного обсягу знань, нерівномірності розподілу знань та зниження ефективності навчання окремих груп чи індивідуумів.

1.2. Цифровізація освітніх процесів

Сучасна трансформація освітнього ландшафту характеризується інтеграцією цифрових технологій, що докорінно змінює методологію викладання та адміністрування навчального процесу. Під терміном «цифровізація освіти» (англ. digitalization of education) слід розуміти процес переходу від аналогових форм передачі знань до використання цифрових інструментів, платформ та екосистем із метою підвищення ефективності, доступності та гнучкості навчання. Хоча розвиток інтернет-технологій планомірно готував ґрунт для цих змін протягом останніх десятиліть, справжнім каталізатором, що змусив глобальну академічну спільноту форсувати впровадження інновацій, стала пандемія COVID-19. Кризові умови виступили свого роду стрес-тестом для освітніх систем, прискоривши імплементацію хмарних технологій, які забезпечили стійкість та безперервність навчального процесу навіть в умовах тотальної ізоляції [41].

Масовий перехід в онлайн-формат не лише став вимушеним заходом реагування на зовнішні для сфери «загрози», а й дозволив накопичити значну емпіричну базу для оцінки ефективності цифрових інструментів. Проведений метааналіз, що охоплює широкий спектр досліджень, демонструє стійку кореляцію між використанням цифрових платформ та покращенням

навчальних результатів студентів [13], що спростовує поширений міф про дистанційне навчання як про неповноцінну освіту. Водночас науковці наголошують на важливості моделі змішаного навчання – підходу, що органічно поєднує онлайн-інструменти з класичними аудиторними практиками, дозволяючи оптимізувати організаційну структуру сучасного університету [45].

Ключовим елементом цифрової інфраструктури ВНЗ стали системи управління навчанням (англ. Learning Management Systems, LMS). Сучасний ринок освітніх технологій пропонує різноманітні платформи, що суттєво відрізняються моделлю ліцензування, відкритістю архітектури та інструментами для організації комунікації. Систематизований аналіз ключових технічних та функціональних характеристик найпоширеніших систем наведено в табл. 1.1.

Таблиця 1.1

Порівняльна характеристика систем управління навчання

Критерій	**Moodle**	**Google Classroom**	**Canvas**	**Blackboard Learn**
Ліцензія, вартість	Open Source (GPL), безкоштовна	Комерційна, але безкоштовна для закладів освіти	Комерційна (SaaS)	Комерційна (Enterprise)
Легкість інтеграції з існуючими системами	REST API, вебсервіси, модульна архітектура	Classroom API дозволяє керувати курсами	Підтримка стандартів та відкритого API	Складна інтеграція без партнерських угод
Соціальні функції	Форуми, чати, повідомлення	Коментарі до завдань, стрічка оголошень	Відеоконференції, форуми, запис відео-відповідей.	Дискусійні дошки
Інтерфейс	Інтерфейс часто перевантажений. Високий поріг входження	Мінімалістичний та інтуїтивний дизайн	Сучасний, адаптивний дизайн, зручний додаток	Сприймається як застарілий

Аналіз користувацького досвіду на прикладі провідних платформ, таких як Moodle та Blackboard, виявив цікаві закономірності поведінки здобувачів освіти. Зокрема, користувачі Moodle демонструють вищу залученість, проводячи в системі більше часу, та високо оцінюють можливості двосторонньої взаємодії «викладач-студент», функціонал завантаження робіт та інструменти самоперевірки [21]. Цікавим є той факт, що попри технологічні можливості створення інтерактивного контенту (на базі HTML/Markdown за допомогою інтегрованих редакторів), викладачі та студенти все ще віддають перевагу традиційним форматам файлів, що свідчить про необхідність поступової адаптації педагогічного дизайну до звичок користувачів, а саме:

- документам Microsoft Word (DOC/DOCX) та інші (RTF, ODT);
- презентаціям Microsoft PowerPoint (PPT/PPTX);
- експортованим портативним документам (PDF).

В українському контексті цифровізація набуває особливого значення, виступаючи інструментом подолання не лише епідеміологічних, а й безпекових та географічних бар'єрів частіше пов’язаних із війною та її наслідками для багатьох громад. Незважаючи на постійні виклики, вітчизняні практики швидко інтегрують найкращі міжнародні рекомендації та нововведення для сприяння демократизації освіти, забезпечення інклюзивності та персоналізації навчання [8].

Важливо відзначити, що створення так званих «електронних кабінетів студента» – інтегрованих вебпорталів, що об'єднують розклад, журнал успішності, доступ до бібліотечних фондів та адміністративні послуги – стало стандартом де-факто для провідних українських університетів. Такі системи трансформують студента з пасивного отримувача інформації на активного менеджера власного освітнього процесу.

Проте цифрова трансформація не позбавлена викликів, які потребують системного вирішення:

- інфраструктурна неготовність університетів – необхідність значних інвестицій у серверну архітектуру та захист персональних даних

здобувачів;

- технічна нерівність, причому розрив у доступі до якісного обладнання та швидкісного інтернету меншає, але все ще впливає на студентів чи викладачів, які працюють та навчаються з віддалених районів [8];
- цифрова компетентність – потреба у постійному підвищенні кваліфікації викладачів, які часто стикаються з труднощами при опануванні нових інструментів (таких як стримінгові сервіси для проведення лекцій у реальному часі).

Незважаючи на подолані та все ще актуальні труднощі, цифровізація освітніх процесів перейшла з фази експерименту у фазу сталого розвитку і є невід'ємною ланкою освіти сьогодення в Україні та світі. Успіх цієї трансформації залежить не стільки від вибору конкретної платформи, скільки від комплексної стратегії, що враховує педагогічні цілі, технічні можливості та соціальні аспекти взаємодії учасників освітнього процесу, а грамотна інтеграція й навчання персоналу дозволить продовжувати трансформацію без негативного впливу на ефективність.

1.3. Основні проблеми пошуку знань у великих базах

Стрімка цифровізація сучасного освітнього простору призвела до виникнення парадоксального явища: безпрецедентна доступність інформації не гарантує легкості отримання знань. Університетські репозиторії, електронні бібліотеки, платформи дистанційного навчання та хмарні сховища щороку наповнюються гігабайтами корисних даних. Проте, зі зростанням обсягів цієї інформації, здатність студента швидко знайти потрібну відповідь стрімко знижується. Щоб зрозуміти глибинні причини цієї проблеми, необхідно звернутися до теорії управління знаннями та розглянути природу самої інформації, якою оперують у вищих навчальних закладах.

Ключовим аспектом, який часто випускають з уваги при проектуванні освітніх систем, є поділ знань на дві фундаментально різні категорії: явне та

неявне знання. Явне знання (англ. explicit knowledge) – це інформація, яка вже пройшла процес кодифікації та існує у відчуженій від людини формі [12]. Традиційні університетські інформаційні системи побудовані саме навколо цього типу знань: вони чудово вміють зберігати, сортувати та видавати документи. Однак існує неявне, або так зване «мовчазне» знання (англ. tacit knowledge), яке невіддільне від людини та її досвіду. Воно включає в себе інтуїтивне розуміння предметної області, практичні навички, «лайфхаки» вирішення типових проблем, контекстуальне розуміння та професійне чуття [24]. Особливістю мовчазного знання є те, що його вкрай важко, а іноді й неможливо, повністю викласти на папері чи формалізувати у вигляді алгоритму. Наприклад, студент може прочитати інструкцію із запуску певної програми чи кроками виконання лабораторної роботи з програмування (явне знання), але зіткнутися з нестандартною помилкою (наприклад, неправильні закінчення файлів), вирішення якої знає лише викладач або досвідчений одногрупник, спираючись на свій попередній досвід (неявне знання). Приклади порівняння форм та способів передавання явного та неявного наведено в табл. 1.2.

Проблема сучасних пошукових систем полягає в тому, що вони є «сліпими» до мовчазного знання. Вони індексують файли, але не індексують людей та їхній досвід. Дослідження показують, що обмін таким типом знань критично залежить від соціальних факторів: довіри, особистих зв'язків та культури спільноти, а не лише від наявності цифрового сховища [14]. Оскільки більшість існуючих систем управління знаннями (англ. knowledge management systems) у ВНЗ ігнорують соціальний компонент і фокусуються виключно на документах, величезний масив критично важливої інформації залишається недосяжним для студента [39], що створює ситуацію, коли відповідь на питання існує поруч – у голові колеги чи наставника – але система не дає інструментів, щоб ідентифікувати цього носія знань. Уже стало традиційно, що пошук рішення, яке існує поруч, тепер часто веде в мережу Інтернет, що ізолює обмін досвідом.

Таблиця 1.2

Порівняння явного (explicit) та неявного (tacit) знання

Критерій	**Явне знання**	**Неявне знання**
Сутність	Формалізована, кодифікована інформація	Особистісний, інтуїтивний досвід
Форми	Документи, підручники, методичні вказівки, наукові статті, файли, відео- та аудіозаписи, формули, алгоритми	Ментальні моделі, певний шаблон мислення, переконання, інтуїція, розуміння «на пальцях», практичні навички, власний досвід, майстерність, «лайфхаки»
Передавання	Традиційні та цифрові канали, придатне для масового поширення	Важко формалізується й передається частіше через особисте спілкування, менторство, спільну діяльність
Локалізація	Бібліотеки, репозиторії, LMS, бази даних	Голови викладачів та успішних студентів, неформальні групи

Окрім ігнорування людського фактору, існують суто технічні проблеми навігації навіть серед явного, задокументованого знання. Основним інструментом пошуку в більшості освітніх платформ досі залишається пошук за ключовими словами (англ. keyword search). Метод працює за принципом прямого співставлення символів: якщо користувач вводить слово «програмування на С++», система знайде всі документи, де зустрічається саме цей набір слів/літер у тому чи іншому порядку залежно від конкретної реалізації. Проте такий підхід має суттєві вади в контексті природної мови.

По-перше, пошук за ключовими словами не враховує (не «розуміє») синонімії та контексту окремого запиту. Якщо в проіндексованому документі вжито слово «об’єктно-орієнтоване програмування» або «мови програмування», а студент шукає «написання скриптів», алгоритм цей документ не знайде, хоча він може бути ідеально релевантним. По-друге, виникає проблема багатозначності слів: запит «ключ» може стосуватися як криптографії, так і ключів баз даних або навіть фізичного інструменту. Без

розуміння семантики (змісту) система видасть купу інформаційного «сміття», яке користувачеві доведеться фільтрувати вручну.

У результаті студенти стикаються з ефектом інформаційного перевантаження. Замість того, щоб витрачати час на аналіз матеріалу та навчання, вони змушені витрачати час на механічний перебір файлів, спроби вгадати правильне ключове слово та перевірку десятків посилань. Дослідники пропонують вирішувати цю проблему шляхом впровадження семантичних порталів та інтелектуальних систем пошуку, які використовують метадані та онтології – структуровані описи понять та зв'язків між ними. Такі системи здатні «розуміти» запит користувача на рівні сенсу, а не просто як набір літер, що значно підвищує точність та швидкість знаходження інформації. Без переходу до таких інтелектуальних інструментів ефективність самостійної роботи студентів у цифровому середовищі залишатиметься нижчою за очікувану, незважаючи на постійне зростання обсягів доступних матеріалів.

1.4. Вплив менторингу та комунікації на академічну успішність

У сучасному педагогічному дискурсі поняття «ефективності навчальної діяльності» часто помилково зводиться виключно до фінальних оцінок або інтелектуальних здібностей здобувача освіти. Проте більш прагматичний підхід, орієнтований на процеси, дозволяє стверджувати, що для досягнення академічної успішності виникає гостра потреба у зовнішній підтримці для подолання «тупиків», адже час, витрачений на цьому етапі стає визначальним фактором продуктивності. Доцільно висунути гіпотезу, що для досягнення максимальної ефективності навчання поєднання двох незалежних потоків підтримки – миттєвого доступу до фактологічної інформації та доступу до соціальної експертизи – створює синергетичний ефект, який суттєво перевищує результативність кожного з цих компонентів окремо.

Фундаментом гіпотези слугує кореляція між наявністю підтримки однокурсників (англ. peer-to-peer support) та академічними результатами.

Експериментальні дослідження демонструють помірний позитивний ефект від взаємодії студентів у форматі «рівний рівному», причому цей вплив спостерігається у різних дисциплінах та вікових групах [1]. Важливо зазначити, що пірингова підтримка працює не лише як канал передачі знань, а і як інструмент психологічної адаптації, що своєю чергою мотивує студента не кидати складні завдання на півдорозі [30]. Окрім того, відома цитата Джона Максвела «Ти ніколи не знаєш чогось до кінця, поки не навчиш цього когось іншого» лише підсумовує широко відомий та експериментально підтверджений факт, що менторинг ще й донавчає вчителя [26].

Втім, варто зберігати критичний погляд на цю модель, адже вона не позбавлена обмежень. Ефективність піринговового навчання значною мірою залежить від структурованості взаємодії і хаотичне спілкування може не дати бажаного ефекту. Також слід враховувати, що обмін неявним знанням вимагає певного рівня довіри та сформованої культури спілкування, яку цифрові інструменти можуть полегшити, але не замінити повністю [29].

1.5. Вплив систем генеративного штучного інтелекту на освіту

Стрімкий розвиток технологій та масове впровадження генеративного штучного інтелекту став передумовою появи нового покоління освітніх інструментів, які докорінно змінюють парадигму навчального процесу.

Штучний інтелект (ШІ) – це галузь інформатики, ціллю якої є створення та дослідження роботи інтелектуальних машин, здатних виконувати завдання, що, як прийнято, потребують людського розуму. Генеративний штучний інтелект або породжувальний штучний інтелект (англ. generative artificial intelligence) – це галузь штучного інтелекту, спрямована на створення або поступове генерування нового вмісту в системі покрокового передбачення, як-от: тексту, аудіо, зображень, за допомогою алгоритмів і моделей машинного навчання. Під моделлю тут мається на увазі математична модель, яка описує досліджуваний процес або явище, у цьому випадку –

взаємозв’язки між частинами слів, словами, словосполученнями, та символами в людській мові [5], отримана шляхом використання алгоритмів машинного навчання. Для створення (генерування) переважно текстової інформації використовуються великі мовні моделі (англ. large language models, LLM), що складається з нейронних мереж із мільйонами-мільярдами параметрами, навчених на великій кількості немаркованого тексту.

Якщо традиційні системи управління навчанням, що домінували в освітньому просторі останнє десятиліття, виконували переважно функцію статичних репозиторіїв для зберігання лекцій та адміністрування тестів, то сучасні адаптивні середовища використовують потенціал генеративного ШІ (маються на увазі мовні моделі та інструменти їх впровадження) для створення інтерактивного досвіду. Генеративний ШІ виступає в ролі технологічного каталізатора, що дозволяє перейти від пасивної моделі споживання контенту до активної персоналізованої взаємодії, де система адаптується під потреби здобувача освіти в реальному часі.

Еволюція освітніх технологій рухається в напрямку створення так званих AI-LMS – систем, де студент взаємодіє не просто з файлами, а з інтелектуальними агентами. Дослідники зазначають, що поєднання генеративних моделей з принципами адаптивного навчання (англ. adaptive learning) дозволяє трансформувати роль системи в бібліотекаря чи навіть на розумного тьютора/ментора [43]. Такий підхід базується на теоріях конструктивізму та коннективізму, передбачаючи, що навчання є активним процесом побудови нових зв'язків, а не простою репродукцією інформації [44]. AI-LMS здатні генерувати унікальні навчальні траєкторії, створювати пояснення та вправи «на льоту», забезпечуючи миттєвий зворотний зв'язок, що раніше було можливо лише при індивідуальній роботі з викладачем.

Серед ключових переваг впровадження таких систем варто виділити:

- персоналізацію навчання – здатність алгоритмів аналізувати історію запитів та помилок студента для адаптації складності матеріалу, що є

критично важливим для гетерогенних груп з різним рівнем підготовки;

- оперативність підтримки – генеративні агенти можуть надавати відповіді 24/7, знижуючи когнітивне навантаження на викладачів та адміністративний персонал і дозволяючи студентам отримувати допомогу в момент виникнення питання, а не очікувати наступної можливості задати питання;
- масштабованість – можливість забезпечення якісного освітнього супроводу для великої кількості студентів одночасно при мінімальних витратах людських ресурсів.

Проте, незважаючи на технологічний оптимізм, існуючі кейси впровадження інтелектуальних агентів (наприклад, інструменти Moodle AI чи плагіни на базі ChatGPT) демонструють низку концептуальних недоліків. Найвагомішим з них є ізольованість від соціального контексту та внутрішньої локальної інформаційної області університету. Окрім того, більшість сучасних AI-асистентів функціонують у форматі «один на один» (студент – бот), ігноруючи потенціал академічної спільноти. Студент отримує відповідь від машини, але втрачає можливість соціальної верифікації знань, дискусії з одногрупниками та обміну неявним досвідом, який формується лише у процесі піринговой взаємодії. Така атомізація навчального процесу несе ризик втрати навичок командної роботи та критичного оцінювання інформації, адже істина сприймається як результат генерації алгоритму, а не як продукт колективного обговорення.

Окрім проблеми соціальної відчуженості, серйозним викликом є питання академічної доброчесності та якості контенту. Без впровадження механізмів когнітивної фільтрації та верифікації через авторитетні джерела, використання ШІ може призвести до засвоєння хибних концепцій [46]. До цього також додається доступність інструментів автоматичної генерації текстів створює спокусу для зловживань, що вимагає від університетів розробки нових етичних кодексів та політик, які б регулювали використання ШІ, зміщуючи акцент з заборон на розвиток культури відповідального

використання технологій [9].

Інструменти генеративного ШІ відкривають безпрецедентні новітні можливості для модернізації освіти, проте його ефективна інтеграція неможлива без врахування соціальних та етичних аспектів. Перехід від простої автоматизації до створення повноцінних освітніх екосистем вимагає, щоб ШІ виступав не лише джерелом інформації, а й посередником, що з'єднує студентів, викладачів та знання у єдину мережу, сприяючи розвитку як індивідуальних, так і колективних компетентностей.

1.6. Висновки до розділу

У першому розділі проаналізовано шлях освіти до первинної цифровізації, який включав розгортання систем управління навчанням, електронних репозиторіїв та хмарних сервісів, що стало загальноприйнятим стандартом у світі для забезпечення безперервності навчального процесу навіть у кризових умовах. Однак, попри технологічне насичення, сучасна цифрова екосистема університетських сервісів в Україні залишається фрагментованою та не збалансованою, в тому числі й через умови її форсованого розвитку, що стримує подальше зростання ефективності навчання. Можна виділити три ключові проблеми, які є наслідками цього:

1. Розрив між інформацією та комунікацією. З одного боку існують LMS, які чудово справляються зі передаванням академічних артефактів й адмініструванням курсів. З іншого боку – існують месенджери та соціальні мережі, де відбувається живе спілкування студентів.

2. Проблема пошуку «мовчазного знання». Традиційні пошукові механізми ефективно виконують пошук за ключовими словами, страждають, коли сенс захований «між рядками», але абсолютно безсилі, коли потрібно знайти носія досвіду – людину, яка може пояснити складний матеріал, через що цінна експертиза залишається прихованою в головах окремих студентів чи викладачів і не стає надбанням спільноти через відсутність інструментів

сприяння її поширення.

3. Ізольованість штучного інтелекту. Поява генеративного ШІ створила потужний інструмент для індивідуальної допомоги, але більшість існуючих реалізацій працюють у вакуумі. Студент спілкується з ізольованим від університетського інформаційного середовища чат-ботом сам-на-сам, втрачаючи можливість перевірити здобуті знання в дискусії з одногрупниками.

Таким чином, головним викликом сьогодення є не створення ще однієї бази даних чи ще одного чат-боту, а інтеграція цих компонентів у єдину систему. Ефективність навчальної діяльності студента прямо залежить від швидкості подолання «когнітивних тупиків» і ця швидкість досягається лише тоді, коли доступ до фактичної інформації (артефактів) і доступ до соціальної допомоги (менторів) знаходяться на відстані одного кліку.

Саме підхід, що поєднує ці «два світи» дозволить перейти від простої цифровізації контенту до цифровізації освітнього середовища, де технології працюють на підсилення людського капіталу та створення піринговых зв’язків між учасниками навчального процесу. Детальний аналіз шляхів реалізації такої системи та вибір відповідних технологій буде проведено у наступних розділах роботи.

РОЗДІЛ 2.
АНАЛІЗ ПЕРСПЕКТИВНИХ НАПРЯМІВ ДОСЛІДЖЕННЯ

2.1. Наслідки цифровізації освіти та перспективи покращення

Пост-ковідний освітній ландшафт характеризується феноменом, який у бізнесі називають «клаптиковою цифровізацією» (англ. patchwork digitalization). Екстрений перехід в онлайн змусив університети впроваджувати рішення ситуативно, що призвело до накопичення великої кількості розрізнених інструментів. Сьогодні середньостатистичний студент змушений оперувати в середовищі, де критично важливі дані розпорошені по ізольованих платформах.

Типову інфраструктуру сучасного ВНЗ можна зобразити як набір джерел інформації, які не обмінюються даними між собою. Студент виступає єдиним інтегратором, вручну переносячи інформацію з однієї системи в іншу (рис. 2.1). Важливо зазначити, що така система працює ефективно до тих пір, поки студент розуміє, яка інформація де знаходиться й не витрачає багато часу на орієнтування, що аналогічно й для викладачів, які багато років працюють у такому середовищі. Якщо ж мова ітиме про першо-другокурсників, це гроно інформаційних ресурсів призводить до «цифрової втоми».

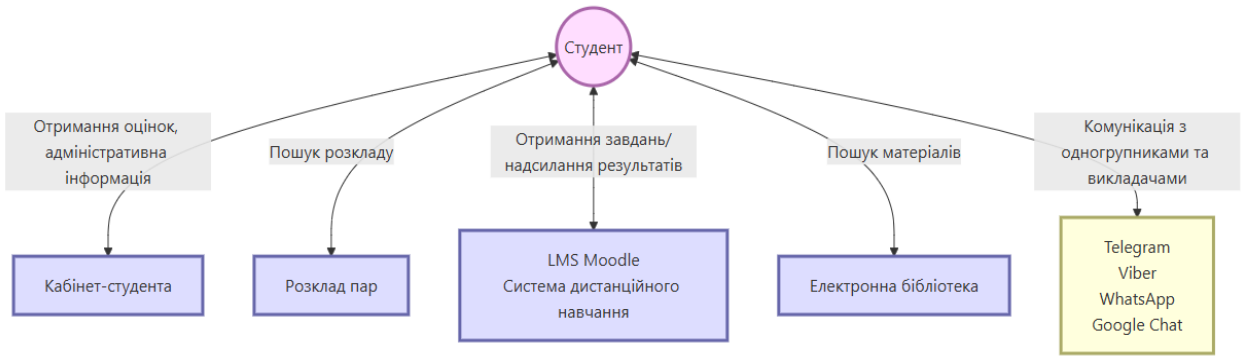


Рис. 2.1. Приклад взаємодії студента із системами та середовищами для отримання та опрацювання інформації з різних джерел з метою виконання поставлених академічних завдань

Концепція «Єдиного вікна» (англ. single window) – перспективний напрям вирішення цієї архітектурної кризи є перехід від множини інтерфейсів до єдиної точки входу (англ. single point of entry). У рамках даної роботи пропонується концепція інтелектуального агента, що діє за принципом «шлюзу» за аналогію з теперішньою медичною системою:

1) коли у пацієнта виникає проблема, він йде до сімейного лікаря;

2) лікар вирішує прості питання самостійно (виписує довідку, дає пораду);

3) якщо потрібно зробити конкретні аналізи – сімейний лікар виписує направлення з необхідною інформацією;

4) якщо потрібна експертиза в конкретній області, лікар направляє до вузькопрофільного спеціаліста, передаючи історію хвороби;

5) пацієнту не потрібно шукати кабінети самому – його веде лікар.

У технічному сенсі це означає створення надбудови, яка агрегує дані через різноманітні інтерфейси програмування застосунків (англ. application programming interface, API), але не замінює спеціалізовані системи повністю, оскільки для задоволення деяких потреб іноді достатньо простих запитів, а не складних маніпуляцій даними й керування системами (наприклад, в системі розкладу отримати його на сьогодні – це тривіальна й часто використовувана функція, в той час як перепланувати пари – значно рідше). Окрім того, надмірна централізація створює ризики, пов’язані з точковими збоями: якщо система «шлюзу» виходить з ладу, користувач тимчасово втрачає доступ до всіх пов’язаних сервісів – саме тому важливо будувати подібні системи за принципом архітектурного патерну проєктування «Фасад» [10].

Загальна тенденція розвитку освітніх екосистем свідчить про рух у напрямі інтегрованих рішень. Міжнародний досвід указує, що системи з єдиною точкою входу підвищують задоволеність користувачів, зменшують кількість технічних звернень, скорочують час виконання організаційних завдань і формують більш стійку логіку взаємодії між інституційними сервісами. Для українських університетів, які сьогодні перебувають у фазі

активної модернізації, інтеграційні підходи можуть стати ефективним інструментом переходу від постпандемічної та воєнної цифровізації до системної, керованої та структурно послідовної цифрової трансформації.

2.2. Системи на основі генеративного штучного інтелекту в освіті

Поява генеративного штучного інтелекту та великих мовних моделей змінила парадигму взаємодії людини з комп'ютером. Якщо попередні покоління програмного забезпечення вимагали від користувача вивчення спеціальних команд або інтерфейсів, то LLM дозволяють керувати обчислювальними системами за допомогою природної мови. Для багатьох систем, орієнтованих на опрацювання даних, це означає перехід від стандартизованої взаємодії до адаптивного користувацького досвіду, однак ефективне використання цих технологій вимагає глибокого розуміння не лише їхніх переваг, а й архітектурних обмежень.

В основі сучасних інтелектуальних систем на основі генеративного ШІ лежать мовні моделі (наприклад, GPT-5, Claude 4, Llama 3 тощо), натреновані на колосальних масивах текстових даних. У контексті вищої освіти їхній потенціал розкривається у двох напрямках:

1. Персоналізація навчання за рахунок адаптації стилю викладу контенту під когнітивний профіль студента. Тут варто згадати метод, який набув великого поширення завдяки GenAI – ELI5 (англ. «Explain Like I’m 5» – поясни, ніби мені 5 років), узятого з однойменного форуму, де користувачі мережі одне одному пояснюють теми, питання, у яких часто виникають в буденності, але які також вимагають експертизи в певній області [18]. Суть полягає в тому, що мовна модель для, наприклад, студента-філолога може пояснити лінійну регресію через лінгвістичні аналогії, чим сильно знижує бар'єр входження у міждисциплінарні предмети.

2. Автоматизація рутинних запитів, адже значна частина роботи викладачів та деканату припадає на відповіді на питання «Де знайти

викладача?», «Коли починається сесія?», «Де взяти реквізити для оплати за навчання?», «Які вимоги для поселення в гуртожиток?». Чат-бот на основі LLM здатен автоматизувати цю комунікацію, звільняючи людський ресурс для цікавіших/складніших завдань.

Попри потужність, базові моделі загального призначення мають дві критичні вади для академічного середовища, а саме:

1) галюцинації у цьому контексті – це генерування великою мовною моделлю переконливого тексту, що містить неправдиві факти, описи подій, що ніколи не відбувалися, імена неіснуючих людей тощо. Така поведінка є наслідком природи мовних моделей – машин, які передбачають наступне слово (токен, символ – залежно від реалізації) [2];

2) дата відсікання знань (knowledge cutoff) – дата, по яку зібрано тренувальні дані для моделі. Відповідно, знання моделі обмежені цією датою. Наприклад, якщо модель Llama 4 натренована у 2024 році, вона фізично не може «знати» про нові вимоги до курсових робіт, затверджені вчора на кафедрі й буде видавати застарілу інформацію під виглядом актуальної [25].

Першим рівнем покращення якості відповідей є не зміна коду, а зміна способу взаємодії з моделлю. Одним із найпростіших способів досягнення цього є конструювання (інженерія) контекстних підказок (англ. prompt engineering) – набір технік задання початкового контексту для готової моделі для спрямування генерації в потрібне русло [4]. Використання технік на кшталт надання прикладів у запиті (так званий прийом few-shot prompting) дозволяє задати моделі формат відповіді та тон.

Іншим засобом є використання спеціалізованих моделей з інкорпорованими можливостями Chain-of-Thought (ланцюг думки) та Reasoning Models (міркуючі моделі). Більшість сучасних моделей (наприклад, OpenAI o1, GPT OSS, DeepSeek R1, Qwen3, Magistral) використовують прихований процес «мислення» перед генерацією відповіді. Система розбиває складну задачу на послідовність логічних кроків, перевіряє сама себе і лише потім видає результат, що призводить до суттєвого зниження кількості

логічних галюцинацій при розв'язанні задач чи побудові алгоритмів, хоча й не вирішує проблему відсутності актуальних даних [33].

Переломним моментом в еволюції ШІ-асистентів стало введення технології використання інструментів (Tool Use/Function Calling), за якого модель навчається розпізнавати ситуації, коли їй потрібна зовнішня допомога, і формувати запит до відповідного програмного модуля (API). Тут в тому числі можна використовувати запити до баз даних, вебпошук новин тощо для отримання актуальної інформації за запитом користувача [20]. Важливо зазначити, що сама модель не виконує жодного коду, а виконання покладається на проксі-модуль між зовнішньою системою, моделлю та користувачем (рис. 2.2).

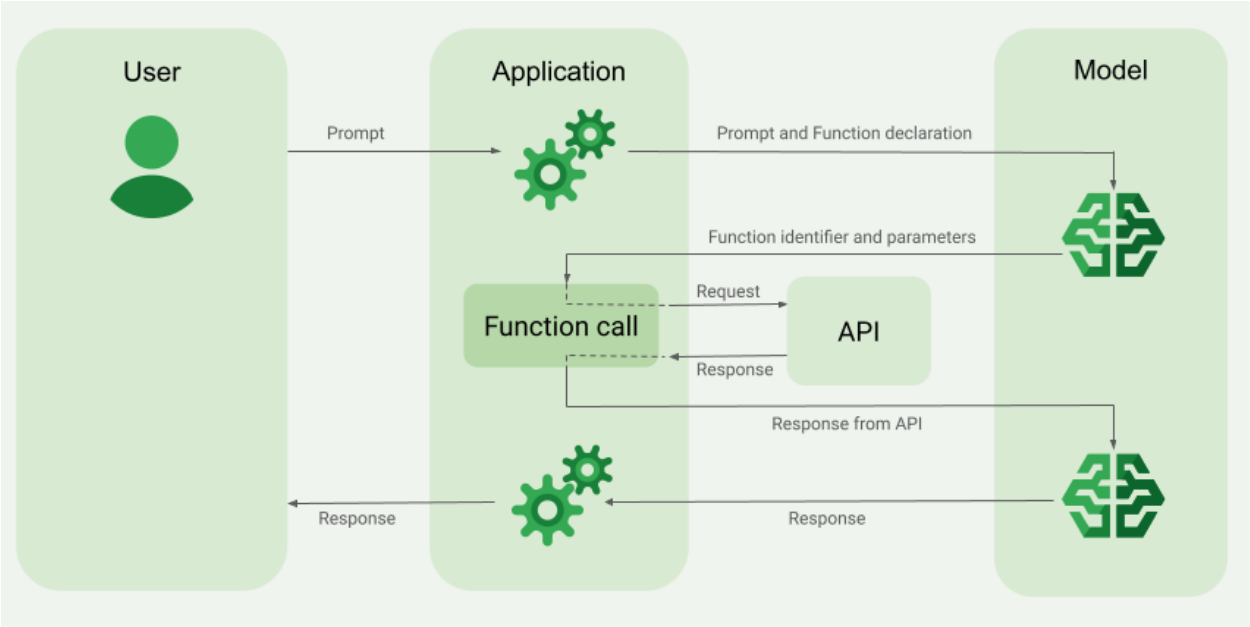


Рис. 2.2. Взаємодія моделі з інструментом (функцією) відбувається на стороні застосунку, а модель лише отримує результат на подальше опрацювання

Зі зростанням кількості інструментів виникла проблема їх інтеграції, адже кожен сервіс вимагав написання унікального коду для з'єднання з ШІ. Model Context Protocol (MCP) – це новітній відкритий стандарт, який уніфікує спосіб підключення джерел даних та інструментів (функцій) до LLM. Він діє за принципом «USB-порту» для штучного інтелекту і:

- дозволяє один раз написати з’єднувач, і використовувати його з різними клієнтами;
- забезпечує безпечний доступ до даних, контролюючи, що саме модель може читати [38].

Функціонально різниці між Tool Use та Model Context Protocol (MCP) немає, але останній значно спрощує підключення до LLM за рахунок один раз написаної інтеграції. Візуалізація різниці інтеграції використання інструментів та MCP зображена на рисунку 2.3. Недоліком є те, що для роботи MCP-протоколу з’єднувач повинен бути запущеним у режимі сервера, що споживає незначний додатковий ресурс.

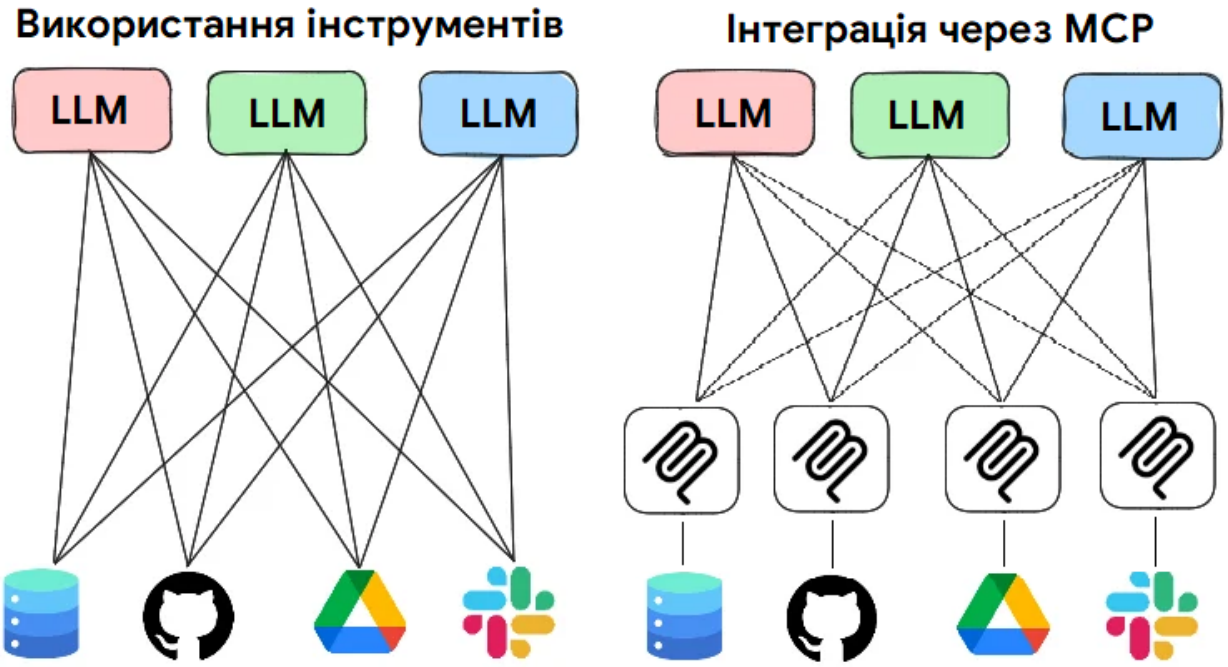


Рис. 2.3. Порівняння інтеграції різних сервісів за допомогою прямої взаємодії з API та через Model Context Protocol

Отже, хоч сучасні LLM володіють потужними когнітивними здібностями, використання їх у «чистому» вигляді в університетському середовищі є неефективним і навіть ризикованим через галюцинації та застарівання знань. Технології Tool Use та стандарти типу MCP вирішують проблему доступу до функцій, але залишається відкритим питання ефективної

роботи з великими масивами неструктурованих текстових знань. Для вирішення цього питання існує архітектурний компонент RAG, який буде розглянуто далі.

2.3. Генерування з доповненням через пошук як адаптер до знань

Фундаментальною проблемою класичних інформаційно-пошукових систем є так званий «лексичний розрив» (англ. lexical gap) – явище виникає, коли запит користувача та шуканий документ описують одну й ту саму концепцію, але використовують різні слова (наприклад: «іспит», «колоквіум», «екзамен», «контрольна робота», «тест» та «контроль знань»). Для подолання цього обмеження в сучасних системах штучного інтелекту застосовується метод векторизації тексту, який дозволяє оперувати не символами, а векторним представленням сенсу терміну.

Процес перетворення тексту в числовий вигляд називається вкладанням слів або частіше створенням «ембеддингів» (англ. embeddings). Під ембеддингом мається на увазі вектор v у n-вимірному просторі дійсних чисел, який відображає семантичне значення вхідного тексту (слова, речення або абзацу).

Формально функцію E перетворення вхідного тексту в ембеддинг можна записати як відображення простору слів W (словник) у векторний простір V ($\mathbb{R}^n$):

$$E: W \to V \tag{2.1}$$

$$E(\text{слово}) = [x_1, x_2, x_3, \dots, x_n] \tag{2.2}$$

Кожен вимір отриманого вектору неявно відповідає певній абстрактній ознаці (наприклад, «академічність», «терміновість», «позитивна тональність», «належність до точних наук», «тваринoподібність» тощо) в семантичному просторі обраної попередньо натренованої моделі (рис. 2.4).

Рис. 2.4. Графічне представлення слів у 2-вимірному просторі з ознаками «природність» та «небезпечність»

Ключова властивість такого простору ознак полягає в тому, що семантично близькі поняття розташовуються геометрично поруч. Наприклад, вектори слів «ключі» та «дверний замок» будуть мати меншу відстань між собою, ніж вектори «дверний замок» та «Острозький замок».

Сучасні моделі, такі як `text-embedding-3-small` від OpenAI, що використовує аналогічні мовні перетворення, як і модель GPT-5, перетворюють букви, частини слова, слова, речення чи абзаци тексту (до ліміту самої ембеддинг-моделі, що часто становить 2048-4096 символів тексту в кодуванні Unicode UTF-8) на вектори високої розмірності (наприклад, сьогодні $n = 384; 768; 1536$). Приклад отриманого вектору з кількістю вимірів $n = 768$ від однієї із таких моделей зображено рисунку 2.5.

Маючи векторне представлення слів можна виконувати математичні операції над словами у векторних просторах, наприклад, операції додавання та віднімання ембеддингів слів будуть давати вектори, подібні до розв’язання певного «ребусу» (варто пам’ятати, що операції виконуються з деякою похибкою), наприклад: $\overrightarrow{\text{Париж}} - \overrightarrow{\text{Франція}} + \overrightarrow{\text{Італія}} = \vec{v} \approx \overrightarrow{\text{Рим}}$.

Що найбільш важливо в контексті цієї магістерської роботи, над ембеддингами можна виконувати операції порівняння векторів. Поширеними метриками подібності векторів є Евклідова відстань, скалярний добуток векторів та косинус подібності векторів.

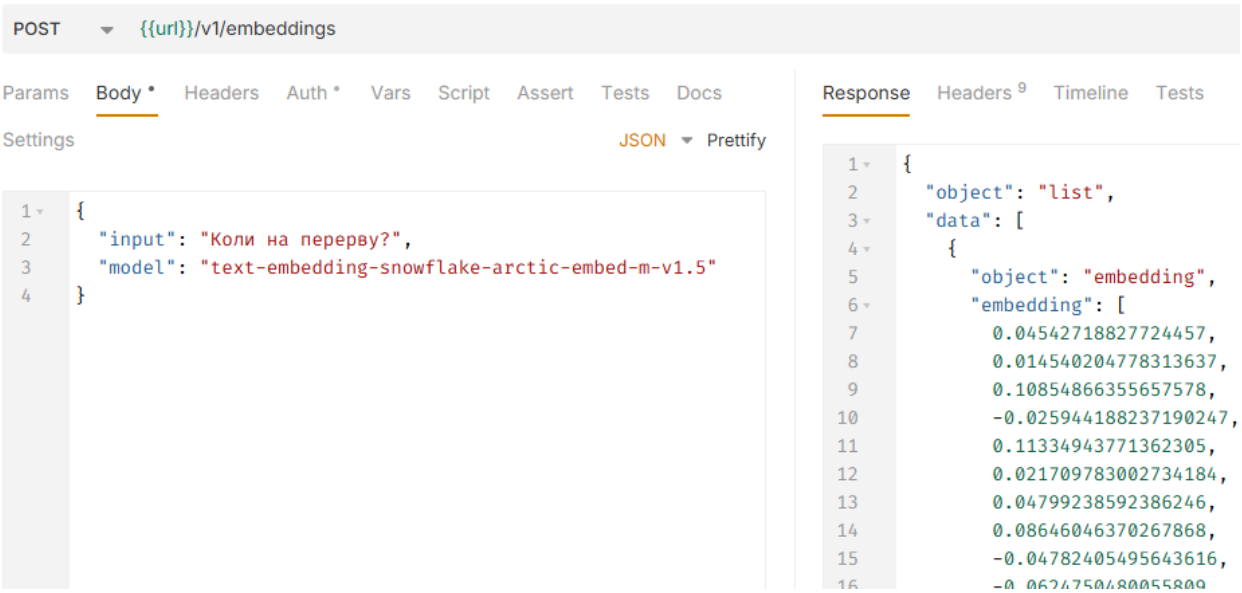


Рис. 2.5. Приклад отримання ембеддингів для невеликої репліки за допомогою OpenAI API запиту та відповідь моделі у вигляді 768-вимірного вектору

Евклідова відстань (також згадується як Евклідова метрика) – метрика, що показує довжину найкоротшого шляху між двома точками і також застосовується з векторами [16]. Формула обчислення Евклідової відстані для n-вимірних векторів p та q може бути представлена наступним чином:

$$|\vec{p} - \vec{q}| = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \quad (2.3)$$

Скалярний добуток векторів – метрика, яка є рівною добутку довжин цих векторів на косинус кута між ними й обчислюється за формулою:

$$\vec{p} \cdot \vec{q} = |\vec{p}| \cdot |\vec{q}| \cdot \cos \angle(\vec{p}, \vec{q}) = \sqrt{\sum_{i=1}^{n} p_i^2} \cdot \sqrt{\sum_{i=1}^{n} q_i^2} \cdot \cos \angle(\vec{p}, \vec{q}) \quad (2.4)$$

І хоч обидві метрики певним чином показують відстань між векторами, але їх використання в порівнянні ембеддингів непоширене. Евклідова відстань у високих вимірах стає менш інформативною, оскільки всі вектори віддаляються один від одного приблизно однаково. Скалярний добуток також залежить від довжин і може давати завищені значення для великих за нормою векторів, хоча їх напрям може бути менш подібним.

Косинус подібності векторів – це метрика, що використовує косинус кута між векторами, тобто скалярний добуток векторів, поділений на добуток довжин:

$$S_c(\vec{p},\vec{q}) = \cos\angle(\vec{p},\vec{q}) = \frac{\vec{p}\,\vec{q}}{|\vec{p}|\cdot|\vec{q}|} = \frac{\sum_{i=1}^{n} p_i q_i}{\sqrt{\sum_{i=1}^{n} p_i^2}\cdot\sqrt{\sum_{i=1}^{n} q_i^2}} \qquad (2.5)$$

Із формули (2.6) випливає, що область значень функції є завжди в межах інтервалу $[-1;+1]$, причому:

- значення, близькі до 1 означають повну ідентичність змісту (вектори колінеарні);
- наближення 0 вказує на ортогональність векторів, тобто повну відсутність семантичного зв'язку (теми абсолютно різні);
- близькість до -1 означає протилежність змісту (антоніми).

Через такі особливості косинус подібності векторів частіше використовується для порівняння ембеддингів, тому що метрика нормалізує довжину, фактично порівнюючи кут між векторами. У задачах текстового пошуку, рекомендаційних системах та кластеризації вона краще корелює зі змістовною близькістю об’єктів, проте коли масштаб несе важливу інформацію (наприклад, у фізичних моделях чи моделях геноміки), Евклідова відстань або скалярний добуток можуть бути доречнішими.

Генерація з розширеним пошуком (англ. retrieval-augmented generation, RAG) – методика, яку вперше формалізовано дослідниками Facebook AI

Research як метод поєднання параметричної пам'яті (знань, «зашитих» у вагах моделі) та непараметричної пам'яті (зовнішньої бази знань), чим зменшує необхідність перенавчання LLM на нових даних, що дозволяє заощадити на обчислювальних і фінансових витратах [42]. Додавання зовнішніх знань моделі також виступає одним із механізмів «заземлення» (англ. model grounding) – процесом, що прив'язує відповіді генеративної моделі до перевірених фактів, мінімізуючи ризик галюцинацій [23].

Реалізація RAG-системи не є монолітною, а складається з послідовності взаємопов'язаних етапів. Архітектуру можна розділити на дві фази. Фаза індексації складається з етапів підготовки початкової бази знань:

1) Завантаження сирих даних (документи, таблиці, сторінки HTML) – усього, що вважається за потрібне додавати до бази знань моделі;

2) Розбиття тексту на логічні фрагменти (англ. chunks), адже занадто малий чанк втрачає контекст, а занадто великий – зашумлює вектор;

3) Перетворення чанків у вектори та збереження їх у векторній БД.

Фаза виконання – кроки, необхідні для вбудовування знань із БД:

1) Система перетворює запит користувача на вектор

2) Система знаходить k (часто в межах від 5 до 10) найбільш релевантних чанків у базі.

3) Знайдені фрагменти додаються до системного промпту як контекст для надання відповіді з усіма супутніми інструкціями.

4) LLM отримує запит разом із контекстом і генерує відповідь, спираючись включно на надану інформацію.

Узагальнену схему наведено на рисунку 2.6.

Важливим архітектурним кроком у будь-якій системі індексації є етап підтримки бази знань в актуальному стані. Векторна база знань непридатна для редагування й часто реалізовується як тіньова копія основної, особливо якщо мова йде про дані різної природи (файли, записи, документи, сторінки, посилання). Для підтримки актуальності бази використовуються механізми фонової індексації при внесенні змін до головних баз.

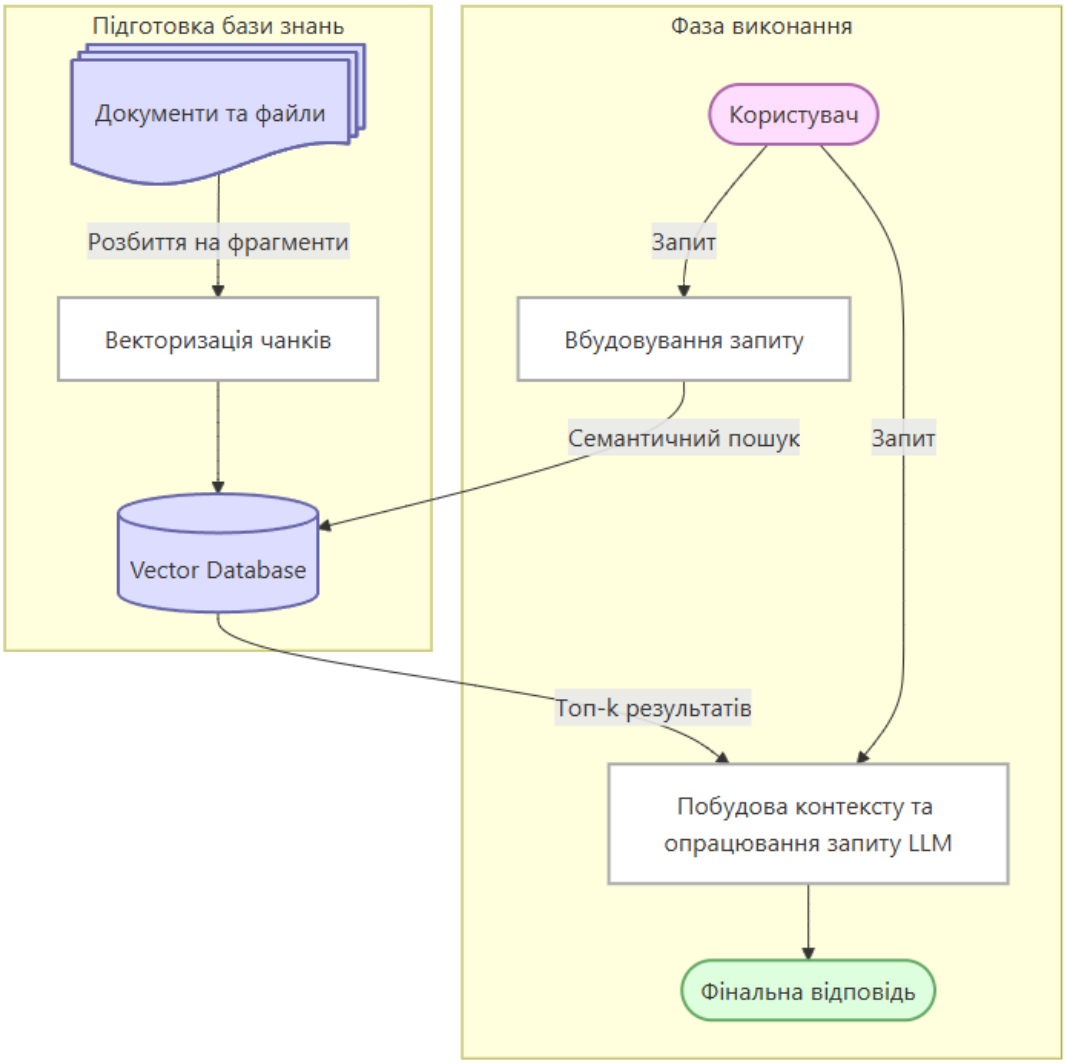


Рис. 2.6. Узагальнена схема роботи системи, що забезпечує генерацію відповіді моделлю з доповненням через пошук по підготовленій базі знань

Важливо запримітити, що при проектуванні інтелектуальних систем іноді виникає питання, чи не краще довчати модель на даних університету замість реалізації складних RAG-систем. Відповідь на це лежить у особливостях файн-тюнінгу (англ. fine-tuning) – процесу донавчання нейромережі на специфічному наборі даних для корекції ваг. Порівняння цих особливостей, переваг та недоліків обох методів наведено в табл. 2.1.

Для завдань даної магістерської роботи, де дані є динамічними (наприклад: файли, які завантажуються на LMS, адміністративна інформація, що змінюється кілька раз протягом семестру), підхід RAG демонструє ряд переваг перед донавчанням моделі.

Таблиця 2.1

Коротка порівняльна характеристика fine-tuning та RAG підходів донавчання

Критерій	**Донавчання**	**Пошукова генерація**
Сутність знань	Зберігаються у вагах моделі	Беруться із динамічного зовнішнього сховища
Оновлення	Складне та дороге. Потребує повторного циклу тренування моделі	Достатньо додати/замінити файл у базі даних за умови робочої індексації
Боротьба з галюцинаціями	Модель все одно може вигадувати факти, змішуючи їх з тренувальними даними	Відповідь базується на цитуванні наданого контексту
Прозорість	Неможливо точно відстежити джерело помилок	Система може надавати посилання на конкретні документи (цитування)
Вартість впровадження	Ручне й дороге донавчання	Вартість додаткової бази + фонової синхронізації

Попри очевидні переваги, базова реалізація RAG стикається з низкою специфічних проблем при роботі з гетерогенними даними. Основна проблема полягає у втраті контексту при розбитті тексту на фрагменти. Її часто долають завдяки реалізації гібридного пошуку, який поєднує пошук за ключовими словами для точного знаходження специфічних даних (термінів, прізвищ, кодів), а також семантичний пошук для розуміння суті запиту. Результати обох алгоритмів об'єднуються за допомогою методу переранжування (англ. reciprocal rank fusion, RRF), де використовується спеціалізована модель-реранкер, яка ставить точну оцінку релевантності [11]. Хоча цей процес повільніший, він критично підвищує точність фінальної відповіді, але часто не реалізується через складність архітектури та тривалість опрацювання запиту.

Отже, технологія RAG є оптимальним архітектурним рішенням для інтеграції динамічних університетських знань у генеративні моделі, однак «наївна» імплементація не здатна забезпечити високу точність у середовищі зі схожими документами. Необхідний пошук рішень на заміну гібридному

пошуку для побудови легкої в реалізації, але надійної системи інтелектуальної підтримки студентів.

2.4. Соціальний нетворкінг для стимуляції пірингового навчання

У теорії управління знаннями існує фундаментальна задача, відома як «локалізація експертизи» (англ. expertise location). Вона полягає у складності ідентифікації носіїв специфічних знань усередині великої організації [17]. У контексті університету можна розглянути приклад, що для будь-якої складної теми існує студент, який вже опанував цей матеріал на високому рівні, однак для здобувача, який зіткнувся з проблемою, цей «експерт» залишається невидимим.

Ефективність пірингового навчання (англ. peer learning) – процесу взаємного навчання студентів – прямо залежить від здатності навчальної екосистеми формувати транзактивну пам'ять – стан групи осіб, коли члени групи знають не лише предмет, а й те, хто ще знає цей предмет, тобто знають можливі джерела отримання знань [36]. У більшості ВНЗ транзактивна пам'ять розвинена слабко через атомізацію навчального процесу та розмитість соціальних зв'язків у великих академічних групах чи на великих потоках. Незважаючи на це, студенти намагаються використовувати кілька інтуїтивних стратегій для пошуку допомоги, кожна з яких має обмеження, що знижують загальну ефективність системи:

1. Використання «сильних зв'язків» – звернення до найближчого кола друзів (одногрупників, з якими склалися дружні стосунки), що ґрунтується на високому рівні довіри. Найбільшим недоліком є невідповідність рівня цього кола необхідному, адже, за ствердженням досліджень, здобувачі освіти частіше формують зв’язки з іншими здобувачами свого рівня, що робить цю стратегію малоефективною [40].

2. Альтернативою є стратегія широкомовних запитів у загальні чати чи групи, що дозволяє залучити «слабкі зв'язки» та охопити широку аудиторію

потенційних експертів. Проте на практиці цей канал страждає від феномену «дифузії відповідальності», коли кожен учасник чату припускає, що на запитання відповість хтось інший, що часто залишає запит без реакції. Додатковим стримуючим фактором виступає соціальний страх – побоювання публічно продемонструвати своє незнання перед великою аудиторією, а також високий рівень інформаційного шуму, в якому губляться конструктивні запити.

3. Третім шляхом є звернення безпосередньо до викладача і цей метод гарантує найвищу якість та достовірність отриманої інформації, проте він є найменш популярним. Викладач фізично не може забезпечити миттєвий зворотний зв'язок для сотень студентів. Крім того, існує суттєвий психологічний бар'єр: студенти часто відчувають «загрозу» при зверненні до викладачів – особливо в традиційному класному форматі, що знижує ймовірність, що вони скористаються офіційним каналом [31].

Виникає потреба створення спеціалізованого інструменту пошуку партнера для навчання (англ. study buddy), щоб такий вибір базувався не на особистих симпатіях, а на компетентності в тій чи іншій темі. Перспективним напрямом подолання розриву між потребою в допомозі та наявною експертизою є впровадження систем алгоритмічної оркестрації соціальних зв'язків. В основі такого підходу лежить ідея використання штучного інтелекту не як генератора контенту чи семантичний пошук, як у випадку з RAG (див. п. 2.3), а як інтелектуального посередника, що аналізує дані про успішність студентів для створення тимчасових навчальних пар.

Підґрунтям для реалізації такої системи є те, що сучасні LMS накопичують колосальні обсяги даних про навчальну активність, формуючи академічний цифровий слід кожного студента із даних про:

- історію оцінок за конкретні модулі та лабораторні роботи;
- час, витрачений на виконання завдань;
- активність на форумах та кількість переглядів матеріалів.

Традиційно ці дані використовуються лише для звітності, але з точки

зору нетворкінгу, вони є мапою компетенцій, яку можна використати в тому числі для пошуку допомоги. Наприклад, якщо система аналізує групу студентів й виділяє студента А, який отримав 95 балів за модуль «Багатопотоковість в Java» і здав його на тиждень раніше дедлайну, вона може з високою ймовірністю кваліфікувати його як локального експерта з цієї теми. Окрім того, персональна академічна інформація студентів залишається в «чорній скриньці» системи, не порушуючи питань конфіденційності. Таким чином, використання систем на основі ШІ для аналізу академічного сліду відкриває шлях до створення саморегульованих навчальних спільнот, де обмін знаннями відбувається цілеспрямовано та ефективно.

2.5. Висновки до розділу

У другому розділі проведено комплексний аналіз перспективних напрямів розвитку освітніх технологій й ідентифіковано ключові архітектурні та методологічні прогалини в існуючих системах. Дослідження підтвердило, що просте нарощування функціоналу LMS або впровадження ізольованих чат-ботів не вирішує фундаментальної проблеми сучасної освіти – розриву між статичними знаннями та динамічною соціальною взаємодією.

Узагальнюючи результати аналізу, можна сформулювати наступні ключові положення, які ляжуть в основу проєктування системи:

1. Необхідність подолання фрагментації цифрового освітнього простору. Перспективним вирішенням є створення системи типу «єдиного вікна» на базі інтелектуального агента, який виступає адаптером до різних систем, оперуючи даними із різних джерел (розклад, оцінки, бібліотека) через уніфікований розмовний інтерфейс.

2. Аналіз обмежень великих мовних моделей довів, що для університетського середовища використання LLM загального призначення є малоефективним. Оптимальним підходом є використання архітектури RAG, яка забезпечує «заземлення» генеративного ШІ на актуальні та достовірні

документи університету, дозволяючи працювати з динамічними даними без необхідності дороговартісного постійного донавчання.

3. Описано проблему «локалізації експертизи», через яку студенти часто не можуть знайти допомогу, хоча в їхньому оточенні є компетентні одногрупники. Запропоновано концепцію використання системи на базі генеративного штучного інтелекту не лише як довідника, а як соціального посередника. Аналізуючи академічний цифровий слід, система здатна пропонувати навчальні пари, долаючи соціальні бар'єри та ефект дифузії відповідальності.

Таким чином, наукова задача даної магістерської роботи – розробка гібридної інтелектуальної екосистеми, що має базуватися на мікросервісній архітектурі, поєднувати модулі семантичного пошуку по документах та соціального матчингу студентів. Обґрунтування вибору конкретних технологій, мов програмування та протоколів взаємодії для реалізації цієї концепції буде наведено у наступному розділі.

РОЗДІЛ 3.
ВИБІР ТЕХНОЛОГІЙ ТА ОБҐРУНТУВАННЯ АРХІТЕКТУРИ ДЕМОНСТРАЦІЙНОЇ СИСТЕМИ

3.1. Опис функціональних та нефункціональних вимог до програмної реалізації системи

Ґрунтуючись на результатах аналізу, проведеного у попередніх розділах, ключовими проблемами сучасної цифрової освіти визначено фрагментацію інформаційного простору та розрив між статичними навчальними матеріалами і динамічною соціальною взаємодією. Існуючі системи ефективно зберігають дані, але не забезпечують оперативної навігації та не сприяють горизонтальному обміну досвідом між студентами.

Оскільки метою роботи є розробка та програмна реалізація демонстраційного варіанту гібридної інтелектуальної системи, яка об'єднує функції семантичного пошуку по базі знань університету та сервіс для пошуку менторів для мінімізації часу подолання студентами когнітивних бар'єрів, то для досягнення поставленої мети необхідно вирішити такі науково-практичні задачі:

1) розробити архітектуру «єдиного вікна», що забезпечує уніфікований доступ до розрізнених джерел даних;

2) створити підсистему семантичного пошуку на основі RAG та застосувати методику покращення релевантності результатів пошуку та контекстуальної обізнаності моделей;

3) реалізувати соціальний модуль, який на основі аналізу академічного цифрового сліду студентів формує рекомендації щодо партнерів для навчання;

4) провести експериментальну перевірку роботоздатності системи.

Враховуючи гібридну архітектуру, функціональні вимоги доцільно розділити на три блоки, що відповідають трьом основним складовим системи

– вимоги до користувацького інтерфейсу взаємодії та основної великої мовної моделі, вимоги до підсистеми пошуку артефактів і вимоги до підсистеми стимулювання менторингу.

Оскільки система позиціонується як «єдине вікно», слід розпочати з вимог до інтерфейсу та інтелектуального модуля.

1. Клієнтська частина системи реалізується як вебзастосунок (Single Page Application), що має забезпечувати інтуїтивно зрозумілу взаємодію користувача з бекенд-частиною. Основним елементом взаємодії має бути чат-інтерфейс, аналогічний сучасним LLM-клієнтам, щоб мінімізувати поріг входження враховуючи поширення такого інтерфейсу сьогодні (пошта, месенджери, чати).

2. Рендеринг. Інтерфейс повинен підтримувати відображення не лише тексту, а й Markdown-розмітки для коректного відображення технічної інформації.

3. Адаптивність. Враховуючи патерни поведінки студентів, інтерфейс повинен коректно відображатися та функціонувати на мобільних пристроях, зберігаючи доступ до всіх функцій системи.

4. Індикація стану системи. Користувач повинен отримувати візуальний зворотний зв'язок про процеси, що відбуваються на сервері (індикатори «опрацьовую запит» при генерації відповіді, «користуюся інструментом» при запиті до бази знань тощо).

5. Контекстна обізнаність – здатність системи підтримувати «діалог», пам'ятаючи попередні репліки користувача для уточнення запитів.

Велика мовна модель у даній системі виконує роль не просто генератора тексту, а інтелектуального оркестратора, що приймає рішення про виклик відповідних задачі інструментів. Вимоги до LLM можна викласти наступним чином:

1) підтримка виклику функцій (Tool Use) – критична вимога, оскільки модель повинна вміти розпізнавати наміри користувача та замість генерації тексту повертати структурований JSON-запит для виклику зовнішніх

API (пошук у векторній БД або запит до соціального сервісу);

2) підтримка режиму міркування для обдуманого використання інструментів та маніпуляції даними;

3) контекстне вікно великого розміру для забезпечення можливості обробки великого обсягу вхідних даних (мінімум 16 тис. токенів, оптимально 128 тис. токенів), що важливо при роботі з пошуком та для ведення діалогу без втрати «нитки розмови»;

4) якісна підтримка української мови – здатність коректно розуміти специфічну університетську термінологію та генерувати природні відповіді українською мовою без синтаксичних кальок з англійської чи перемішування токенів;

5) керованість – здатність моделі чітко слідувати системному запиту (system prompt), зокрема дотримуватися заданого тону спілкування, ігнорувати спроби злому (jailbreaking) та відмовлятися відповідати на питання, що виходять за межі академічного контексту;

6) баланс швидкості та вартості – модель має забезпечувати прийнятну затримку (time to first token) при помірній вартості, що є важливим для масштабування системи.

Підсистема роботи зі знаннями (модуль пошуку на базі технології RAG) має низку технічних вимог щодо вхідних та вихідних даних:

1) широкоформатна індексація – система повинна автоматично завантажувати, обробляти та векторизувати навчальні матеріали різних популярних на LMS платформах форматів документів (часто PDF, DOC, DOCX, HTML);

2) семантичний пошук – реалізація механізму пошуку відповідей на запитання, сформульованих природною мовою, з урахуванням контексту та синонімічних зв'язків, а не лише ключових слів;

3) генерація відповідей на основі джерел – система повинна генерувати стислу відповідь на запит користувача враховуючи отримані результати пошуку.

Підсистема соціальної взаємодії для стимулювання піринговой академічної активності:

1) пошук експертизи – функція надання рекомендацій щодо потенційних менторів за запитом, наприклад, «Хто може допомогти з мікросервісною архітектурою на Spring Boot?», базуючись на об'єктивних даних успішності, а не на суб'єктивних заявах;

2) аналіз академічного сліду – автоматичний збір та обробка даних про успішність студентів із системи керування навчанням Moodle для виявлення корисних компетенцій.

Для забезпечення якісного користувацького досвіду та технічної стійкості система повинна задовольняти наступні нефункціональні вимоги:

- продуктивність (латентність) – час генерації відповіді кожним модулем не повинен перевищувати 30 секунд (враховуючи особливість міркуючої моделі); отримання даних повинне відбуватися миттєво за рахунок кешування;
- масштабованість – архітектура має бути побудована за мікросервісним принципом (окремі контейнери для API, баз даних, індексаторів) та з дотриманням API-контрактів, що дозволятиме незалежне масштабування модулів при зростанні навантаження чи надаватиме гнучкість перебудови системи;
- актуальність даних – індексація нових матеріалів та оновлення соціального графа повинні відбуватися у фоновому режимі з періодичністю, достатньою для підтримки актуальності (наприклад, раз на годину або раз в хвилину для цілей демонстрації);
- стійкість до помилок – система повинна мінімізувати генерацію неправдивої інформації шляхом налаштування низького параметру температури моделі та суворих інструкцій у системному промпті; поламки будь-якої з підсистем не повинні зупиняти мовну модель від надання відповіді, у тому числі незадовільної (наприклад, «не вдалося знайти результати»).

У рамках магістерської роботи розробляється демонстраційний

прототип (англ. minimum viable product, MVP), який має такі архітектурні та функціональні обмеження:

1) інтеграція реалізована для LMS Moodle, розгорнутої локально в тестових цілях;

2) контроль та обмеження доступу до даних не налаштовується;

3) соціальний граф будується на основі спрощеної моделі компетенцій (лише фінальні оцінки за курс).

Сформульовані в пункті вимоги слугують основою для вибору технологічного стека та проєктування архітектури, що буде детально розглянуто далі.

3.2. Опис засобів та API для побудови й інтеграції ШІ-систем

Проєктування сучасних інтелектуальних систем вимагає відходу від традиційної монолітної архітектури на користь розподілених рішень. Враховуючи гібридну природу розроблюваної системи, яка поєднує ресурсномісткі процеси обробки природної мови (RAG) та операції з базами даних у реальному часі, як базовий архітектурний патерн обрано мікросервісну архітектуру.

Вибір підходу зумовлений необхідністю забезпечення масштабованості та ізольованості відмов. Процеси індексації навчальних матеріалів є асинхронними та створюють значне навантаження на обчислювальні ресурси, тоді як отримання заздалегідь підготовленої інформації вимагає миттєвої реакції з мінімальною латентністю. Розділення цих функцій на незалежні сервіси, що комунікують через чітко визначені API-контракти, дозволяє оптимізувати навантаження та гарантувати, що можливі проблеми в одному модулі не поширяться на всю систему й не вплинуть на доступність чат-бота для користувача. Таким чином, архітектура, де зовнішні сервіси інтегруються через API-адаптери, а внутрішня логіка інкапсульована в ізольованих контейнеризованих мікросервісах, дозволяє інтеграцію системи

в існуючий IT-ландшафт університету. Таку архітектуру доцільно розгортати на основі трьох груп програмних інтерфейсів (API), про які детальніше йтиметься далі, а саме: OpenAI API для виконання запитів до LLM, Moodle API для взаємодії з однойменною LMS та інші (власні) API.

Для реалізації функцій генеративного штучного інтелекту використовуватиметься API компанії OpenAI (зокрема моделей серій GPT-4o, GPT-5). Вибір цього провайдера, на відміну від розгортання локальних моделей (LLaMA, Mistral), зумовлений необхідністю високої точності слідування інструкціям та вбудованою підтримкою інструментарію Function Calling (виклик функцій). Використання спеціальних бібліотек, про які йтиметься далі, дозволяє системі не просто генерувати текст, а виступати в ролі агента: аналізувати запити користувача і самостійно вирішувати, який програмний інструмент викликати – пошук у базі знань чи запит до соціального графа.

Система управління навчанням Moodle виступає головним джерелом даних про навчальний контент та успішність студентів. Взаємодія здійснюється через стандартний Moodle API (Web Services) для виконання дій від імені спеціалізованого користувача (service account) й отримання доступу до наступного функціоналу в системі управління навчанням:

- Отримання структури курсів та завантаження контенту, ресурсів та посилань на файли (лекції, методичні вказівки) й подальшого їх аналізу з метою створення бази знань, придатної для семантичного пошуку (див. п. 2.3).
- Отримання метаданих – проходження по структурах курсів та матеріалів, а також користувачах та групах для визначення ієрархічної структури інформації та вивчення горизонтальних зв’язків (наприклад, студент-курси, викладачі-групи, студенти-група).
- Доступ до журналу оцінок для формування «академічного сліду» в соціальному модулі. Оскільки Moodle API не підтримує механізми вебхуків (webhook) для всіх подій та детальний моніторинг успішності, подібна інтеграція може бути реалізована непрямим шляхом через періодичне

опитування вузла (polling, data scrapping).

Для виконання проєкту доцільно розгорнути власний екземпляр LMS Moodle [15] в локальному середовищі розробки. Забезпечення доступу до системи відбуватиметься через загальнодоступний зовнішній домен та реверсивний проксі за допомогою Caddy [34], оскільки обраний знімок контейнера Docker для повноцінної роботи вимагає використання протоколу HTTPS для забезпечення надійності з’єднання через TLS шифрування.

До третьої групи програмних інтерфейсів відносяться власні API внутрішніх мікросервісів. Для зв'язку між компонентами системи (наприклад, для доступу до векторного сховища) розробляються відповідний RESTful API для надання необхідного рівня абстракції над базами даних, приховуючи складність SQL, внутрішніми системами університету, застарілими підсистемами для уніфікації взаємодії компонентів незалежно від мови їх реалізації (Python, Java) та забезпечення контролю доступу до даних – відкриття тільки необхідних моделі для виконання завдань API (що є вимогою безпеки згідно принципу найменших привілеїв).

Реалізацію програмних модулів системи доцільно здійснювати мовою програмування Python [35], вибір якої є стандартом де-факто у сфері штучного інтелекту та машинного навчання (AI/ML) завдяки розвиненій екосистемі бібліотек та високій швидкості прототипування продуктів. Для забезпечення надійності коду та мінімізації помилок на етапі виконання активно використовується механізм анотування типів (строга типізація в динамічно типізованій мові програмування).

Оскільки архітектура системи базується на мікросервісах, необхідний інструмент для створення швидких та надійних API-ендпоїнтів (англ. endpoint – кінцева точка, шлюз взаємодії).

- HTTPX – бібліотека для здійснення асинхронних HTTP-запитів до зовнішніх сервісів. Вона дозволяє виконувати паралельні запити (наприклад, одночасно отримувати дані про три різні курси з Moodle) без блокування основного потоку виконання програми, що суттєво знижує загальну

латентність системи.

- FastAPI – обрано як основний вебфреймворк. На відміну від традиційних Flask або Django, FastAPI побудований на стандарті ASGI (Asynchronous Server Gateway Interface), що дозволяє ефективно обробляти одночасні з'єднання та підтримувати тривалі з’єднання, що важливо для системи, яка часто перебуває в стані очікування відповіді (I/O bound) від зовнішніх API (OpenAI або Moodle).
- Uvicorn – високопродуктивний ASGI-сервер, що використовується для запуску додатків, написаних на FastAPI.

Для побудови логіки роботи інтелектуального агента, управління промптами та ланцюжками викликів використовується екосистема бібліотек LangChain. Вона виступає прошарком абстракції між «сирим» API моделі та бізнес-логікою додатку за допомогою спеціалізованих модулів:

- `langchain_core` – забезпечує базові примітиви: структури повідомлень, інтерфейси для серіалізації виводу та шаблони промптів для стандартизації формату даних, що передаються між різними компонентами системи.
- `langchain_openai` – спеціалізований адаптер для роботи з моделями OpenAI. Він інкапсулює логіку обробки помилок мережі, налаштування параметрів генерації (`temperature`, `top_p`) та автоматичну конвертацію Python-функцій у формат JSON-схеми для механізму Function Calling.
- `langchain_community` – надає функціональні інструменти та завантажувачі документів, що пришвидшує інтеграцію з поширеними форматами даних.

Оскільки великою частиною проекту є читання та зберігання різних форматів даних, необхідно визначити набір спеціалізованих бібліотек-драйверів, роль яких полягатиме в абстрагуванні від низькорівневих операцій читання файлів та мережевої взаємодії з базами даних. Для забезпечення збереження стану системи та швидкого доступу мікросервісів до даних

використовуватимуться офіційні клієнтські бібліотеки (SDK) та драйвери інтеграції з базами даних:

- PyMilvus обрано як основний конектор до векторної бази даних Milvus, яка підтримує асинхронні операції та наявність типізованих методів для роботи зі схемами колекцій, що спрощує валідацію даних на рівні коду Python перед їх відправкою у сховище.
- Redis – високопродуктивна документо-орієнтована NoSQL база даних, яка забезпечує інтеграцію механізмів кешування та черг повідомлень, необхідних для обміну даними між синхронним API чат-бота та фоновими процесами індексації. Використання цієї бібліотеки дозволяє реалізувати патерн «ключ-значення» для збереження проміжних результатів роботи соціального модуля (кеш).

Оскільки Moodle API повертає контент у вигляді бінарних файлів, система потребує надійних інструментів для їх програмного читання («парсингу»). Замість використання важких пропрієтарних рішень, обрано стек Open Source бібліотек, які забезпечують уніфікацію вхідних даних:

- Unstructured – виступає як універсальний шлюз для обробки складних форматів (Microsoft Word, RTF, ODT, Microsoft Excel, PowerPoint та багато інших). Бібліотека обрана через її здатність автоматично визначати структуру документа та виокремлюючи метадані.
- PyPDF використовується для швидкої обробки стандартизованого PDF формату, а саме трансформувати бінарний потік даних у чистий текст, придатний для подачі на вхід мовним моделям.
- BeautifulSoup4 – інструмент для вебскрапінгу контенту, який дозволяє ефективно очищати HTML-розмітку, отриману з вебсторінок, перетворюючи її на структуровані дані.

Для конфігурації інфраструктури доцільно взяти інструмент Python-dotenv, який забезпечує інтеграцію додатку з середовищем виконання (локальний запуск, Docker-контейнер). Бібліотека дозволяє завантажувати параметри конфігурації (URL-адреси мікросервісів, таймаути, секретні ключі)

зі спеціальних файлів, відокремлюючи код від конфігурації, що є вимогою для побудови безпечних розподілених систем згідно з методологією застосунку дванадцяти факторів [3].

Використання наведеного стеку технологій дозволить створити модульну систему, де заміна будь-якого компонента вимагатиме лише зміни відповідного драйвера, API-адаптера, проксі чи адреси сервісу, не порушуючи загальної архітектури. Слід враховувати, що розробка багатомодульних систем завжди призводить до додаткових затрат на налаштування з'єднання через відповідні клієнти та оркестрацію сервісів за допомогою спеціальних інструментів.

3.3. Підбір великих мовних моделей для систем опрацювання знань

Ефективність гібридної інтелектуальної системи безпосередньо залежить від якості «когнітивного ядра» – обраної великої мовної моделі (Large Language Model) та моделі для векторизації даних (Embedding Model). Оскільки ринок ШІ-рішень розвивається експоненціально, вибір конкретних інструментів вимагає багатофакторного аналізу, що враховує баланс між вартістю інференсу (англ. inference – процес виконання моделі), якістю генерації та швидкістю відгуку, а також вимогливістю до апаратної частини.

Для цілей даної магістерської роботи, яка передбачає створення демонстраційного прототипу, важливим з точки зору зручності розробки, легкості зневадження та гнучкості налаштування є забезпечення можливості автономного розгортання системи, завдяки чому поряд із хмарними рішеннями (SaaS) розглядаються відкриті, оптимізовані для локального запуску.

Для обґрунтованого вибору генеративної моделі слід визначити ряд критеріїв, яким вона повинна відповідати:

1) здатність чітко дотримуватися складних системних інструкцій, наповнених інформацією про використання інструментів, у тому числі за

допомогою механізмів міркування;

2) вбудована підтримка виклику зовнішніх інструментів (пошук у базі, запит на API-ендпоїнт) у форматі структурованих даних з аргументами;

3) достатньо якісна підтримка української мови незважаючи на малі розміри моделі (орієнтовно 3-8 мільярдів параметрів) без видимих порушень логічності та послідовності мислення.

Порівняльний аналіз поширених моделей для реалізації ШІ-систем наведено в табл. 3.1.

Таблиця 3.1

Порівняльний аналіз останніх великих мовних моделей для освітніх задач

Категорія	**Моделі**	**Переваги**	**Недоліки**	**Сфера**
Пропрієтарні SOTA (Хмарні)	GPT-4o, Claude 3.5 Sonnet	Висока якість міркувань та слідування інструкціям; розуміння контексту розміром до 128 тис. токенів	Висока вартість API та питання конфіденційності даних	Професійна, рекомендовано для розгортання системи
Відкриті ваги	Gemma 3 4B IT, Llama 3.2 3B	Безкоштовне локальне використання; повний контроль над даними; низька затримка інференсу	Мале контекстне вікно; деградація якості на логічних задачах	Для розробки MVP, тестування, прототипування, використання в системах IoT
Міркуючі моделі	Qwen 2.5/3 Thinking	Вбудований процес мислення, що знижує рівень галюцинацій, але вимагає додаткового часу на генерування	Повільніша генерація через додаткові міркувальні токени	

Для етапу розробки та демонстрації системи обрано стратегію

локального розгортання моделей з використанням переносного формату пакування ваг моделі GGUF (GPT-Generated Unified Format), який дозволяє використовувати квантування (англ. quantization) – метод стиснення ваг нейромережі (наприклад, з 16 біт до 4 біт) з мінімальною втратою якості, що робить можливим запуск помірно потужних моделей на споживчому обладнанні без необхідності використання дороговартісного серверного обладнання з великими об'ємами ресурсів CPU/RAM/GPU.

Експериментальна перевірка показала, що найбільш оптимальним вибором для ядра системи є модель Qwen 3 4B Thinking (версія 2507) від Alibaba Cloud [32], адже модель демонструє високі показники у тестах на логічне мислення, досить добре «спілкується» українською та підтримує механізм виклику функцій для перемикання між RAG та соціальним модулем.

Ембеддинг-модель відповідає за перетворення текстових фрагментів у вектори. Від її якості залежить точність семантичного пошуку: якщо модель погано сприймає українську мову або специфічну термінологію, то система RAG не знайде потрібний документ або знайде невідповідний. Порівняльний аналіз найпопулярніших моделей для векторизації тексту загального призначення наведено в табл. 3.2.

Таблиця 3.2

Порівняльний аналіз моделей для ембеддингу тексту

Модель	**Розмір вектору**	**Переваги/недоліки**
OpenAI text-embedding-3-small	1536	Платна; висока якість, проблема конфіденційності даних
Intfloat multilingual-e5-large	1024	Висока якість, але вимагає значних обчислювальних ресурсів
Google embeddinggemma-300M	768; обрізаний 256	Новітня архітектура; дуже легка для локального запуску, висока швидкість індексації, висока якість за малих векторів
Nomic embed-text-v1.5	768; обрізані 512, 256	Ефективно використовує ресурси й дозволяє обрізати вектори майже без втрат

На основі тестування для системи обрано модель EmbeddingGemma 300M у форматі GGUF [22] в тому числі через архітектурну спорідненість із сімейством моделей Gemma/Qwen, компактність та швидкість генерації, що важливо при індексації великих об’ємів документів (сотні сторінок). Розмірність вектору є достатньою для розрізнення навчальних контекстів, а швидкість роботи дозволяє індексувати нові файли Moodle майже миттєво на локальній машині.

Локальний інференс обраних моделей забезпечуватиметься через встановлений на робочій машині сервер (наприклад, LM Studio або Llama.cpp), який використовує апаратне прискорення для максимальної ефективності роботи й надає OpenAI-сумісний API, що дозволяє безшовно перемикатися між локальними та хмарними моделями без зміни коду клієнта.

Окрім інтелектуальних здібностей, важливим компонентом є здатність моделі оперувати інструментами. Для цього використовується механізм Function Calling/Tool Use, який дозволяє моделі замість тексту генерувати структурований JSON-об'єкт із назвою функції та аргументами в окремих ділянках відповіді, який далі підхоплюється програмою-спостерігачем та виконується, після чого результати запуску повертаються назад до моделі.

Процес прийняття рішень моделлю керується системним запитом (промптом) – базовою інструкцією, що визначає поведінку агента й задається на початку взаємодії. Важливим аспектом інженерії промптів у цій роботі є баланс між деталізацією та лаконічністю: з одного боку, модель потребує чіткого опису доступних інструментів, а з іншого – надмірно довгий промпт перевантажує контекстне вікно, що може призвести до «забування» важливих деталей діалогу, або порушує увагу до користувацького запиту й знижує продуктивність генерації.

Для оптимізації взаємодії системний промпт розроблено за модульним принципом, де опис інструментів подається у вигляді компактних схем, зрозумілих для моделей класу Thinking (Qwen), що дозволяє ефективно маршрутизувати запити користувача.

3.4. Засоби та методики індексації знань у векторні простори

Ефективність системи RAG визначається не лише якістю обраної моделі (див. п. 3.3), а насамперед якістю підготовки даних. Процес перетворення «сирих» файлів у векторний простір, придатний для семантичного пошуку, називається індексацією. У рамках даної роботи цей процес буде реалізовано з використанням екосистеми LangChain та методів збагачення контексту, спрямованого на подолання семантичної неоднозначності.

Для роботи з гетерогенними артефактами університетської системи керування навчанням (PDF-методички, Word-документи, PowerPoint-презентації, HTML-сторінки Moodle) слід застосовувати спеціалізовані класи-завантажувачі бібліотеки `langchain_community`. Архітектура пайплайну може бути побудована таким чином, що кожен тип файлу обробляється окремим адаптером, який повертає стандартизований об'єкт Document, що містить два поля: `page_content` (текст) та `metadata` (словник з атрибутами файлу):

1) для PDF-файлів – `PyPDFLoader`, який дозволяє витягувати текст посторінково, зберігаючи номер сторінки у метаданих для подальшого цитування;

2) для HTML-файлів – `BSHTMLLoader`, який отримує на вхід URL-кодований HTML та повертає його у форматі UTF-8;

3) для документів DOC/DOCX – `UnstructuredWordDocumentLoader`, для документів RTF – `UnstructuredRTFLoader`, для презентацій PPT/PPTX – `UnstructuredPowerPointLoader`, що забезпечує чисте вилучення тексту без XML-тегів форматування;

4) для документів TXT – `TextLoader` для читання тексту та нормалізування кодування.

Стандартні методики індексації, як правило, використовують дещо сліпий підхід до розбиття тексту, коли документ просто ріжеться на шматки по N символів. Головним недоліком такого підходу є втрата глобального

контексту, а саме: фрагмент тексту «Завдання 1. Визначити похибку вимірювання...» виглядатиме ідентично як для курсу фізики, так і для курсу метрології. У векторному просторі ці два фрагменти будуть розташовані майже в одній точці, що порушує ціль семантичного розрізнення сенсу при пошуку. Для вирішення цієї проблеми у роботі пропонується створювати metadata-padded embeddings – ембеддинг, доповнений метаданими. Суть методу полягає у примусовій модифікації текстового вмісту фрагмента перед його векторизацією шляхом конкатенації з контекстними метаданими.

Алгоритм формування вхідних даних для ембеддинг-моделі починається з порівняння кінцевого рядка $N_{symbols}$ з розміром контекстного вікна моделі N_{inputs}, причому:

- якщо контекстне вікно менше/рівне очікуваної довжини рядка, рядок передається без змін;
- якщо контекстного вікна недостатньо, інформація не буде сприйнята моделлю в повному обсязі (актуально в більшості для файлів), тому вміст буде розділено між кількома рядками з додаванням метаданих.

Алгоритм можна подати формулою:

$$Chunks = \begin{cases} N_{symbols} \leq N_{inputs}, & [text(Metadata) + Delim + Content] \\ N_{symbols} > N_{inputs}, & \begin{bmatrix} text(Metadata) + Delim + Part\ 1, \\ text(Metadata) + Delim + Part\ 2, \\ \dots \\ text(Metadata) + Delim + Part\ n \end{bmatrix} \end{cases} \quad (3.1)$$

Такий підхід дозволяє «запекти» контекст безпосередньо у векторне представлення. Вектор для фрагмента з фізики буде зміщений у бік кластера «природничі науки», а вектор для фрагмента з метрології – у бік «технічні вимірювання», навіть якщо основний текст однаковий.

Процес підготовки даних диференціюється залежно від типу джерела інформації. Навчальні матеріали мають чітку ієрархічну структуру прив'язки до дисципліни, завдяки чому для них формується розширений набір

метаданих, який дозволяє системі орієнтуватися в структурі курсу:

1. Витягання даних: періодичне сканування курсів Moodle, завантаження нових/оновлених файлів, а також створення об’єкту-словника, що містить назву курсу (повну та коротку), назву секції (теми), назву модуля та ім'я файлу (лістинг 3.1);

2. Перетворення даних та метаданих виконується згідно наведеної вище формули: вміст файлу ділиться на частини й кожна частина об’єднується зі структурою метаданих через роздільник – символ переходу на новий рядок `\n`;

Лістинг 3.1. JSON-об’єкт метаданих, що містить метадані про файл (курс, розділ, модуль, ім’я файлу)

```
{
    "Курс": "Інновації сфери штучного інтелекту (ІСШІ)",
    "Розділ": "Лекція 8. Етика штучного інтелекту",
    "Модуль": "Презентація Етика ШІ",
    "Файл": "Лекція Етика ШІ.pptx",
}
```

3. Завантаження даних: набір рядків поступово передається ембеддинг-моделі на вбудовування й зберігається в базі даних, з урахуванням того, що одне джерело може породжувати $[1 \dots n]$ документів у векторній БД.

Загальна схема процесу зображена на рисунку 3.1.

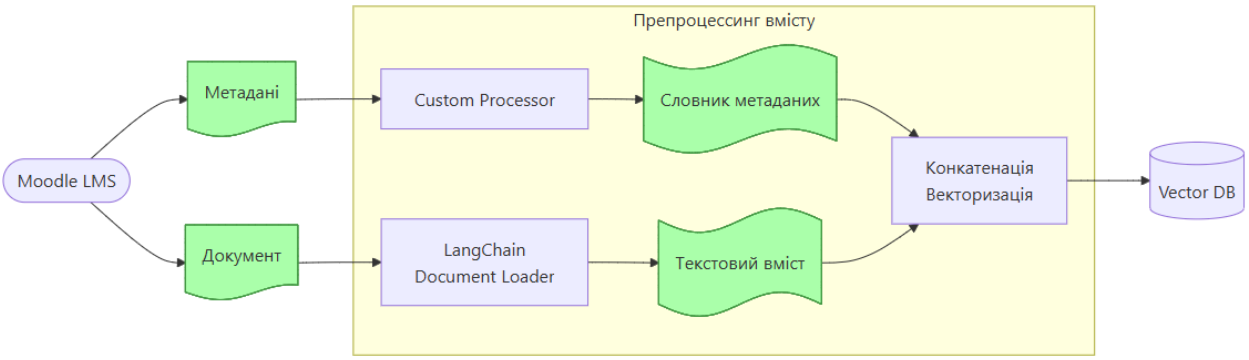


Рис. 3.1. Загальний вигляд ETL-пайплайну індексації файлів у векторну базу даних

Вертаючись до метаданих самого документа, які можна отримати із раніше згаданого інтерфейсу LangChain Document, варто розуміти, що контекстуальна цінність таких метаданих мінімальна, оскільки вони часто містять лише автоматично заповнену інформацію, таку як: дати створення, модифікації, останнього доступу до документа, назви пристроїв, автори документів (причому імена авторів беруться з імен облікових записів на персональних комп'ютерах, які рідко відповідають реальним іменам автора/авторів або містять беззмістовну інформацію, таку як: User, Admin, PC-1.415, Василь тощо). Через низьку цінність такої інформації та міркування щодо надлишковості та її потенційних наслідків («зашумлення» векторного сховища), вона буде відкидатися при обробці документів.

Для структурованих даних типу «питання-відповідь» (FAQ) необхідний окремий підхід до завантаження, який ітерується по списку JSON об'єктів-словників й кожен з них індексує як окремий документ. Записи FAQ часто є короткими та лаконічними, для них застосовується інвертована стратегія, тобто тіло контенту файлу залишається порожнім, а вся семантично значуща інформація переноситься у блок метаданих, який потім повністю подається на вхід моделі векторизації:

1. Витягання даних: періодичне сканування нових JSON-файлів з корисною інформацією для вбудовування у форматі словника з ключами «question» та «answer»;
2. Перетворення даних: створення словника метаданих з полями «Запитання» та «Відповідь» та порожнім файловим вмістом;
3. Завантаження даних: модель індексує пару Q&A як єдине ціле, що дозволяє знаходити відповідь навіть якщо запит користувача семантично близький до тексту відповіді, а не лише до тексту питання.

Використання методу metadata-padded embeddings забезпечить значне підвищення релевантності пошукової видачі. Система отримує здатність розрізняти омонімічні запити, усувати неоднозначності та надавати пріоритет документам, що відповідають поточному академічному контексту студента,

мінімізуючи ймовірність отримання інформації з суміжних джерел, нерелевантних дисциплін тощо.

3.5. Проєктування соціального модуля пошуку експертизи

У багатьох системах пошук менторів реалізовано самозаявленими навичками в профілях користувачів. Такий підхід не є продуктивним, оскільки студенти рідко коли пам’ятають про оновлення свого переліку навичок та актуалізації даних про володіння технологіями/інструментами/темами. Пропонується відійти від традиційного пошуку по заявлених мітках до пошуку на основі об'єктивних даних успішності зібраних із розміщених активностей.

Активність (Activity) – це інтерактивний елемент навчального курсу на платформі Moodle, який вимагає від студента виконання певної дії (вводу даних, завантаження файлу, вибору відповіді, комунікації) і, як правило, передбачає оцінювання або фіксацію факту виконання.

Для реалізації пошуку експертів на основі даних успішності розроблено алгоритм, який складається з наступних кроків:

1. Реалізація ETL-процесу для підготовки структурованого графу компетенцій та пошукових метаданих;
2. Семантичне зв'язування для визначення, яка саме академічна активність (лабораторна робота чи тест) відповідає темі запиту користувача (семантичний пошук активностей на основі запиту користувача);
3. Евристичне ранжування на основі спеціальної формули розрахунку релевантності кандидата.

Оскільки прямі запити до Moodle API є ресурсномісткими та повільними, архітектура передбачає асинхронний компонент, завданням якого буде періодична (наприклад, щогодинна) синхронізація даних та формування «профілів компетенцій» у швидкому сховищі Redis.

Процес обробки даних включає наступні кроки:

1. Нормалізація оцінок, адже різні курси та різні активності мають різні шкали оцінювання (100-бальна, 5-бальна, ECTS, будь-яка інша шкала), за якої усі результати приводяться до єдиного коефіцієнта успішності $G \in [0, 1]$;

2. Назви лабораторних робіт та тестів проходять через ту саму модель ембеддингу, що й документи RAG для їх збереження у векторному просторі для семантичного пошуку (див. п. 3.4);

3. Визначення статусу доступності кандидата за полем активності користувачів Moodle `lastaccess`, класифікуючи їх як Online (наприклад, був менше 30 хв тому), Active (був менше 24 год тому), Present (був менше 3 днів тому) або Offline.

У сховищі Redis дані зберігаються у вигляді сортованої множини (лістинг 3.2).

Лістинг 3.2. Структурне представлення сортованої за спаданням значень множини експертів, де ключем є ID активності в системі Moodle, а значенням – нормалізована оцінка студента

```
moodle:activity:42:experts: [
    student_id_621:1.0,
    student_id_205:0.7,
    student_id_101:0.2,
]
```

Центральною проблемою є інтерпретація наміру користувача. Якщо студент запитує «Хто знає рекурсію?», система повинна зрозуміти, які саме завдання в курсі покривають тему «рекурсія». Для цього використовується гібридний алгоритм пошуку, що пов'язує результати RAG та соціального графа:

1. LLM аналізує запит і виділяє ключові концепти. Наприклад, із наведеного вище запиту можна виділити тему «рекурсія»;

2. Система виконує векторний пошук по назвах та описах завдань у Moodle, які були попередньо векторизовані воркером (англ. worker – виконавець завдання). До прикладу: слово «рекурсія» семантично близька до

вектору назви завдання «Лабораторна робота №4: Рекурсивні алгоритми», звідки система отримає список ID активностей Moodle, що відповідають темі;

3. За отриманими ID система робить запит до Redis й отримує список студентів, які мають позитивну оцінку за ці конкретні завдання.

Отриманий список кандидатів може бути великим, тому необхідно застосувати алгоритм ранжування для вибору найкращих менторів. Пропонується використовувати функцію зваженої оцінки.

Рейтинг кандидата розраховується за формулою:

$$R = w_1 \cdot G_{topic} + w_2 \cdot A_{status} + w_3 \cdot S_{impact} \quad (3.2)$$

де:

- G_{topic} – нормалізована оцінка студента за релевантну активність в діапазоні 0.6 ... 1.0;
- A_{status} – мапа коефіцієнтів доступності (наприклад, 1.0 для Online, 0.85 для Active, 0.6 для Present, 0.1 для Offline) для ранжування за онлайн-близькістю;
- S_{impact} – індекс соціальної активності, що базується на попередніх успішних допомогах (якщо такі дані доступні);
- w_1, w_2, w_3 – вагові коефіцієнти, що налаштовуються для вказання пріоритету знань, доступності та близькості для визначення кінцевого рангу.

Останнім етапом є трансформація структурованих даних у формат, зрозумілий для великої мовної моделі. Сервіс повертає JSON-об'єкт, який містить лише необхідний мінімум даних для збереження приватності.

Наведений підхід забезпечує повну автоматизацію процесу пошуку експертизи, використовуючи об'єктивні дані Moodle, але приховуючи складність обчислень за зручним розмовним інтерфейсом.

3.6. Висновки до розділу

У третьому розділі здійснено концептуальне проєктування та обґрунтування технологічного стека гібридної інтелектуальної системи. На основі аналізу функціональних вимог обрано мікросервісну архітектуру, яка дозволяє ефективно розділити ресурсномісткі процеси семантичної обробки тексту (RAG) та чутливі до затримок операції пошуку соціальних контактів для гарантування масштабованості системи та стійкості до пікових навантажень, характерних для навчального процесу.

У результаті проведеного дослідження технологій прийнято ряд ключових інженерних рішень:

1. Для реалізації когнітивних функцій обрано використання великих мовних моделей з підтримкою виклику інструментів, що дозволить трансформувати LLM в агента. Для етапу розробки обрано локальних оптимізованих моделей.

2. Розроблено методику індексації навчальних матеріалів, що базується на підході metadata-padded embeddings, який вирішує проблему втрати контексту при нарізці документів. Для збереження векторних репрезентацій текстового вмісту обрано високопродуктивну базу даних Milvus.

3. Спроектовано алгоритм побудови та використання соціальних зв'язків, який використовує «академічний цифровий слід» студента. Для забезпечення миттєвого доступу до даних про компетентність студентів обрано архітектурне рішення на базі Redis.

4. Визначено набір API та бібліотек, які забезпечують безшовну взаємодію між користувацьким інтерфейсом, інтелектуальним агентом та джерелами даних.

Сформована архітектура та обраний інструментарій створюють надійний фундамент для програмної реалізації системи. Вони дозволяють вирішити поставлену наукову задачу – поєднати доступ до інформації та

соціальної експертизи в єдиному інтерфейсі. Детальний опис процесу розробки, налаштування та тестування програмних модулів буде наведено у наступному розділі.

РОЗДІЛ 4.
РЕАЛІЗАЦІЯ ПРОГРАМНИХ МОДУЛІВ

4.1. Встановлення програмних засобів та ініціалізація проекту

Розпочинаючи етап програмної реалізації системи, першочерговим завданням визначаю підготовку надійного та відтворюваного середовища розробки. Від правильної конфігурації інструментарію залежить швидкість написання коду, зручність зневадження та простота подальшого розгортання.

Для написання програмного коду та управління проєктом обираю інтегровані середовища розробки (IDE) від компанії JetBrains, які є галузевим стандартом для обраного технологічного стека. Оскільки для більшості сервісів проєкту очікується написання мовою Python, обираю PyCharm Professional за можливості глибокого статичного аналізу коду, інтегровані інструменти для роботи з базами даних та підтримку вебфреймворку FastAPI. Для роботи з клієнтською частиною, конфігураційними файлами інфраструктури та загальної оркестрації проєкту застосовую IntelliJ IDEA Ultimate, що забезпечує відмінну підтримку TypeScript, React, Java та синтаксису Docker.

У якості базового інтерпретатора для проєкту встановлюю версію Python 3.13, що є передостанньою актуальною й гарантує підтримку більшості бібліотек. Вибір цієї версії зумовлений наявністю останніх оптимізацій продуктивності (зокрема, покращеною роботою JIT-компілятора в експериментальному режимі) та сучасними синтаксичними конструкціями, що спрощують написання асинхронного коду.

Для забезпечення ізольованості та переносимості системи використовую платформу Docker Desktop, який дозволяє розгортати всі залежності проєкту (бази даних, Moodle, мікросервіси, проксі) у контейнерах, не засмічуючи хост-систему бібліотеками та конфліктуючими версіями ПЗ.

Структуру репозиторію організовую за принципом монорепозиторію,

адже такий підхід значно спрощує управління версіями, дозволяє зберігати контракти API та код усіх мікросервісів в одному місці, що полегшує колаборацію невеликих команд. Ініціалізую Git-репозиторій та формую структуру каталогів для модулів проєкту (лістинг 4.1).

Лістинг 4.1. Команди ініціалізації структури каталогів проєкту

```
git init                        # Ініціалізація монорепозиторію Git

md bruno                        # Сценарії REST-клієнта для API-
тестів
md chat-ui                      # Фронтенд, користувацький інтерфейс
md docs                         # Документація по проєкту
md embedding-store              # Сховиже ембеддингів
md faq-store                    # Сховище запитань-відповідей
md moodle                       # Засоби розгортання тестового Moodle
md moodle-client                # Бібліотека для взаємодії із Moodle
md rag-indexer                  # Індексатор файлів та частих питань
md study-buddy-store            # Графове сховище академічних даних
md study-buddy-worker           # ETL-процесор академічних даних
md thinking-tool-model-proxy # Підключення LLM до RAG та соцмодуля
```

Для уникнення конфліктів між залежностями різних мікросервісів (так званий ефект Dependency Hell), для кожного сервісу ініціалізую ізольоване віртуальне середовище за допомогою модуля `venv`, що дозволить конфігурувати специфічні версії бібліотек та фіксувати їх у `requirements.txt` файлах (лістинг 4.2).

Лістинг 4.2. Активація віртуального середовища PIP та встановлення залежностей модуля проєкту

```
python -m venv .venv
.\.venv\Scripts\activate.ps1
pip install -r requirements.txt
```

Клієнтську частину додатку ініціалізую за допомогою інструменту збірки Vite. Виконую ініціалізацію за шаблоном React із підтримкою TypeScript (лістинг 4.3), що забезпечить сувору типізацію на фронтенді та

високу швидкість гарячого перезавантаження (HMR) під час розробки.

Лістинг 4.3. Ініціалізація вебпроєкту за допомогою NPM та встановлення залежностей

```
npm create vite@latest -- --template react-ts
npm install
npm run dev
```

Для кожного контейнеризованого мікросервісу фінальним етапом початкового налаштування є створення кореневого файлу `docker-compose.yml`, у якому описую базові сервіси інфраструктури (бази даних, кеш), необхідні для початку роботи над бізнес-логікою. Такий підхід гарантує, що при клонуванні репозиторію на іншу робочу машину достатньо виконати команду `docker-compose up -d`, щоб отримати повністю розгорнуте робоче оточення.

4.2. Проєктування архітектури системи

Архітектура системи спроєктована за мікросервісним принципом, де кожен функціональний модуль є ізольованим контейнером, що виконує чітко окреслене завдання. Така декомпозиція дозволяє незалежно масштабувати компоненти та використовувати різні технологічні стеки для різних задач.

Підсистема семантичного пошуку та індексації знань є елементом, який забезпечує «заземлення» генеративної моделі на актуальні дані університетської бази знань. Проєктування цього модуля базується на розділенні ETL-процесів, що дозволяє оптимізувати навантаження та забезпечити високу доступність сервісу, на окремі три сервіси, а саме: сервіс індексації, сервіс векторного сховища і сервіс поширених запитань.

Сервіс індексації (RAG Indexer) – це компонент, який реалізовано як фоновий процес, що відповідає за ETL-пайплайн. Оскільки система LMS Moodle не надає нативних механізмів миттєвого оповіщення про зміни для всіх

типів подій, у модулі реалізовано стратегію періодичного опитування (polling).

Сервіс векторного сховища (Embedding Store) виступає єдиним шлюзом до бази даних Milvus. Він інкапсулює логіку роботи з векторами, приховуючи від інших компонентів системи складність перетворення тексту у вектор. Для забезпечення персистентності векторних даних використовується Milvus у режимі Standalone, що включає в себе:

- Milvus Server для обробки векторних запитів і перетворення даних.
- MinIO – AWS S3-сумісне об'єктне сховище для фізичного зберігання файлів індексів та логів.
- Etcd – сховище метаданих для координації стану кластера та конфігурацій.

Сервіс поширених запитань (FAQ Store) використовується для обробки статичних, часто повторюваних запитань, які не вимагають складного семантичного пошуку по документах, через що його виділено в окремий мікросервіс на базі Redis. Основним завданням сервісу є інкапсуляція взаємодії з підлеглою базою даних, а також періодичне сканування визначеної в конфігурації директорії завантажень із JSON-файлами для автоматичного споживання оновлень інформації.

Загальна архітектура рішення зображена на рисунку 4.1.

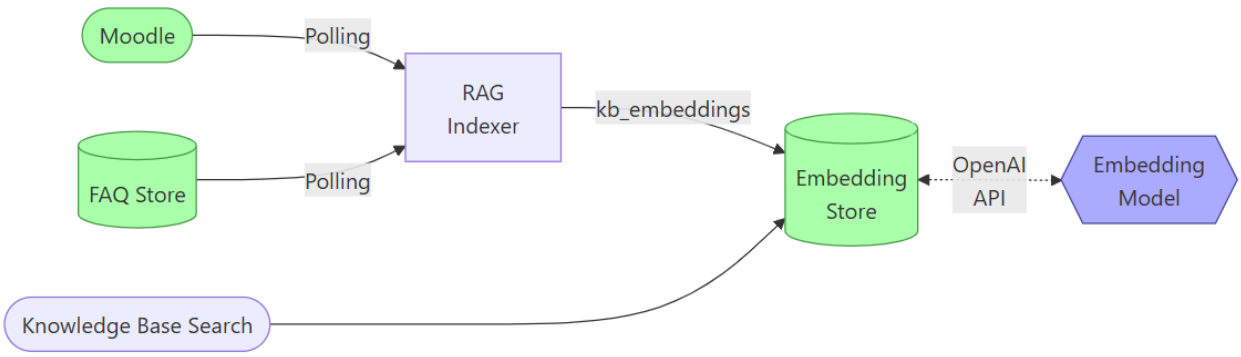


Рис. 4.1. Архітектура рішення семантичного пошуку по Moodle та БД FAQ

Проєктування модуля пошуку менторів базується на вимогах високої

швидкодії та ізоляції чутливих даних про успішність студентів. На відміну від RAG-системи, яка працює з неструктурованим текстом, цей модуль оперує структурованими даними (рейтинги, статуси, ID користувачів), що обумовило вибір специфічного технологічного стека. Реалізація включає три взаємопов'язані компоненти, про кожен з яких йтиметься нижче.

Сервіс збереження даних соціального графа (Study Buddy Store) – це API-сервіс, що виступає інтерфейсом до сховища даних Redis, вибір якого зумовлений необхідністю миттєвого доступу до даних $O(1)$ та нативною підтримкою наступних структур даних:

1. Sorted Sets (ZSET) для зберігання ранжованих списків експертів для кожної активності Moodle, що дозволяє отримувати топ-k менторів одним запитом без додаткових обчислень на стороні додатку.

2. Hashes (HSET) використовуються для зберігання профілів користувачів (контакти, статус, ім'я) для доступу за ID.

Сервіс агрегації компетенцій (Study Buddy Worker) – це фоновий ETL-процес, який забезпечує синхронізацію із зовнішньою LMS Moodle. Воркер виконує подвійну функцію маршрутизації даних, що назви та описи навчальних активностей (наприклад, «Лабораторна робота 3: Основи HTML») відправляються у векторне сховище, а дані про успішність – у сервіс соціального графу.

Соціальний модуль не має власного механізму обробки природної мови, а покладається на результати роботи RAG-системи. Агент спочатку звертається до векторної бази для визначення ідентифікатора активності за темою запиту, а потім використовує цей ідентифікатор як ключ для пошуку в Redis. Така архітектура дозволяє уникнути дублювання логіки й забезпечує розподіл відповідальності: векторна база відповідає за «що шукаємо», а соціальний граф – «кого шукаємо».

Загальна архітектура соціальної підсистеми зображена на рисунку 4.2.

Окремі компоненти, такі як інтерфейс користувача та LLM-проксі відокремлені від бізнес-логіки баз даних, що дозволяє змінювати інтерфейс

або модель ШІ без впливу на структуру збереження даних. Thinking & Tool Use Model Proxy перехоплює запити моделі на виклик функцій і перенаправляє їх до відповідних внутрішніх мікросервісів, повертаючи результат виконання.

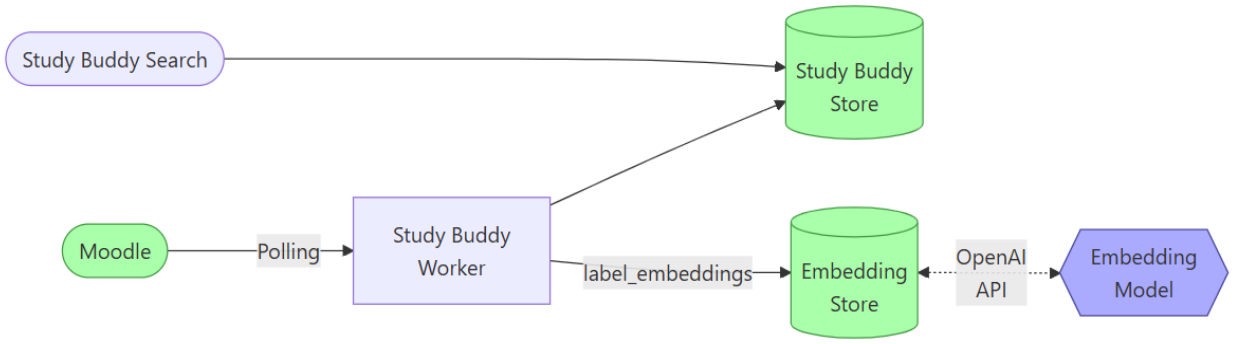


Рис. 4.2. Архітектура рішення соціального пошуку по академічних даних із системи керування навчанням Moodle

Клієнтська частина реалізована як SPA й архітектурно є «тонким клієнтом»: не містить бізнес-логіки обробки даних, а відповідає лише за рендеринг діалогу. Взаємодія з проксі-сервісом здійснюється з підтримкою потокової передачі даних (стримінг), що дозволяє відображати відповідь токен за токеном (ефект друкарської машинки).

Узагальнена архітектура обох підсистем, допоміжних компонентів та їх інтеграцій наведена на рисунку 4.3.

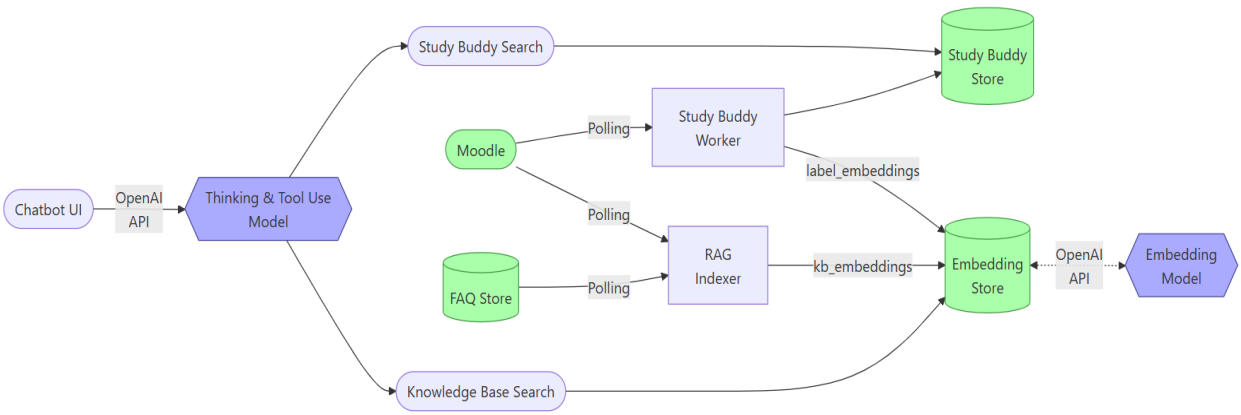


Рис. 4.3. Спрощена архітектура системи, що включає рішення семантичного та соціального пошуку, а також підключення до клієнтської частини

Діаграма послідовності взаємодії користувача з цільовим мікросервісом крізь ШІ-інтерфейс показана на рисунку 4.4.

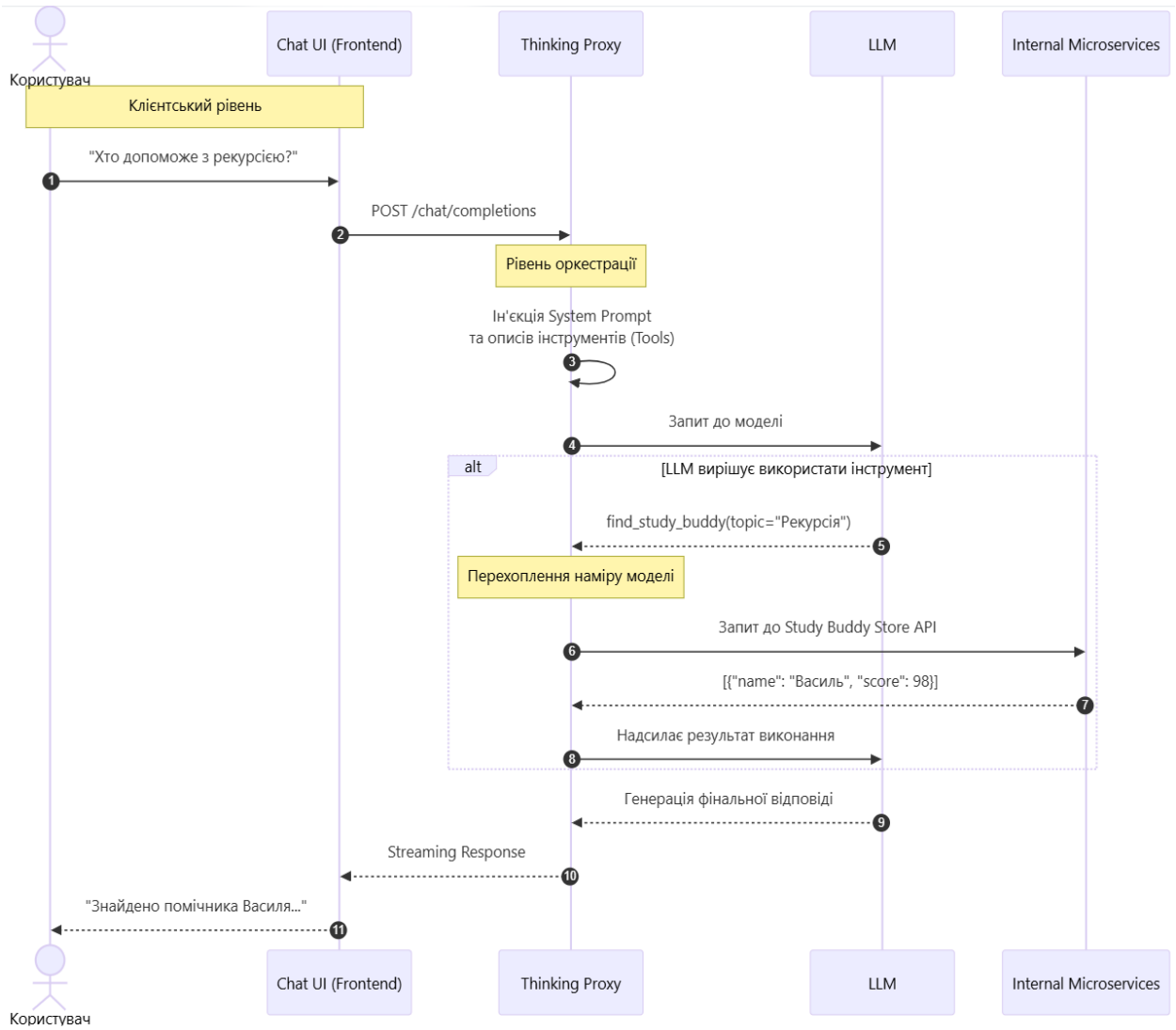


Рис. 4.4. Діаграма послідовності UML отримання доступу до кешованих даних мікросервісів за допомогою ШІ-центричного інтерфейсу

Основою взаємодії усіх компонентів є протокол HTTP (REST API) для синхронних запитів та фонові процеси для асинхронної обробки даних (ETL). В умовах локального розгортання системи мережева взаємодія організована через систему портів на хості `localhost`.

Для доступу до ресурсномістких сервісів, які неможливо або недоцільно розгортати на машині розробника (наприклад, тестовий екземпляр LMS Moodle, який вимагає HTTPS, або сервер LLM моделі, що вимагає

потужного GPU), налаштовую реверсивний проксі Caddy та присвоюю доменне ім’я цільовій машині (лістинг 4.4).

Лістинг 4.4. Caddyfile-конфігурація для HTTP-серверу Caddy з необхідними HTTP-заголовками, необхідними для функціонування Moodle

```
moodle.paws.maws.io {
    log {
        output file {$LOGS_DIR}/moodle.log
    }

    handle {
        reverse_proxy 192.168.50.230:48613 {
            header_up Host {host}
            header_up X-Forwarded-Proto {scheme}
            header_up X-Forwarded-For {remote}
        }
    }
}
```

Для дотримання принципів безпеки та переносимості коду (відповідно до методології застосунку дванадцяти факторів) конфігурація кожного мікросервісу винесена з коду в змінні оточення, а для реалізації зручного налаштування середовища без зміни середовища системи використовуються файли ENV. Кожен сервіс має власний набір конфігураційних параметрів, які визначають:

- топологію зв'язків на основі URL-адрес та портів суміжних сервісів;
- автентифікацію за ключами доступу до Moodle API, OpenAI API чи інших API;
- параметри моделей: назви та їх технічні характеристики (розмірність вектору, ліміт токенів тощо).

Приклад конфігурації фонового сервісу індексації файлів та бази коротких питань-відповідей з параметрами самого індексатора, а також URL-адресами зовнішніх сервісів, від яких даний має другорядну залежність, і ключами доступу наведено в лістингу 4.5.

Лістинг 4.5. Файл конфігурації змінних середовища ENV для налаштування сервісів та систем без зміни коду

```
INDEXER_TIMESTAMP_FILE=data/last_run_timestamp.txt
INDEXER_POLL_INTERVAL_SECONDS=30
INDEXER_FETCH_FAQ_STORE=1
INDEXER_FETCH_MOODLE_COURSE_FILES=1
FAQ_STORE_URL=http://localhost:8300
EMBEDDING_STORE_URL=http://localhost:8200
MOODLE_API_URL=https://moodle.paws.maws.io
MOODLE_API_KEY=59a30c31b009e8330f175e7c308c8e25
```

Загальну діаграму мікросервісної комунікації через оголошені порти наведено на рисунку А.1.

4.3. Підготовка API контрактів та налаштування бекенд-сервісів

Реалізація серверної частини системи базується на фреймворку FastAPI, що дозволяє автоматично генерувати інтерактивну документацію (Swagger UI) та забезпечувати асинхронну обробку запитів. Для гарантії цілісності даних, що передаються між мікросервісами, розроблено API-контракти з використанням бібліотеки Pydantic.

Визначення схем запитів та відповідей (DTO) дозволяє валідувати вхідні дані до початку їх обробки бізнес-логікою. Для векторного пошуку визначено модель `SearchEmbeddingsPayload`, яка уніфікує параметри запиту для RAG та пошуку активностей (лістинг 4.6).

Лістинг 4.6. Pydantic-модель корисного навантаження для виконання векторного пошуку

```
from pydantic import BaseModel, Field
from typing import Optional, Dict, List

class SearchEmbeddingsPayload(BaseModel):
    query: str = Field(..., description="User request")
    top_k: int = Field(5, description="Number of results")
    filter: Optional[Dict[str, str]] = Field(None,
description="Metadata filters (ex. {'type': 'activity'})")
```

```
class SearchResult(BaseModel):
    id: str
    distance: float
    doc_id: str
    metadata: Dict[str, str]
    content: str
    score: float
```

Для соціального сервісу розроблено модель «профілю» експерта, що повертається агенту (лістинг 4.7).

Лістинг 4.7. Pydantic-модель корисного навантаження для виконання векторного пошуку

```
class ExpertProfile(BaseModel):
    user_id: int
    name: str
    score: float = Field(..., description="Score (0.0–1.0)")
    status: str = Field(..., description="Status: online, present,
active, offline")
    contact: str = Field(..., description="Contact link")
```

Мікросервіси надають спеціальні ендпоїнти для виклику агентом через механізм Function Calling. Embedding Store Service обробляє семантичний пошук й реалізує два ключові маршрути:

1. `POST /embeddings/search`: приймає `SearchEmbeddingsPayload` і виконує перетворення тексту `query` у вектор за допомогою ембеддинг-моделі, здійснює пошук у колекції Milvus та повертає фрагменти з найвищою подібністю;

2. `POST /activities/search` використовує ту ж логіку векторизації, але перенаправляє запит в іншу колекцію, що містить ідентифікатори навчальних активностей `activity_id` за їх семантичним описом (наприклад, «лабораторна по циклах»);

Study Buddy Store надає доступ до даних Redis через REST API:

3. `GET /experts/{activity_id}` приймає ID активності, знайдений на попередньому кроці, звертається до Redis Sorted Set

`activity:{id}:experts` для отримання топ-k студентів з найвищим балом, витягує детальні профілі з Redis Hash `user:{id}:profile` та фільтрує результати.

Сервіс Thinking & Tool Use Model Proxy згідно патерну «проксі» передає всі запити на модель, але також вбудовує додаткові дані в запит із фронтенду. Таким чином він виступає оркестратором, який базується на LangChain та інтегрується із сервером інференсу. Для підключення до локальної моделі, запущеної в LM Studio, використовується клас `ChatOpenAI` з перевизначеним базовим URL, який наведено в лістингу 4.8.

Лістинг 4.8. Спосіб перевизначення OpenAI API-ендпоїнту в LangChain

```
llm = ChatOpenAI(
    base_url="https://lms.paws.maws.io/v1",
    # Ключ для серверу локального інференсу непотрібний
    api_key="lm-studio",
    # Параметр, який поважається не всіма серверами
    model="qwen/qwen3-4b-thinking-2507",
    temperature=0.7,
    # Параметр для створення ефекту друкарської машинки
    streaming=True,
)
```

Тут же створюється шаблон чату з додаванням історії та допоміжних елементів, таких як системний промпт, місцезаповнювач повідомлення користувача та чернетка агента (лістинг 4.9).

Лістинг 4.9. Створення шаблону взаємодії, який складається із системного запиту, історії чату, нового запиту користувача та чернетки для моделей, що підтримують міркування

```
prompt = ChatPromptTemplate.from_messages([
    SystemMessage(SYSTEM_PROMPT),
    MessagesPlaceholder("chat_history"),
    ("user", "{input}"),
    MessagesPlaceholder("agent_scratchpad"),
])
```

Поведінка агента регламентується системним повідомленням, яке завантажується зі змінних оточення. Воно інструктує модель спочатку аналізувати запит, обирати інструмент, отримувати JSON-відповідь від API та формувати фінальну відповідь українською мовою.

Функції пошуку оголошуються саме як інструменти LangChain з чітким описом, який модель використовує для прийняття рішень (лістинг 4.10).

Лістинг 4.10. Створення шаблону взаємодії, який складається із системного запиту, історії чату, нового запиту користувача та чернетки для моделей, що підтримують міркування

```
@tool
def find_study_buddy(topic: str):
    """
    Finds students who are experts in a specific topic.
    Args: topic (str): concept or task name ("Recursion", "Lab
1").
    """
    try:
        best_match =
requests.post(f"{EMBEDDING_STORE_URL}/activities/search", json={
            "query": topic,
            "filter": {"type": "activity"},
            "top_k": 1,
        }).json().get("results", [])[0]

        act_id = best_match['metadata']['activity_id']
        act_name = best_match['metadata'].get('name')
        experts =
requests.get(f"{STUDY_BUDDY_STORE_URL}/experts/{act_id}").json()

        result_text = f"Знайдено менторів для '{act_name}':\n"
        for expert in experts[:3]:
            result_text += f"{expert['status']} {expert['name']}
(Score: {int(expert['score'] * 100)}%) - Contact:
{expert['contact']}\n"
        return result_text.strip()
```

Нарешті, функції чи інструменти підключаються до об’єкту «агент» та створюється виконавець (executor), завданням якого буде власне виконання та отримання результатів роботи й передавання їх моделі назад на опрацювання (лістинг 4.11).

Лістинг 4.11. Список інструментів, створення агента та виконавця команд

```
all_tools = [
    search_knowledge_base,
    find_study_buddy,
]
agent = create_openai_tools_agent(llm, all_tools, prompt)
agent_executor = AgentExecutor(agent=agent, tools=all_tools)
```

Для роботи агента налаштовується локальний сервер моделей, який в тому числі надає можливості виклику інструментів. У налаштуваннях сервера також слід активувати JIT-завантаження моделей та CORS для дозволу запитів (рис. 4.5).

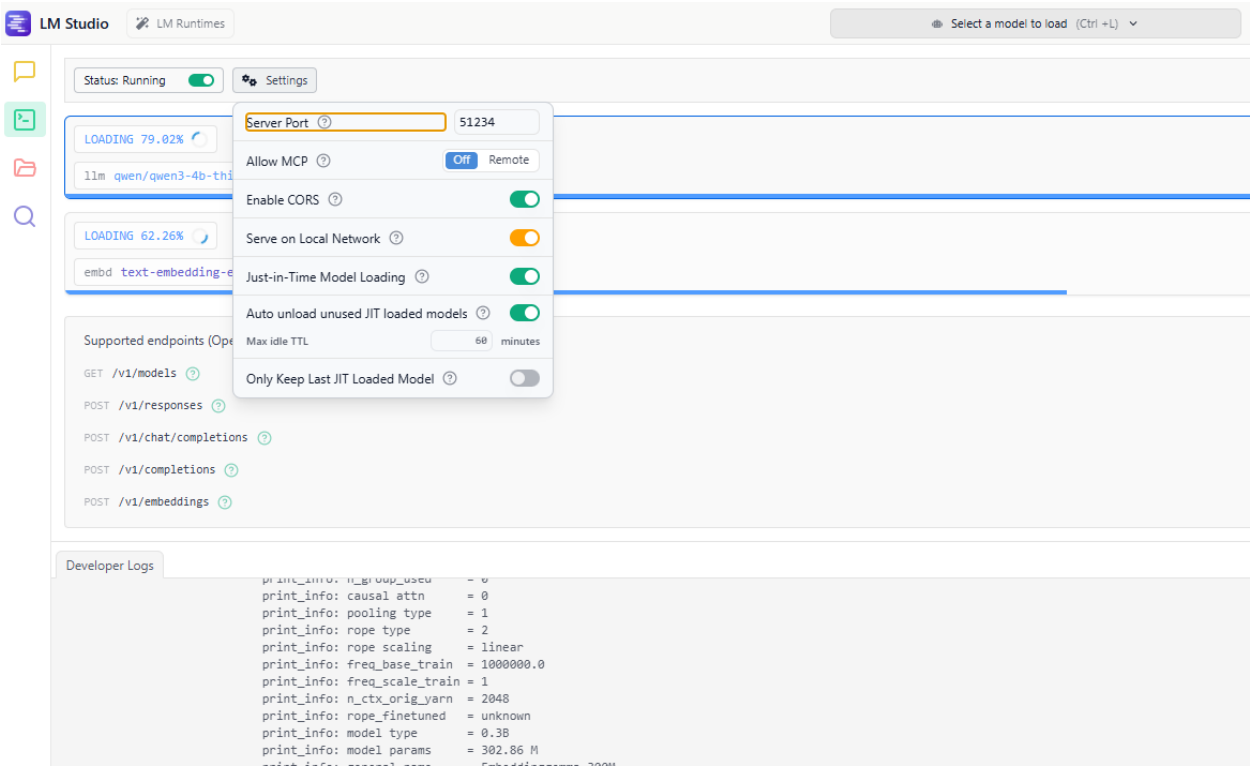


Рис. 4.5. Вигляд інтерфейсу LM Studio на вкладці «Розробник» з параметрами сервера та активним завантаженням моделей

Оскільки архітектура системи передбачає роботу з приватними даними університету (оцінки, закриті методичні матеріали), необхідно налаштувати доступ до системи керування навчанням Moodle. Розгортаю власний

екземпляр системи для тестування, конфігурацію наведено в лістингу 4.12.

Лістинг 4.12. Файл `docker-compose.yml` для розгортання Moodle в контейнері Docker

```yaml
services:
  postgres:
    image: postgres:18-alpine
    environment:
      - POSTGRES_DB=moodle
      - POSTGRES_USER=moodle
      - POSTGRES_PASSWORD=moodle
    volumes:
      - postgres-data:/var/lib/postgresql/data
    restart: unless-stopped

  moodle:
    image: erseco/alpine-moodle:v5.1.0
    ports:
      - "48613:8080"
    environment:
      - LANG=uk_UA.UTF-8
      - LANGUAGE=uk_UA:uk
      - SITE_URL=${MOODLE_URL}
      - REVERSEPROXY=false
      - SSLPROXY=true
      - DB_TYPE=pgsql
      - DB_HOST=postgres
      - DB_PORT=5432
      - DB_NAME=moodle
      - DB_USER=moodle
      - DB_PASS=moodle
      - MOODLE_SITENAME=Moodle4Noodle
      - MOODLE_LANGUAGE=uk
      - MOODLE_EMAIL=${MOODLE_ADMIN_EMAIL}
      - MOODLE_USERNAME=${MOODLE_ADMIN_USERNAME}
      - MOODLE_PASSWORD=${MOODLE_ADMIN_PASSWORD}
    volumes:
      - moodle-data:/var/www/moodledata
      - moodle-html:/var/www/html
    depends_on:
      - postgres
    restart: unless-stopped

volumes:
  postgres-data:
  moodle-data:
  moodle-html:
```

Після розгортання проводжу початкове налаштування. Оскільки публічний доступ до Moodle є неможливим, для забезпечення програмного доступу мікросервісів до LMS налаштовую підсистему Moodle Web Services виконуючи офіційні вказівки, що включають наступні кроки:

1) В адміністративній панелі Moodle вмикаю підтримку протоколу REST, який є стандартом для обміну даними у JSON-форматі;

2) Створюю зовнішній сервіс (рис. 4.6) та додаю до нього лише необхідний перелік функцій Moodle API для роботи воркерів;

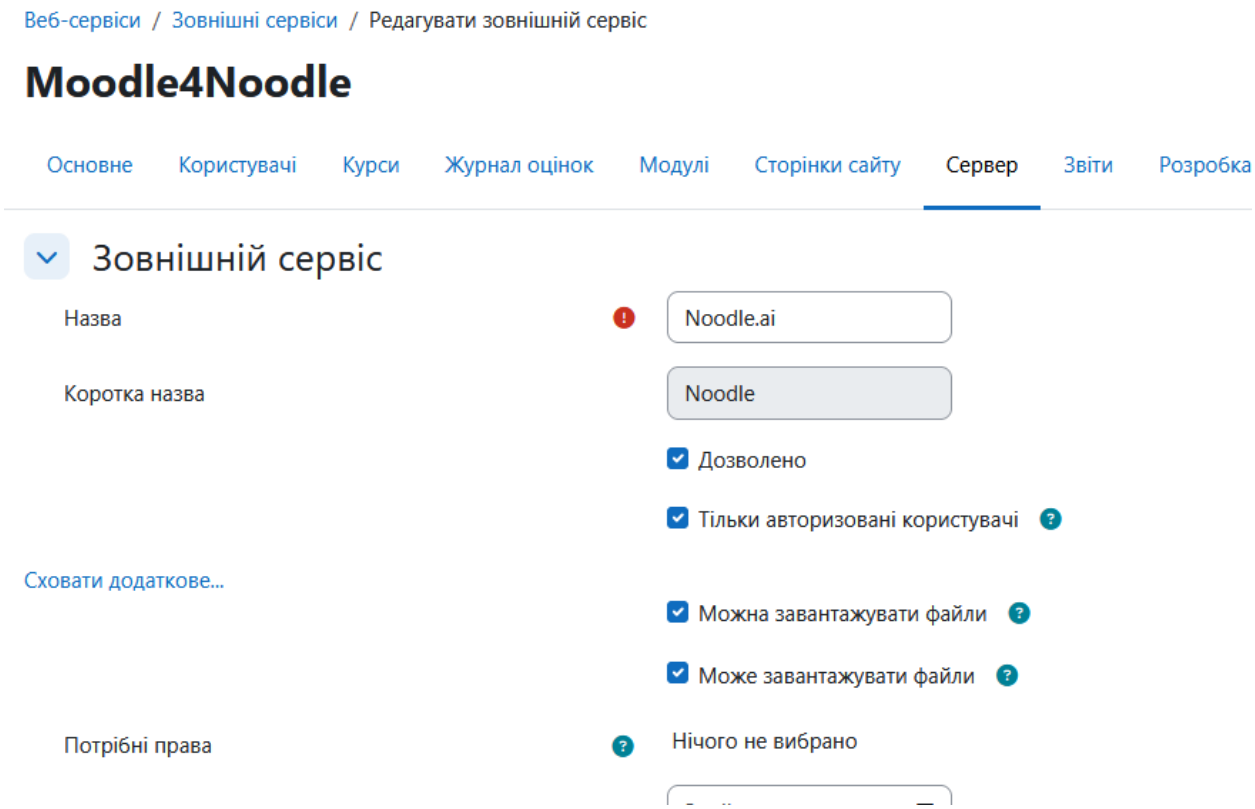


Рис. 4.6. Реєстрація зовнішнього вебсервісу на платформі Moodle

3) Реєструю окремий сервісний обліковий запис (рис. 4.7);

4) Генерую постійний API-токен, який надалі прописую в ENV-файл для сервісу-воркера.

Така конфігурація формує чіткий контракт взаємодії: мікросервіси отримують стабільний інтерфейс доступу до даних, а LMS залишається захищеною від несанкціонованих змін.

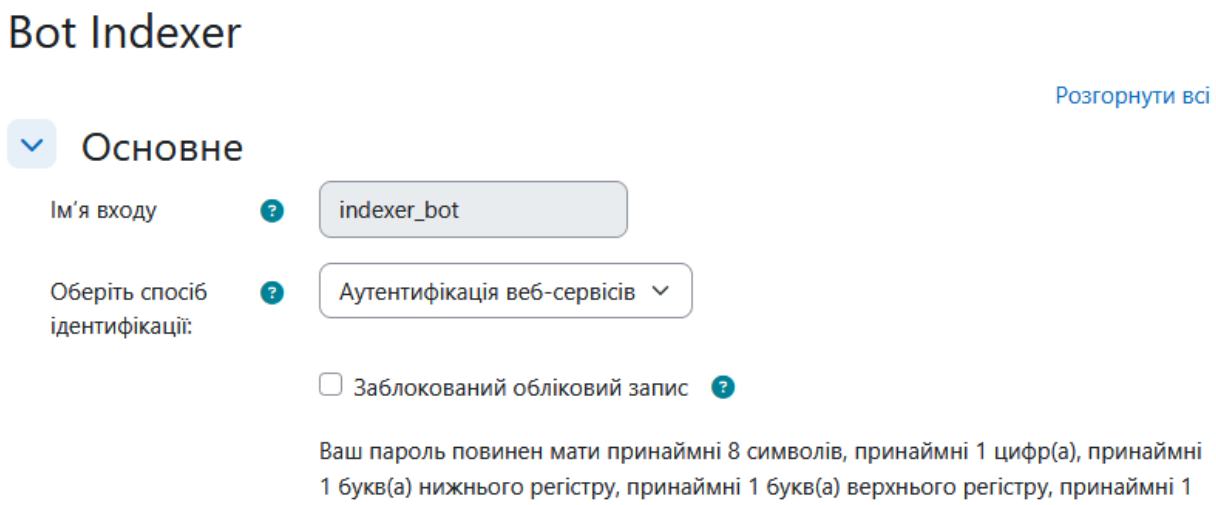


Рис. 4.7. Створення сервісного користувача на платформі Moodle

Повну діаграму архітектури та мікросервісної комунікації через оголошені API-ендпоїнти наведено на рисунку А.1.

4.4. Створення індексатора та налаштування векторної бази даних

Після налагодження джерел даних та API-інтерфейсів переходжу до реалізації серця системи RAG – механізму індексації та збереження знань. Для збереження семантичних векторів використовую базу даних Milvus. На відміну від традиційних реляційних баз, Milvus оперує поняттям «колекція» (collection). Проєктування схеми колекції (collection schema) здійснюю з урахуванням необхідності гібридного пошуку та фільтрації за метаданими.

Створюю колекцію `kb_embeddings`, визначивши таку структуру полів:

- `id` (`Int64`, primary key) – унікальний ідентифікатор запису, який генерую автоматично (`auto_id=True`);
- `doc_id` (`VarChar`) – унікальний ідентифікатор запису, який генерую вручну із даних із системи Moodle;
- `embedding` (`FloatVector`) – поле для зберігання векторного представлення тексту. Розмірність вектору встановлюю рівною 768, що відповідає вихідним параметрам моделі ембеддингу;
- `content` (`VarChar`, `max_length=65535`) – поле для збереження

оригінального тексту, щоб повертати читабельний текст агенту без необхідності звертатися до зовнішнього файлового сховища;

- `metadata` (JSON) – динамічне поле для зберігання атрибутів (ID курсу, назва, тип активності).

Для прискорення пошуку на полі `embedding` створюю індекс типу HNSW (Hierarchical Navigable Small World), адже цей алгоритм забезпечує оптимальний баланс між швидкістю пошуку та використанням оперативної пам'яті, що є важливим для локального розгортання. Як метрику відстані обираю косинусну подібність, оскільки вона найкраще відображає семантичну близькість текстових векторів (див. п. 2.3).

Сервіс індексації (воркер) реалізую мовою Python як постійно діючий фоновий процес. Його архітектура побудована на принципі інкрементального оновлення: система не повинна переіндексовувати всі файли щоразу, а лише ті, що були змінені або додані.

Логіку роботи скрипта організовую наступним чином:

1) воркер періодично опитує Moodle (`core_course_get_contents`), отримуючи дерево курсів та матеріалів. Для кожного файлу перевіряю його `timestamp` – час модифікації, порівнюючи зі збереженим;

2) якщо виявлено новий файл, воркер завантажує його та передає відповідному Document-адаптеру LangChain;

3) текст разом з метаданими далі розбивається на фрагменти згідно описаного раніше алгоритму (див. п. 3.4).

Для запровадження metadata-padded embeddings реалізую функцію-трансформер, яка збагачує текст кожного чанка документа контекстуальними даними. Повний код функції наведено в додатку Б.

Збагачені рядки тексту передаються на ембеддинг-модель для генерації векторів. Важливо зазначити, що у поле `content` записую оригінальний текст, щоб при генерації відповіді LLM бачила чистий контент, а не технічні заголовки й могла надати повну відповідь користувачу.

Індексації також піддаються дані із FAQ Store, отримуваних шляхом

API-виклику (рис. 4.8), після чого вони проходять раніше описаний пайплайн (див. п. 3.5).

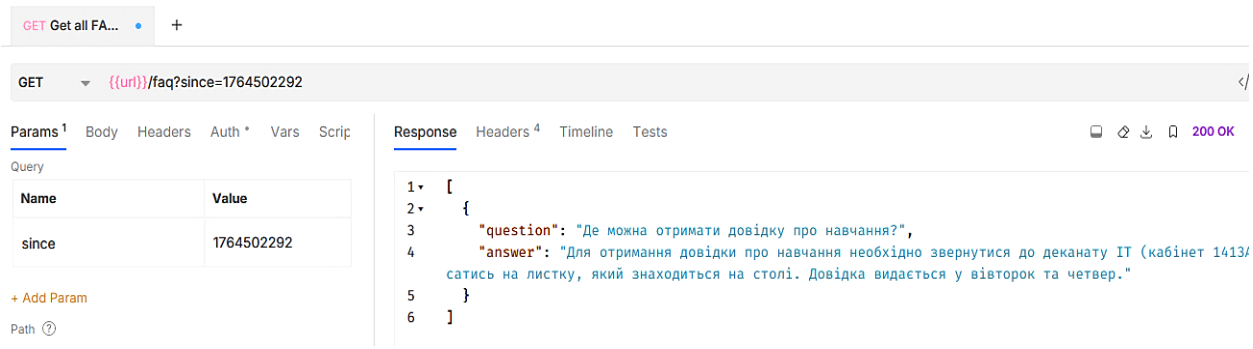


Рис. 4.8. Отримання оновлених наборів питання-відповідь через FAQ API

4.5. Створення соціального модуля та зберігання графу активностей

Для зберігання соціального графа використовую дві основні структури даних Redis, що дозволяє розділити рейтингові показники та персональну інформацію:

1) для кожного студента створюю запис за ключем `social:profile:{user_id}`. У цьому об'єкті зберігаю лише публічні контактні дані, ім'я та часову мітку останньої активності `last_seen`;

2) рейтинги експертизи – це впорядкована множина для кожної навчальної активності (лабораторної роботи чи тесту), яку створюю за ключем `activity:{id}:experts`. У цій структурі зберігаються ідентифікатори студентів, а як `score` використовується їхній нормалізований бал успішності.

Наповнення бази даних покладаю на фоновий сервіс Social Worker, алгоритм роботи якого детально описаний в пункті 3.5.

Алгоритм підбору ментора реалізовано у вигляді інструменту, який викликається агентом. Головна мета інструменту – знайти активності за визначеною агентом тематикою і повернути профілі студентів, для яких пізніше збалансувати академічну успішність та доступність для комунікації (лістинг 4.13). Повний код функції наведено в додатку В.

Лістинг 4.13. Файл `docker-compose.yml` для розгортання Moodle в контейнері Docker

```
@tool
def find_study_buddy(topic: str):
    emb_results = requests.post(
        f"{EMBEDDING_STORE_URL}/activities/search",
        json={
            "query": topic,
            "filter": {"type": "activity"},
            "top_k": 1,
        },
    ).json().get("results", [])

    best_match = emb_results[0]
    activity_id = best_match['metadata']['activity_id']
    activity_name = best_match['metadata'].get('name', 'Unknown
Activity')

    experts = requests.get(
        f"{STUDY_BUDDY_STORE_URL}/experts/{activity_id}",
    ).json()

    result_text = f"Found mentors for '{activity_name}':\n{
experts}"
```

За такого підходу користувач отримає рекомендацію звернутися до того, хто з найбільшою ймовірністю відповість прямо зараз, зберігаючи при цьому високий стандарт академічних знань.

4.6. Створення вебінтерфейсу взаємодії

Для розробки інтерфейсу обираю екосистему React і для забезпечення надійності коду використовую мову-розширення для JavaScript – TypeScript. Сувора статична типізація дозволяє виявляти помилки передачі даних між компонентами ще на етапі компіляції. У якості інструмента збірки застосовую Vite, адже він під час розробки забезпечує миттєвий запуск локального сервера та надшвидку гарячу заміну модулів (HMR).

Структуру додатку організовую навколо головного контейнера – макету чату, який управляє потоком повідомлень. Інтерфейс декомпозується

на функціональні компоненти (рис. 4.9).

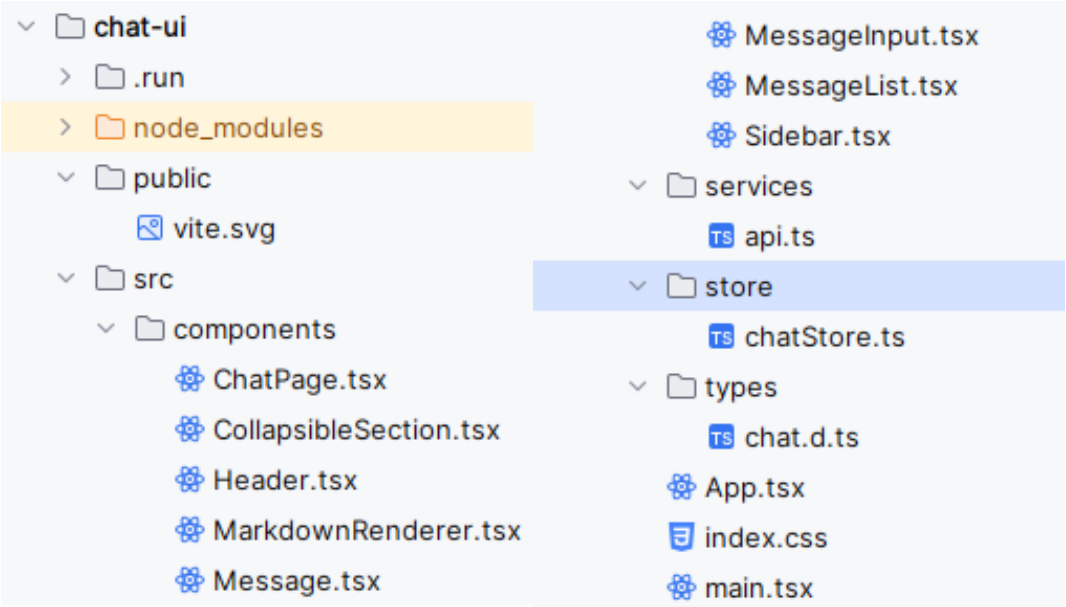


Рис. 4.9. Список компонентів та модулів проєкту фронтенду

Особливістю розробленого інтерфейсу є здатність відображати гетерогенний контент. Відповідь агента не є простим рядком тексту, а потоком, що містить як форматований текст, так і технічні блоки виконання команд, таблиці, заголовки, стилі, а також згорткові блоки (рис. 4.10). Відображення їх користувачу забезпечує рівень прозорості та дозволяє зрозуміти підґрунтя відповідей.

Оскільки генерація відповіді LLM може займати певний час, важливим є реалізація стримінгу відповіді за наступним алгоритмом:

1. При отриманні нового фрагмента тексту від сервера, він миттєво додається до стану поточного повідомлення, що створює ефект друкарської машинки, коли текст з'являється на екрані в реальному часі;
2. У цей же час парсер перевіряє потік на наявність спеціальних маркерів початку/кінця виклику інструментів, динамічно оновлюючи структуру повідомлення в DOM-дереві (Document Object Model).

Для управління даними та станом на SPA-клієнті використовую хуки React `useState`, `useEffect`, `useRef`.

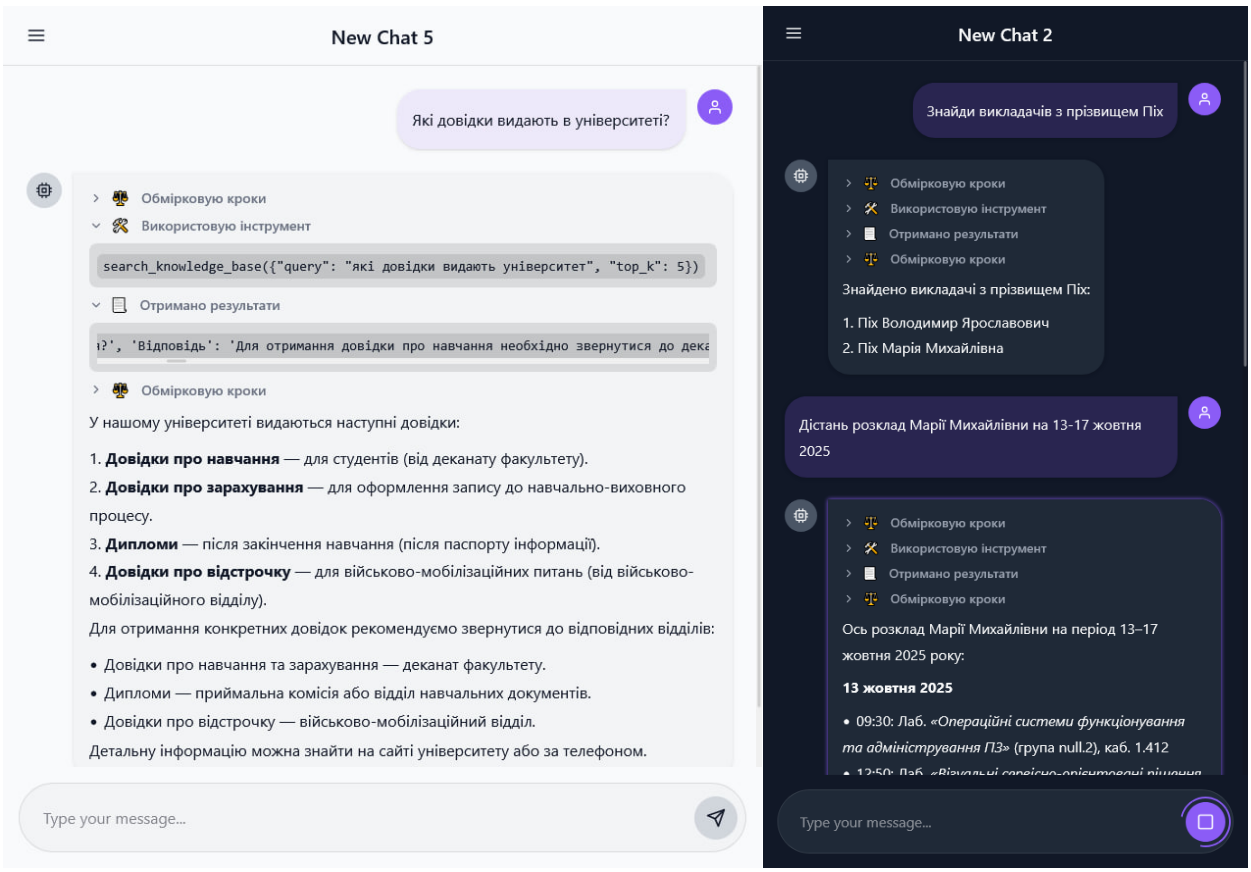


Рис. 4.10. Вигляд вебінтерфейсу користувача (світла тема, ландшафтний режим макету зліва; темна тема, мобільний режим справа)

4.7. Тестування семантичного та соціального пошуку

Заключним етапом реалізації будь-якої системи є верифікація працездатності цієї системи та оцінка ефективності запропонованих алгоритмів. Тестування проводилося за методикою «чорної скриньки» на рівні інтеграції сервісів (end-to-end testing), перевіряючи коректність відповідей системи на запити природною мовою.

Перед початком здійснено повне розгортання системи у середовищі Docker та зовнішніх сервісів:

1. Вебінтерфейс чат-боту (фронтенд) для взаємодії із системою.
2. Thinking & Tool Use Model Proxy для координації взаємодії фронтенду, великої мовної моделі та бекенд-сервісів.
3. Сервер LM Studio із попередньо завантаженими моделями: Qwen 3

4B Thinking (версія 2507) та EmbeddingGemma 300M (рис. 4.11).

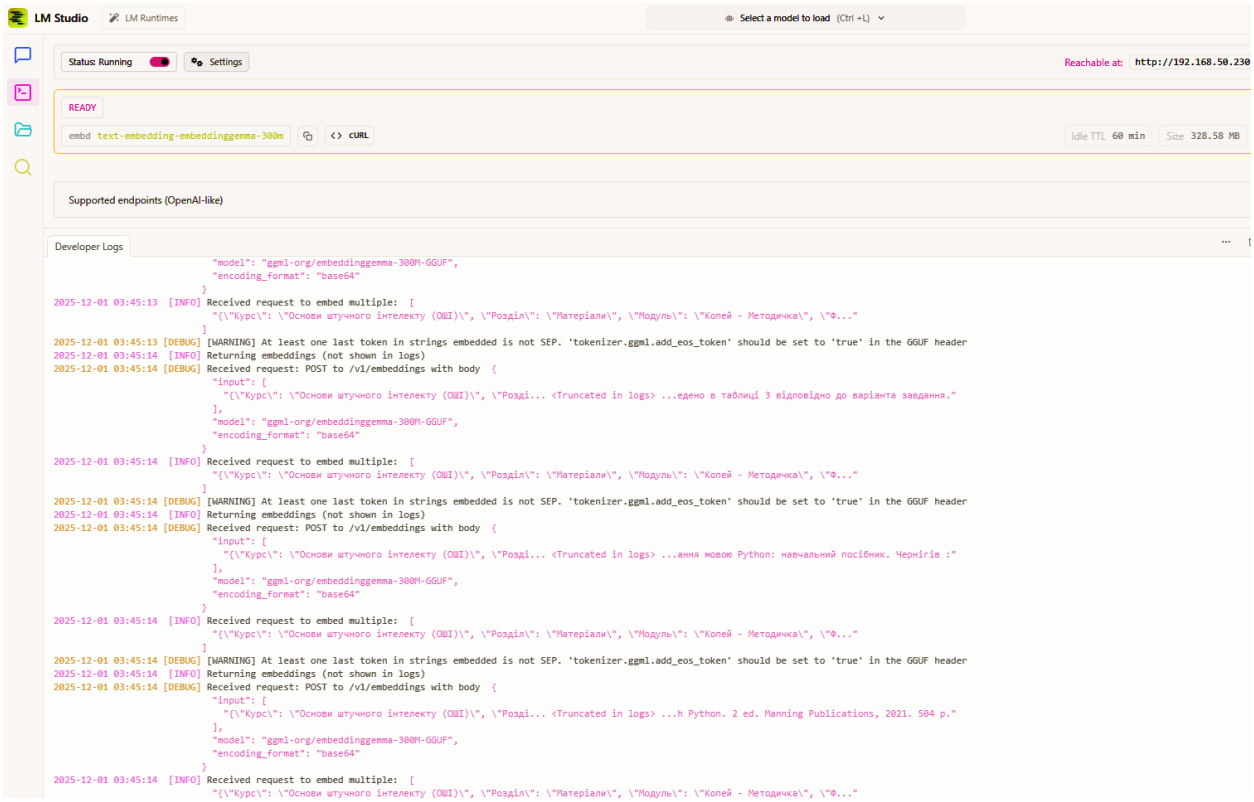


Рис. 4.11. Процес векторизації тексту (файлів, завантажених з Moodle, та питань-відповідей) в консолі LM Studio

4. Локальну копію LMS Moodle, наповнену синтетичними даними (завантажено декілька курсів, кожен з яких містить набір PDF-файлів, презентацій, документів Microsoft Word, сторінок HTML та посилань на зовнішні ресурси; додано акаунти студентів із різними патернами успішності; імітовано результати виконання завдань, щоб сформувати базу академічних компетенцій).

5. Caddy сервер із конфігурацією реверсивного проксі для Moodle для надання зовнішнього доступу через HTTPS.

6. FAQ Store із JSON-файлами для наповнення/оновлення бази короткої інформації у форматі запитань-відповідей.

7. Сховище Embedding Store для збереження векторних даних семантичного пошуку по файлах та метаданих.

8. Study Buddy Store для кешування академічної активності студентів.

9. RAG Indexer, який виконує ETL-пайплайн векторизації даних на Moodle та у FAQ Store (рис. 4.12).

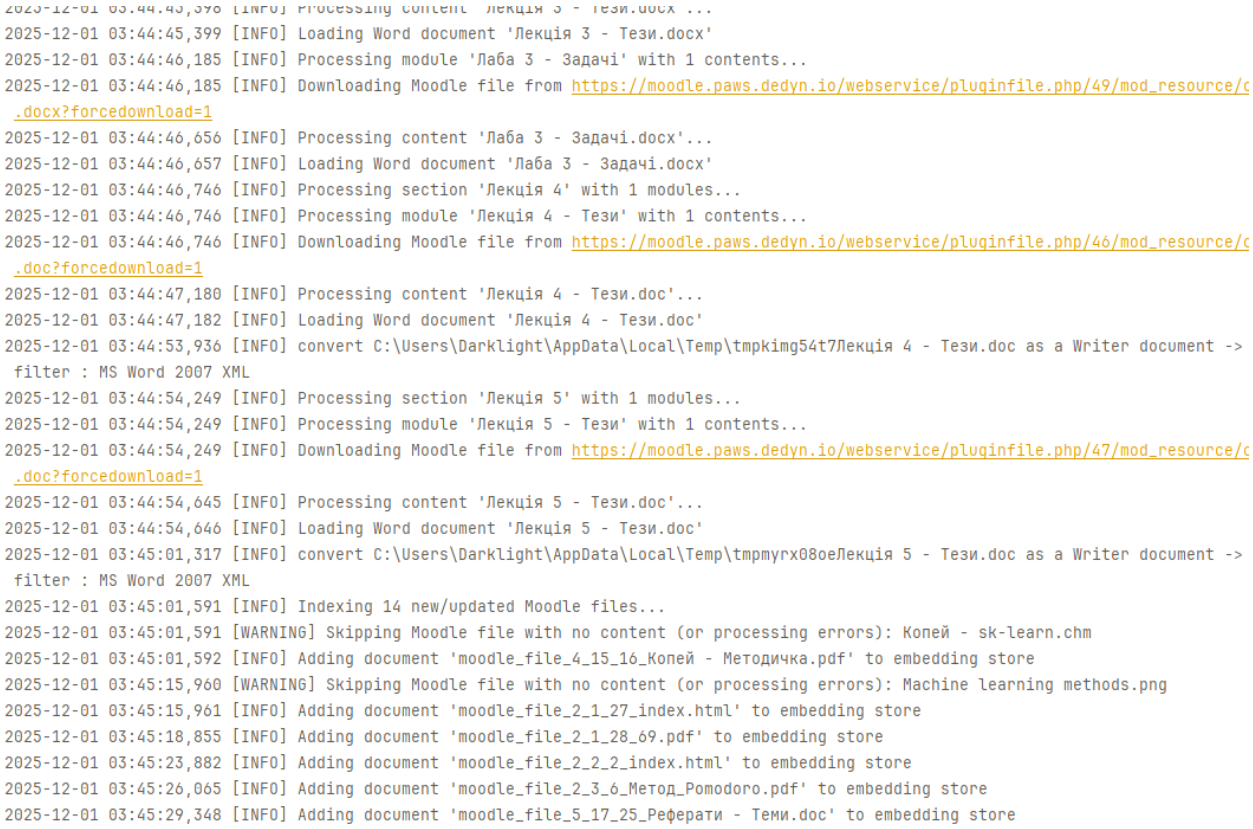
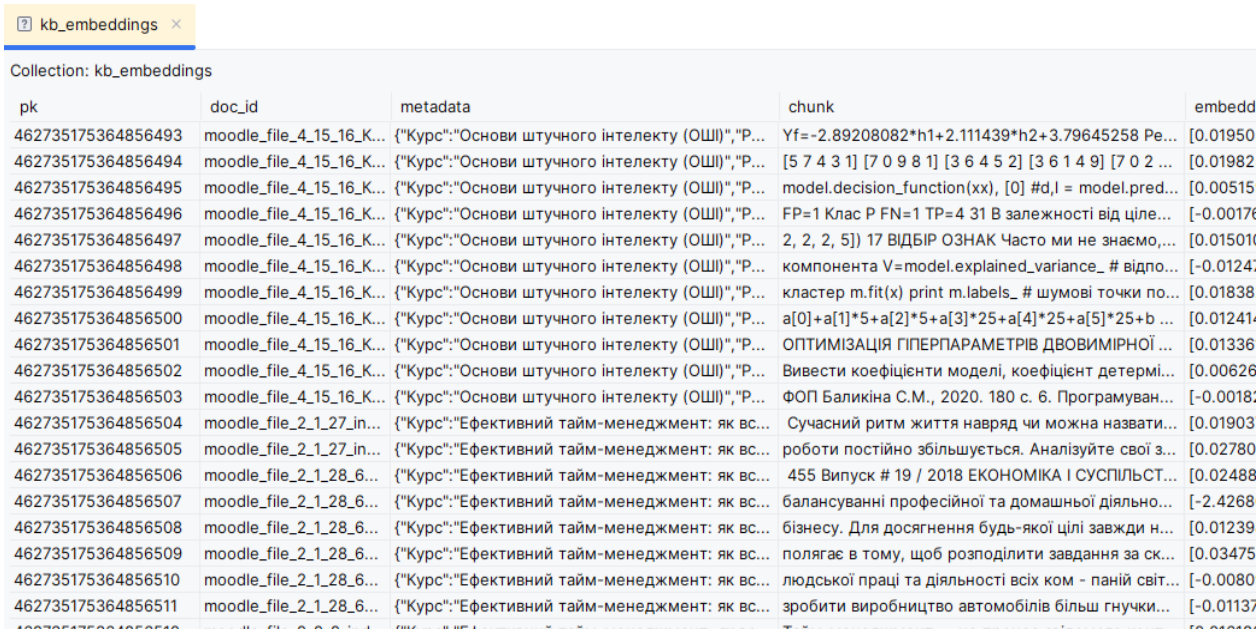


Рис. 4.12. Процес індексації деяких завантажених на LMS Moodle файлів

10. Study Buddy Worker, який виконує ETL-пайплайн підготовки та синхронізації сховища академічних даних.

Очікую на завершення процесів індексації. Проіндексовані файли Moodle видно в колекції БД Milvus на рисунку 4.13.

Для проведення експериментального дослідження швидкодії та точності роботи розробленої системи сформовано набір із тестових сценаріїв взаємодії (use case). Кожен сценарій спроєктовано таким чином, щоб перевірити роботу окремого архітектурного компонента системи та оцінити ефективність у порівнянні з традиційними методами пошуку інформації в LMS Moodle.



kb_embeddings

Collection: kb_embeddings

pk	doc_id	metadata	chunk	embedd
462735175364856493	moodle_file_4_15_16_K...	{"Курс":"Основи штучного інтелекту (ОШІ)","Р...	Yf=-2.89208082*h1+2.111439*h2+3.79645258 Ре...	[0.01950
462735175364856494	moodle_file_4_15_16_K...	{"Курс":"Основи штучного інтелекту (ОШІ)","Р...	[5 7 4 3 1] [7 0 9 8 1] [3 6 4 5 2] [3 6 1 4 9] [7 0 2 ...	[0.01982
462735175364856495	moodle_file_4_15_16_K...	{"Курс":"Основи штучного інтелекту (ОШІ)","Р...	model.decision_function(xx), [0] #d,l = model.pred...	[0.00515
462735175364856496	moodle_file_4_15_16_K...	{"Курс":"Основи штучного інтелекту (ОШІ)","Р...	FP=1 Клас P FN=1 TP=4 31 В залежності від ціле...	[-0.0017
462735175364856497	moodle_file_4_15_16_K...	{"Курс":"Основи штучного інтелекту (ОШІ)","Р...	2, 2, 2, 5]) 17 ВІДБІР ОЗНАК Часто ми не знаємо,...	[0.01501
462735175364856498	moodle_file_4_15_16_K...	{"Курс":"Основи штучного інтелекту (ОШІ)","Р...	компонента V=model.explained_variance_ # відпо...	[-0.0124
462735175364856499	moodle_file_4_15_16_K...	{"Курс":"Основи штучного інтелекту (ОШІ)","Р...	кластер m.fit(x) print m.labels_ # шумові точки по...	[0.01838
462735175364856500	moodle_file_4_15_16_K...	{"Курс":"Основи штучного інтелекту (ОШІ)","Р...	a[0]+a[1]*5+a[2]*5+a[3]*25+a[4]*25+a[5]*25+b ...	[0.01241
462735175364856501	moodle_file_4_15_16_K...	{"Курс":"Основи штучного інтелекту (ОШІ)","Р...	ОПТИМІЗАЦІЯ ГІПЕРПАРАМЕТРІВ ДВОВИМІРНОЇ ...	[0.01336
462735175364856502	moodle_file_4_15_16_K...	{"Курс":"Основи штучного інтелекту (ОШІ)","Р...	Вивести коефіцієнти моделі, коефіцієнт детермі...	[0.00626
462735175364856503	moodle_file_4_15_16_K...	{"Курс":"Основи штучного інтелекту (ОШІ)","Р...	ФОП Баликіна С.М., 2020. 180 с. 6. Програмуван...	[-0.0018
462735175364856504	moodle_file_2_1_27_in...	{"Курс":"Ефективний тайм-менеджмент: як вс...	Сучасний ритм життя навряд чи можна назвати...	[0.01903
462735175364856505	moodle_file_2_1_27_in...	{"Курс":"Ефективний тайм-менеджмент: як вс...	роботи постійно збільшується. Аналізуйте свої з...	[0.02780
462735175364856506	moodle_file_2_1_28_6...	{"Курс":"Ефективний тайм-менеджмент: як вс...	455 Випуск # 19 / 2018 ЕКОНОМІКА І СУСПІЛЬСТ...	[0.02488
462735175364856507	moodle_file_2_1_28_6...	{"Курс":"Ефективний тайм-менеджмент: як вс...	балансуванні професійної та домашньої діяльно...	[-2.4268
462735175364856508	moodle_file_2_1_28_6...	{"Курс":"Ефективний тайм-менеджмент: як вс...	бізнесу. Для досягнення будь-якої цілі завжди н...	[0.01239
462735175364856509	moodle_file_2_1_28_6...	{"Курс":"Ефективний тайм-менеджмент: як вс...	полягає в тому, щоб розподілити завдання за ск...	[0.03475
462735175364856510	moodle_file_2_1_28_6...	{"Курс":"Ефективний тайм-менеджмент: як вс...	людської праці та діяльності всіх ком - паній світ...	[-0.0080
462735175364856511	moodle_file_2_1_28_6...	{"Курс":"Ефективний тайм-менеджмент: як вс...	зробити виробництво автомобілів більш гнучки...	[-0.0113

Рис. 4.13. Колекція у векторній базі Milvus, де видно метадані з Moodle, проіндексований вміст файлів та початок їх векторного представлення

Сценарій 1. Користувач намагається з'ясувати процедурні питання, зокрема, які документи видає університет для засвідчення факту навчання, але надає нечіткий запит «Які довідки видають в університеті?».

Передумова: у базі питань-відповідей є записи попередніх запитів студентів, які можна використати для отримання відповіді. Мета: перевірка коректності роботи модуля доступу до статичної бази знань (FAQ Store) та здатності системи надавати точні інструкції на основі регламентних документів.

Чат-бот обмірковує кроки для вирішення проблеми користувача та перевіряє надані йому інструменти й обирає пошук по університетській базі знань (Embedding Store). Пошук 5 найбільш відповідних записів дає корисні результати й модель використовує їх для генерування відповіді. Відповідь відображається в інтерфейсі користувача (рис. 4.14).

Сценарій 2. Користувач намагається знайти навчальний курс, але не пам’ятає точної назви дисципліни, тому формує запит на кшталт «Чи є в нас дисципліна big data?».

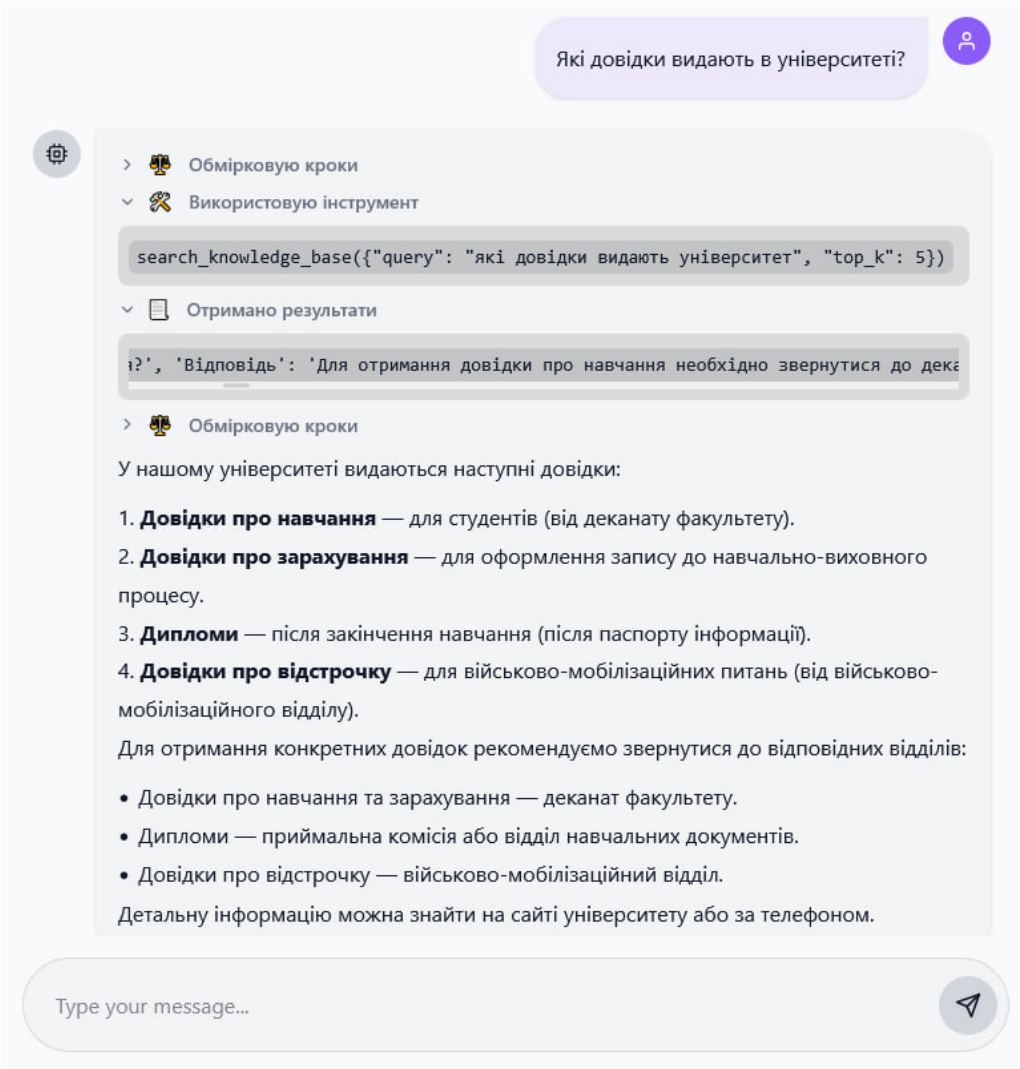


Рис. 4.14. Відповідь чат-боту на основі пошуку в базі частих запитань

Передумова: у базі знань є проіндексовані документи із системи керування навчанням Moodle. Особливістю сценарію є те, що в назвах курсів не фігурує точна фраза «Big Data», а використовуватись синонімічні поняття (дисципліна на Moodle: «Великі дані та великі масиви даних»), що дозволить оцінити ефективність векторного пошуку та здатність моделі ембеддингу встановлювати семантичні зв'язки між поняттями, що є недоступним для звичайного пошуку за ключовими словами.

Чат-бот перевіряє надані йому інструменти й обирає пошук по університетській базі знань. Пошук найбільш відповідних записів дає результати із завантажених файлів, семантична сутність яких є найбільш близькою до запиту. Далі модель генерує відповідь користувачу (рис. 4.15).

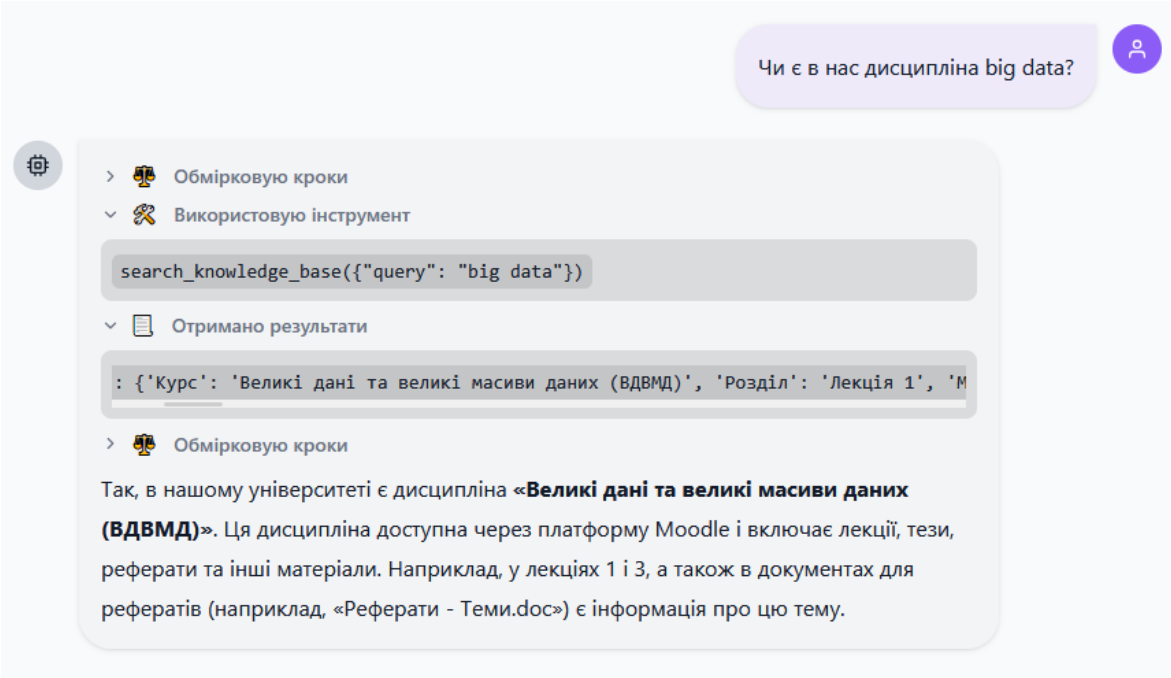


Рис. 4.15. Відповідь чат-боту на основі пошуку по файлах із системи дистанційного навчання Moodle

Отже, використання збагачення метаданими підвищує семантичну диференціацію документів й дозволяє системі знаходити відповідні документи та інформацію ефективніше.

Сценарій 3. Користувач шукає допомогу із конкретної академічної проблеми: «знайти ментора для допомоги із завданням по рекурсії в Java».

Передумова: інформація про академічну успішність була опрацьована й збережена в Redis-БД сервісу. Тут мета – комплексна перевірка гібридної архітектури. Система повинна спочатку ідентифікувати відповідну лабораторну роботу через RAG, а потім звернутися до соціального графа в Redis для пошуку компетентних студентів, відфільтрувавши їх за статусом доступності («онлайн»).

Система повертає трьох кандидатів із Study Buddy Store на менторство, а мовна модель рекомендує, до кого звернутися (рис. 4.16). Варто зауважити, що в Embedding Store немає «Recursion in Java», але є «Рекурсивні алгоритми в ООП». Незважаючи на це, система коректно ідентифікувала студентів, які мають найвищі бали в системі з відповідних завдань.

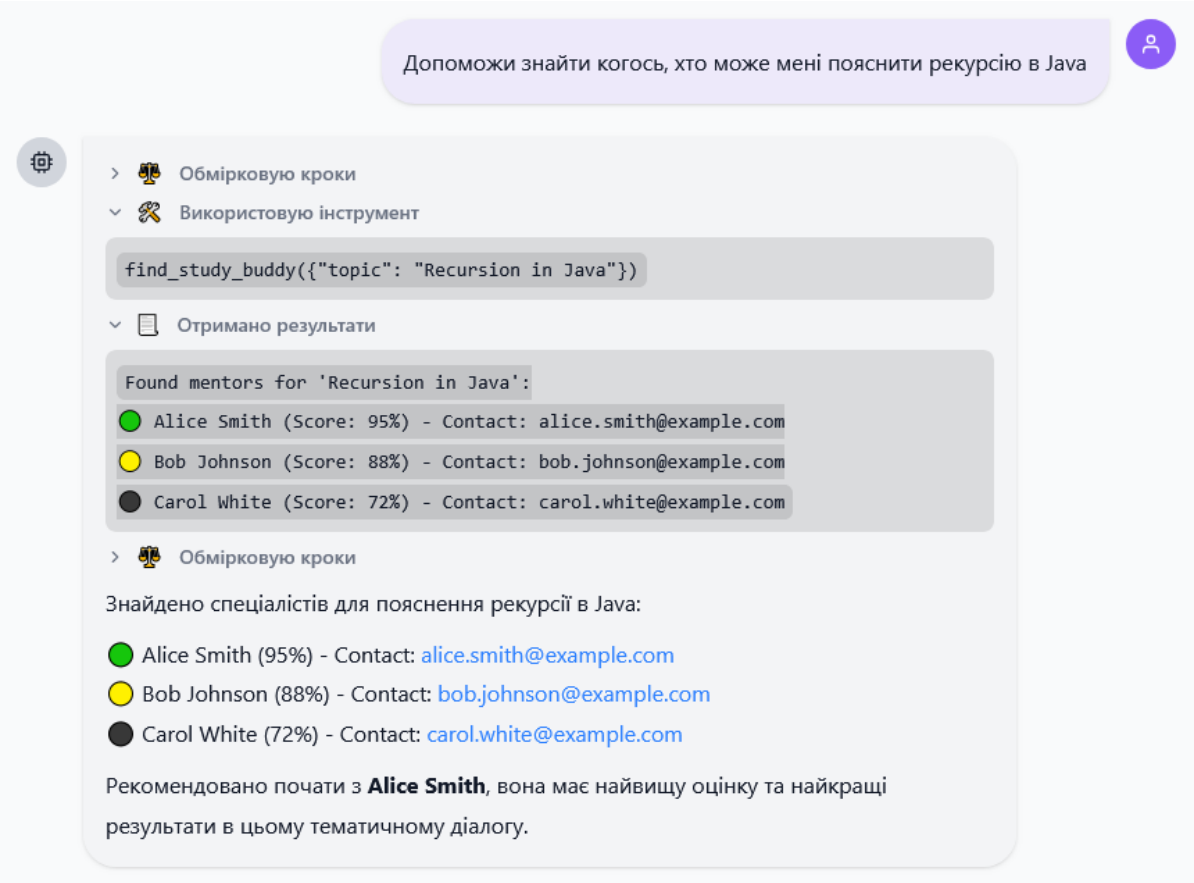


Рис. 4.16. Відповідь чат-боту на основі інформації із Study Buddy Store

Окрім трьох основних сценаріїв розроблено ще два додаткові для тестування складніших запитів, які вимагають багатокрокових дій.

Сценарій 4. Користувач запитує роз'яснення щодо різниці, призначення та способів отримання довідок форми 9 та форми 20. Такий запит націлений на перевірку когнітивних здібностей LLM. На відміну від простого пошуку (Сценарій 1), тут модель повинна знайти два різні документи, проаналізувати їхній зміст та синтезувати порівняльну характеристику.

Сценарій 5. Користувач шукає визначення специфічного терміну «філософія науки» серед матеріалів, завантажених на платформу Moodle (PDF-лекції, методичні вказівки, презентації). Тест демонструє здатність системи знаходити вузькоспеціалізовану інформацію всередині великого набору проіндексованих файлів у векторній базі даних.

Протягом тестування сценаріїв виконано оцінку зручності та швидкості отримання відповіді у порівнянні зі звичним пошуком. Методика виконання

вимірювання часу початку та кінця наступна:

- для ручного режиму – час від моменту відкриття браузера (або вкладки Moodle) до моменту, коли очі користувача знайшли потрібну інформацію на екрані. Враховує логін, навігацію по меню курсів, завантаження файлу, відкриття файлу, пошук;
- для чат-бота – час від початку введення запиту до повного завершення генерації відповіді.

Результати вимірювання часу виконання зазначених сценаріїв у ручному режимі та з використанням розробленої системи наведено в табл. 4.1. Видно, що використання розробленої системи дозволяє наблизити інформацію до користувача й скоротити час пошуку. Найбільший приріст ефективності спостерігається у сценарії пошуку соціальних контактів, адже ручний пошук ментора є неструктурованим процесом із високим рівнем невизначеності, тоді як алгоритм системи виконує запит до бази даних Redis за фіксований час.

Таблиця 4.1

Варіанти використання та емпіричні дані часу на отримання інформації

Опис варіанту використання	**Час виконання**		**Відносний приріст, разів**
	Вручну (мінімум), с	**Через чат-бот (діапазон), с**	
Сценарій 1: пошук адміністративної інформації	100	10-15	8
Сценарій 2: семантичний пошук дисциплін	20-40	15-20	1.7
Сценарій 3: пошук ментора	180	25-30	6.5
Сценарій 4: аналітичне порівняння документів	200	10-15	16.3
Сценарій 5: глибинний пошук у навчальному контенті	40	12-30	1.9

У сценаріях семантичного пошуку економія часу досягається завдяки

усуненню необхідності завантажувати та відкривати файли чи виконувати ручний пошук за ключовими словами, який часто дає хибні результати при відсутності точного співпадіння.

Отже, використання поєднання семантичних алгоритмів пошуку та індексації, а також графових представлень академічних даних дозволяє побудувати БД, якою асистенти на основі LLM здатні ефективно користуватися.

4.8. Рекомендації з впровадження

На основі результатів розробки та тестування прототипу сформовано комплекс рекомендацій щодо інтеграції подібної системи в IT-інфраструктуру вищого навчального закладу. Демонстраційний прототип розгорнуто в локальному середовищі на базі Docker Compose та деяких зовнішніх сервісів. Для переведення системи у режим промислової експлуатації необхідно забезпечити ряд вимог системи:

1. Апаратні ресурси для LLM. У разі використання локальних моделей (Qwen, Gemma чи інших) критичним є виділення сервера з графічним прискорювачем (GPU) для забезпечення низької латентності інференсу при одночасному зверненні десятків користувачів. Альтернативою є використання корпоративної підписки на OpenAI API, що знімає вимоги до локального «заліза», але потребує бюджетування операційних витрат.

2. Оркестрація контейнерів. При масштабуванні на весь університет рекомендовано перехід від Docker Compose до Docker Swarm/Kubernetes, адже остання платформа дозволить налаштувати автоматичне масштабування кількості подів агента залежно від навантаження.

3. Кешування даних. Для оптимізації роботи соціального модуля та зниження навантаження на Moodle рекомендується налаштувати час життя кешу (TTL) у Redis залежно від динаміки змін.

Для забезпечення захисту даних та зручності користувачів необхідна

єдина точка входу (SSO). Можлива інтеграція з корпоративною системою університету (Google Workspace), що дозволить студентам входити в чат-бот, використовуючи свої університетські облікові записи.

Впровадження соціального модуля торкається чутливих даних про успішність студентів, що вимагає суворого дотримання нормативних вимог, таких як принцип добровільності – тобто функціонал пошуку ментора має працювати виключно за принципом явної згоди. Студент повинен самостійно активувати опцію «Я хочу бути ментором» у налаштуваннях, дозволяючи системі відображати його контактні дані іншим користувачам. Заохочення до менторства можна створювати за рахунок внутрішньої платформи нарахування додаткових балів.

Дотримання цих рекомендацій дозволить нівелювати технічні ризики та забезпечити органічну інтеграцію інтелектуальної системи в освітній процес.

4.9. Висновки до розділу

У четвертому розділі здійснено повний цикл програмної реалізації гібридної інтелектуальної системи, починаючи від налаштування середовища розробки і закінчуючи інтеграційним тестуванням модулів. Розроблений програмний комплекс повністю відповідає сформульованим у попередніх розділах архітектурним вимогам, демонструючи високий рівень модульності та масштабованості.

За результатами етапу реалізації можна зробити наступні висновки. Вибір мікросервісної архітектури виправдав себе на практиці. Розділення системи на незалежні сервіси дозволило ізолювати ресурсномісткі процеси обробки даних від інтерфейсу користувача, забезпечивши стабільність роботи системи навіть під час пікових навантажень на базу даних. Використання платформи контейнеризації Docker та інструменту оркестрації Docker Compose дозволило вирішити проблему конфлікту залежностей. Створена

інфраструктура є повністю відтворюваною: розгортання всього комплексу сервісів, баз даних (Milvus, Redis) та моделей ШІ здійснюється незалежно, що значно спрощує подальшу експлуатацію та демонстрацію.

Проведене тестування підтвердило працездатність ключових алгоритмічних рішень та їх здатність ефективно розрізняти контекст навчальних матеріалів, успішно фільтрувати кандидатів, забезпечуючи пріоритет доступним та компетентним менторам. Реалізація клієнтської частини на React із застосуванням потокової передачі даних забезпечила високу чутливість інтерфейсу. Візуалізація процесу «мислення» агента та структурованих карток даних перетворює складну серверну логіку на зрозумілий та зручний інструмент для студента.

Таким чином, створена система є функціонально завершеним прототипом (MVP), який готовий до демонстрації можливостей генеративного ШІ та соціального нетворкінгу в освітньому процесі. Програмна реалізація створює надійний фундамент для проведення подальших досліджень щодо впливу таких систем на успішність студентів.

ВИСНОВКИ

У магістерській роботі вирішено актуальне науково-практичне завдання підвищення ефективності навчальної діяльності студентів шляхом розробки гібридної інтелектуальної системи, що поєднує методи семантичного пошуку інформації та алгоритмічну оркестрацію соціальних зв’язків.

У ході дослідження проведено комплексний аналіз сучасного стану цифровізації вищої освіти, за результатами якого констатовано успішний розвиток первинної цифрової інфраструктури, але водночас виявлено суттєву фрагментацію університетських екосистем. Встановлено, що ключовими бар’єрами для подальшого зростання ефективності навчання є розрив між статичними освітніми ресурсами та живим спілкуванням, складність доступу до «мовчазного знання» носіїв досвіду, а також ізольованість існуючих рішень на базі штучного інтелекту від соціального контексту університету. Обґрунтовано, що подолання цих проблем можливе лише через інтеграцію доступу до фактичної інформації та соціальної допомоги в межах єдиної платформи.

На основі аналізу архітектурних та методологічних прогалин існуючих систем доведено неефективність використання великих мовних моделей загального призначення для специфічних університетських задач. Натомість обґрунтовано доцільність використання архітектури RAG (Retrieval-Augmented Generation) для забезпечення фактологічної точності генерації відповідей без необхідності постійного донавчання моделей. Розроблено концепцію використання інтелектуального агента як соціального посередника, який, аналізуючи «академічний цифровий слід», здатний автоматизовано формувати компетентні навчальні пари, вирішуючи тим самим проблему локалізації експертизи.

У рамках інженерного проєктування розроблено мікросервісну архітектуру системи, яка дозволила ефективно розділити ресурсномісткі

процеси семантичної обробки тексту та чутливі до затримок операції пошуку соціальних контактів. Запропоновано та впроваджено методику індексації навчальних матеріалів із використанням підходу для збереження контексту при векторизації документів (metadata-padded embeddings). Спроєктовано та програмно реалізовано алгоритми соціального матчингу на базі швидкого сховища Redis, що дозволило миттєво ідентифікувати компетентних менторів на основі їхньої успішності та статусу доступності.

Здійснено повний цикл програмної реалізації прототипу системи (MVP) з використанням контейнеризації Docker, що забезпечило модульність, масштабованість та відтворюваність середовища розробки. Створено клієнтський вебінтерфейс, який реалізує принцип універсального інтерфейсу та візуалізує процес прийняття рішень інтелектуальним агентом. Проведене інтеграційне та сценарне тестування підтвердило працездатність розроблених алгоритмів, зокрема високу точність семантичного пошуку релевантних навчальних матеріалів та коректність пріоритезації менторів, а виконання замірів швидкості – зручність взаємодії за принципом «єдиного вікна» та приємний користувацький досвід.

Отже, досягнуто мети роботи – створено надійний інструментарій, що сприяє швидкому подоланню когнітивних бар’єрів у навчанні та готовий до впровадження в освітній процес.

Вихідний код продукту, розробленого в рамках дипломної роботи, опубліковано на GitHub за посиланням: https://github.com/MikronT/NoodleAI.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Баніт О. Peer-to-Peer: від методу навчання до освітньої технології. Actual Problems in the System of Education: General Secondary Education Institution – Pre-University Training – Higher Education Institution. № 2. С. 264–268. DOI:10.18372/2786-5487.1.16604.

2. Галюцинація (штучний інтелект). Вікіпедія. 16.10.2025. URL: https://w.wiki/GV5v (дата звернення: 04.12.2025).

3. Застосунок дванадцяти факторів. Вікіпедія. 24.11.2025. URL: https://w.wiki/GX2Q (дата звернення: 06.12.2025).

4. Конструювання підказок. Вікіпедія. 08.10.2025. URL: https://w.wiki/GV5z (дата звернення: 04.12.2025).

5. Математична модель. Вікіпедія. 08.11.2025. URL: https://w.wiki/GVhN (дата звернення: 04.12.2025).

6. Немашкало К. Р. Теоретико-методичні основи структуризації соціального капіталу. Підприємництво та інновації. № 25. С. 28–32. DOI:10.32782/2415-3583/25.4.

7. Нохріна І. С. Вплив соціальних мереж на соціальну успішність студентської молоді. Соціальна робота та психологія: освіта і наука. № 1. С. 46–53. DOI:10.32782/3041-1351/2024-1-7.

8. Поліщук Н. В., Бугаєнко Т. І., Лемешева Н. В. Підвищення якості вищої освіти за допомогою цифрових технологій та дистанційного навчання для здобувачів вищої освіти в Україні. 20.12.2024. DOI:10.5281/ZENODO.14537287.

9. Туришева О. О., Дзикович О. В. Регулювання використання генеративного ші здобувачами вищої освіти: європейський і український досвід. Слобожанський науковий вісник. Серія: Філологія. № 11. С. 173–179. DOI:10.32782/philspu/2025.11.29.

10. Фасад. URL: https://refactoring.guru/uk/design-patterns/facade (дата звернення: 04.12.2025).

11. A Comprehensive Hybrid Search Guide. URL: https://www.elastic.co/what-is/hybrid-search (дата звернення: 05.12.2025).

12. Agarwal N. K., Marouf L. N. Initiating Knowledge Management in Colleges and Universities: A template. International Journal of Knowledge Content Development & Technology. Вип. 4, № 2. С. 67–95. DOI:10.5865/IJKCT.2014.4.2.067.

13. Alshammary F. M., Alhalafawy W. S. Digital Platforms and the Improvement of Learning Outcomes: Evidence Extracted from Meta-Analysis. Sustainability. Вип. 15, № 2. С. 1305. DOI:10.3390/su15021305.

14. Alves R. B. C., Pinheiro P. Factors Influencing Tacit Knowledge Sharing in Research Groups in Higher Education Institutions. Administrative Sciences. Вип. 12, № 3. С. 89. DOI:10.3390/admsci12030089.

15. erseco/alpine-moodle. Docker Hub. URL: https://hub.docker.com/r/erseco/alpine-moodle (дата звернення: 06.12.2025).

16. Euclidean distance. Wikipedia. 03.12.2025. URL: https://w.wiki/3rrM (дата звернення: 04.12.2025).

17. Expertise finding. Wikipedia. 28.08.2025. URL: https://w.wiki/GVpv (дата звернення: 05.12.2025).

18. Explain Like I’m Five | Don’t Panic! Reddit. URL: https://reddit.com/r/explainlikeimfive (дата звернення: 04.12.2025).

19. Fan Z., Beh L.-S. Knowledge sharing among academics in higher education: A systematic literature review and future agenda. Educational Research Review. Вип. 42, 02.2024. С. 100573. DOI:10.1016/j.edurev.2023.100573.

20. Function calling - OpenAI API. URL: https://platform.openai.com (дата звернення: 04.12.2025).

21. Gavrus C., Petre I. M., Lupșa-Tătaru D. A. The Role of e-Learning Platforms in a Sustainable Higher Education: A Cross-Continental Analysis of Impact and Utility. Sustainability. Вип. 17, № 7. С. 3032. DOI:10.3390/su17073032.

22. ggml-org/embeddinggemma-300M-GGUF. Hugging Face.

04.09.2025. URL: https://huggingface.co/ggml-org/embeddinggemma-300M-GGUF (дата звернення: 06.12.2025).

23. Grounding LLMs. Microsoft Community Hub. URL: https://techcommunity.microsoft.com/blog/fasttrackforazureblog/grounding-llms/3843857 (дата звернення: 04.12.2025).

24. Josephine Oranga. Tacit Knowledge Transfer and Sharing: Characteristics and Benefits of Tacit & Explicit Knowledge. Journal of Accounting Research, Utility Finance and Digital Assets. Вип. 2, № 2. С. 736–740. DOI:10.54443/jaruda.v2i2.103.

25. Knowledge cutoff. Wikipedia. 08.11.2025. URL: https://w.wiki/GV5x (дата звернення: 04.12.2025).

26. Koh A. W. L., Lee S. C., Lim S. W. H. The learning benefits of teaching: A retrieval practice hypothesis. Applied Cognitive Psychology. Вип. 32, № 3. С. 401–410. DOI:10.1002/acp.3410.

27. Kwayu S., Abubakre M., Lal B. The influence of informal social media practices on knowledge sharing and work processes within organizations. International Journal of Information Management. Вип. 58, 06.2021. С. 102280. DOI:10.1016/j.ijinfomgt.2020.102280.

28. Mishra S. Social networks, social capital, social support and academic success in higher education: A systematic review with a special focus on ‘underrepresented’ students. Educational Research Review. Вип. 29, 02.2020. С. 100307. DOI:10.1016/j.edurev.2019.100307.

29. Nabi G., Walmsley A., Mir M. та ін. The impact of mentoring in higher education on student career development: a systematic review and research agenda. Studies in Higher Education. Вип. 50, № 4. С. 739–755. DOI:10.1080/03075079.2024.2354894.

30. Parmar J. S., Mistry S. K., Micheal S. та ін. Peer Support for Improving Student Engagement and Learning Outcomes in Postgraduate Public Health and Health Sciences: A Qualitative Study. Education Sciences. Вип. 15, № 5. С. 602. DOI:10.3390/educsci15050602.

31. Qayyum A. Student help-seeking attitudes and behaviors in a digital era. International Journal of Educational Technology in Higher Education. Вип. 15, № 1. С. 17. DOI:10.1186/s41239-018-0100-7.

32. Qwen/Qwen3-4B-Thinking-2507. Hugging Face. 06.08.2025. URL: https://huggingface.co/Qwen/Qwen3-4B-Thinking-2507 (дата звернення: 06.12.2025).

33. Reasoning model. Wikipedia. 22.11.2025. URL: https://w.wiki/GV62 (дата звернення: 04.12.2025).

34. Reverse proxy quick-start. Caddy Documentation. URL: https://caddyserver.com/ (дата звернення: 06.12.2025).

35. The official home of the Python Programming Language. Python.org. 05.12.2025. URL: https://www.python.org/ (дата звернення: 06.12.2025).

36. Transactive memory. Wikipedia. 17.06.2025. URL: https://w.wiki/GVq9 (дата звернення: 05.12.2025).

37. Vaquero L. M., Cebrian M. The Weakness of Weak Ties in the Classroom. 2012. DOI:10.48550/ARXIV.1201.1589.

38. What is the Model Context Protocol (MCP)? Model Context Protocol. URL: https://modelcontextprotocol.io/docs/getting-started/intro (дата звернення: 04.12.2025).

39. Abdillah L. A. Managing information and knowledge sharing cultures in higher educations institutions. arXiv, 2014. DOI:10.48550/ARXIV.1402.4748.

40. Golub B., Jackson M. O. How Homophily Affects Diffusion and Learning in Networks. arXiv, 2009. DOI:10.48550/arXiv.0811.4013.

41. Han H., Trimi S. Cloud Computing-based Higher Education Platforms during the COVID-19 Pandemic. arXiv, 2022. DOI:10.48550/ARXIV.2203.03714.

42. Lewis P., Perez E., Piktus A. та ін. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv, 2021. DOI:10.48550/arXiv.2005.11401.

43. Li H., Xu T., Zhang C. та ін. Bringing Generative AI to Adaptive Learning in Education. arXiv, 2024. DOI:10.48550/arXiv.2402.14601.

44. Ra E., Kim S. J., Seo E.-Y. та ін. Designing LMS and Instructional Strategies for Integrating Generative-Conversational AI. arXiv, 2025. DOI:10.48550/arXiv.2509.00709.

45. Striuk A. M., Semerikov S. O. Blended learning models. arXiv, 2018. DOI:10.48550/arXiv.1808.04893.

46. Yaacoub A., Tarnpradab S., Khumprom P. та ін. Enhancing AI-Driven Education: Integrating Cognitive Frameworks, Linguistic Feedback Analysis, and Ethical Considerations for Improved Content Generation. arXiv, 2025. DOI:10.48550/arXiv.2505.00339.

ДОДАТКИ

Додаток А

Діаграма комунікації мікросервісів двох підсистем через API

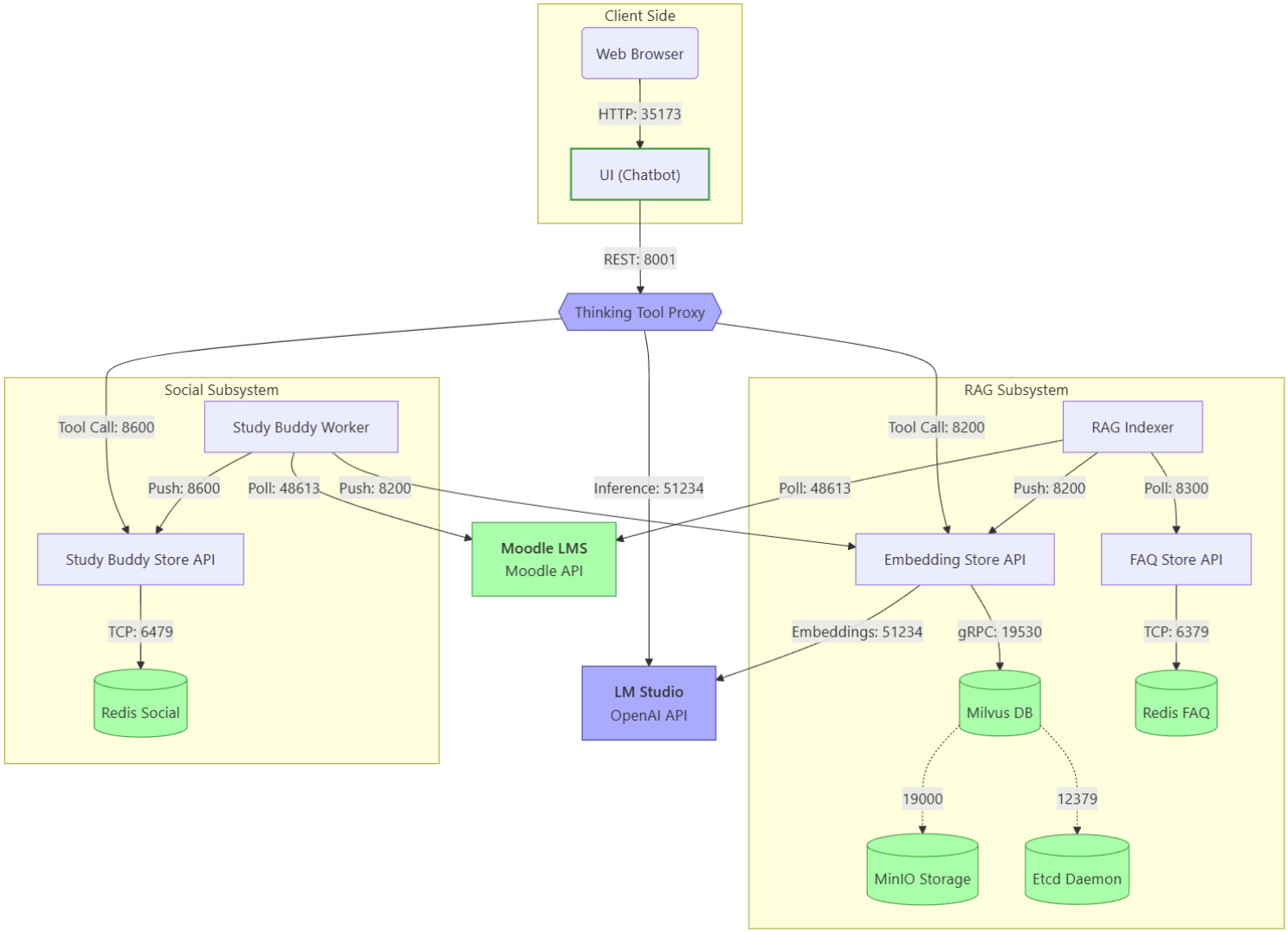


Рис. А.1. Комунікація мікросервісів модулів системи через методи/технології/REST API та порти цієї взаємодії

Додаток Б

Алгоритм вбудовування тексту із додаванням метаданих на мові Python

```
import json
import tiktoken
import random

MODEL_MAX_TOKENS = 2048
MODEL_EMBEDDING_DIMENSION = 768
ENCODING = tiktoken.get_encoding("cl100k_base")

def num_tokens_from_string(string: str) -> int:
    return len(ENCODING.encode(string))

def prepare_document_chunks(metadata: str, content: str) ->
list[str]:
    metadata_tokens = num_tokens_from_string(metadata)
    if metadata_tokens > MODEL_MAX_TOKENS:
        raise ValueError("Metadata is too large to be processed.")

    max_content_tokens = MODEL_MAX_TOKENS - metadata_tokens

    if num_tokens_from_string(content) <= max_content_tokens:
        return [content] if content else []

    words = content.split()
    current_chunk_words = []
    chunks = []

    for word in words:
        if num_tokens_from_string(" ".join(current_chunk_words +
[word])) > max_content_tokens:
            if current_chunk_words:
                chunks.append(" ".join(current_chunk_words))
            current_chunk_words = [word]
        else:
            current_chunk_words.append(word)

    if current_chunk_words:
        chunks.append(" ".join(current_chunk_words))

    if not chunks and content:
        encoded_content = ENCODING.encode(content)
        truncated_tokens = encoded_content[:max_content_tokens]
        return [ENCODING.decode(truncated_tokens)]

    return chunks
```

```
def create_metadata_padded_embeddings(documents: list[dict],
model: SimulatedEmbeddingModel) -> list[dict]:

    vector_database = []

    for doc in documents:
        doc_id = doc["doc_id"]
        metadata = doc["metadata"]
        content = doc["content"]

        metadata_str = json.dumps(metadata, ensure_ascii=False)

        chunks = prepare_document_chunks(metadata_str, content)

        if not chunks and content == "":
            chunks.append("")

        for i, chunk in enumerate(chunks):
            text_to_embed = metadata_str + "\n" + chunk
            embedding = model.embed_query(text_to_embed)

            vector_database.append({
                "pk": f"{doc_id}_chunk_{i}",
                "doc_id": doc_id,
                "metadata": metadata_str,
                "chunk": chunk,
                "embedding": embedding,
            })

    return vector_database
```

Додаток В

Алгоритм пошуку ментора із проіндексованих даних на мові Python

```
def find_study_buddy(topic: str):
    best_match = search_activities(topic)

    metadata = best_match.get('metadata', {})
    activity_id = metadata.get('activity_id')
    activity_name = metadata.get('name', 'Unknown Activity')

    experts = get_experts_for_activity(activity_id)

    result_text = f"Found mentors for '{activity_name}':\n"
    for expert in experts:
        status_icon = "🟢" if expert.status == 'online' else
("🟡" if expert.status == 'active' else "🔴")
        result_text += f"{status_icon} {expert.name} (Score:
{int(expert.score * 100)}%) - Contact: {expert.contact}\n"

    return result_text.strip()


def search_activities(topic: str) -> Optional[Dict]:
    search_params = {"metric_type": "L2", "params": {"nprobe":
10}}
    activity_filter = 'metadata["type"] == "activity"'
    query_embedding = embedding_model.embed_query(topic)

    collection = Collection(ACTIVITY_COLLECTION_NAME)
    collection.load()
    results = collection.search(
        data=[query_embedding],
        anns_field="embedding",
        param=search_params,
        limit=1,
        expr=activity_filter,
        output_fields=['doc_id', 'metadata']
    )

    hit = results[0][0]
    return {
        "id": hit.id,
        "distance": hit.distance,
        "doc_id": hit.entity.get('doc_id'),
        "metadata": hit.entity.get('metadata')
    }
```

```
def get_experts_for_activity(activity_id: int, limit: int = 3) ->
List[ExpertProfile]:
    activity_key = f"activity:{activity_id}:experts"

    redis_client = redis.Redis(host=REDIS_HOST, port=REDIS_PORT,
decode_responses=True)

    expert_ids_with_scores = redis_client.zrevrange(activity_key,
0, limit - 1, withscores=True)

    pipe = redis_client.pipeline()
    for user_id, _ in expert_ids_with_scores:
        profile_key = f"user:{user_id}:profile"
        pipe.hgetall(profile_key)
    profiles_raw = pipe.execute()

    experts = []
    for i, (user_id, score) in enumerate(expert_ids_with_scores):
        profile = profiles_raw[i]
        if profile:
            experts.append(ExpertProfile(
                user_id=int(user_id),
                name=profile.get("fullname", "Unknown"),
                score=score,
                status=profile.get("status", "offline"),
                contact=profile.get("contact", "N/A")
            ))
    return experts
```