Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

факультет прикладних інформаційних технологій та електроінженерії
(повна назва факультету)
кафедра автоматизації технологічних процесів і виробництв
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: «Розроблення чат-бота для інформаційної підтримки студентів та автоматизації комунікаційних процесів кафедри»

Виконав(ла): студент(ка) IV курсу, групи КА-41
спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»
(шифр і назва спеціальності)

(підпис) (прізвище та ініціали)
Полатайко В.Т
(підпис) (прізвище та ініціали)

Керівник (підпис) Коноваленко І.В. (прізвище та ініціали)
Нормоконтроль (підпис) Козбур І.Р. (прізвище та ініціали)
Завідувач кафедри (підпис) Савків В.Б. (прізвище та ініціали)
Рецензент (підпис) Королюк Р.І. (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет прикладних інформаційних технологій та електроінженерії
(повна назва факультету)

Кафедра автоматизації технологічних процесів і виробництв
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Савків В.Б.
(підпис) (прізвище та ініціали)
« ___» _____2026р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 151 «Автоматизація та комп'ютерно-інтегровані технології»
(шифр і назва спеціальності)

студенту Полатайко Василь Тарасович
(прізвище, ім'я, по батькові)

1. Тема роботи «Розроблення чат-бота для інформаційної підтримки студентів та автоматизації комунікаційних процесів кафедри»

Керівник роботи Коноваленко І.В., к.т.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджена наказом ректора від « 14 » травня 2026 року № 4/9-231
2. Термін подання студентом завершеної роботи 25 червня 2026 року
3. Вихідні дані до роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
1) Аналітична частина;
2) Проектна частина;
3) Спеціальна частина;
4) Безпека життєдіяльності, основи охорони праці.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Презентація кваліфікаційної роботи аркушів формату А4

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Спеціальна частина	*доц. Коноваленко І.В.,*		
Безпека життєдіяльності,	*доц. Сенчишин В.С.*		
основи хорони праці			

7. Дата видачі завдання *«____» _____20_____ р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи	Примітка
1.	*Аналітична частина*	*20.05.2026*	
2.	*Проєктна частина*	*25.05. 2026*	
3.	*Спеціальна частина*	*05.05. 2026*	
4.	*Безпека життєдіяльності, основи охорони праці*	*10.06. 2026*	
5.	*Оформлення графічної частини та пояснювальної*	*20.06. 2026*	
	записки		
6.	*Захист кваліфікаційної роботи*	*25.06. 2026*	

Студент _____ *Полатайко В.Т.*
(підпис) (прізвище та ініціали)

Керівник кваліфікаційної роботи _____ *Коноваленко І.В.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Створення локального інтелектуального асистента для кафедр університету // Кваліфікаційна робота освітнього рівня «Бакалавр» // Полатайко Василь Тарасович // Тернопільський національний технічний університет імені Івана Пулюя, факультет прикладних інформаційних технологій та електроінженерії, кафедра автоматизації технологічних процесів і виробництв, група КА-41 // Тернопіль, 2026 // С. 85, рис. - 19, табл. - 3, кресл. - 0, додат. - 0, бібліогр. – 31.

Ключові слова: чат-бот, RAG, Telegram, LLM, Python, Qdrant, автоматизація, бази даних.

Кваліфікаційна робота присвячена створенню та розробці незалежного колінеарного цифрового розумного бота для месенджера Telegram на основі архітектури RAG. Робота спрямована на автоматизацію інформаційного супроводу студентів, оптимізацію рутинних процесів комунікації в межах кафедри, водночас забезпечуючи конфіденційність і безпеку конфіденційних даних закладів вищої освіти (ЗВО). У першому розділі кваліфікаційної роботи розглянуто наявні рішення у сфері чат-ботів та ШІ, обґрунтовано актуальність використання локальної RAG-архітектури та сформульовано завдання проєктування. Другий розділ описує процес розробки мікросервісної архітектури, включно із застосуванням PostgreSQL і Qdrant для баз даних та інтеграцією з моделями через Ollama. В третьому розділі кваліфікаційної роботи розглядаються алгоритми роботи модуля інжектування документів (включно з OCR), розробка програмного забезпечення та результати тестування створеної системи. В четвертому розділі кваліфікаційної роботи розглядаються питання безпеки життєдіяльності та охорони праці при роботі з обчислювальною технікою і серверним обладнанням.

ANNOTATION

Creation of a Local Intelligent Assistant for University Departments // Qualification work of the "Bachelor" educational level // Vasyl Tarasovych Polataiko // Ternopil Ivan Puluj National Technical University, Faculty of Applied Information Technologies and Electrical Engineering, Department of Automation of Technological Processes and Manufacturing, Group KA-41 // Ternopil, 2026 // Pages: 85, figures: 19, tables: 3, drawings: 0, appendices: 0, references: 31.

Keywords: chatbot, RAG, Telegram, LLM, Python, Qdrant, automation, databases.

The qualification work is dedicated to the creation and development of an independent collinear digital smart bot for the Telegram messenger based on the RAG architecture. The work is aimed at automating informational support for students and optimizing routine communication processes within the department, while ensuring the privacy and security of confidential data of higher education institutions (HEIs).

In the first chapter of the qualification work, existing solutions in the field of chatbots and AI are reviewed, the relevance of using a local RAG architecture is substantiated, and the design tasks are formulated.

The second chapter describes the process of developing a microservice architecture, including the application of PostgreSQL and Qdrant for databases, and integration with models via Ollama.

The third chapter of the qualification work examines the operational algorithms of the document ingestion module (including OCR), software development, and the testing results of the created system.

The fourth chapter of the qualification work addresses the issues of life safety and occupational health and safety when working with computing machinery and server equipment.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД — база даних.

ДБЖ — джерело безперебійного живлення.

ЗВО — заклад вищої освіти.

СУБД — система управління базами даних.

ШІ — штучний інтелект.

API (Application Programming Interface) — програмний інтерфейс застосунку для взаємодії між різними компонентами та сервісами.

LLM (Large Language Model) — велика мовна модель, заснована на алгоритмах глибокого навчання, здатна розпізнавати, узагальнювати та генерувати текст.

NLP (Natural Language Processing) — обробка природної мови.

OCR (Optical Character Recognition) — технологія автоматичного розпізнавання оптичних символів з растрових зображень.

RAG (Retrieval-Augmented Generation) — генерація тексту з підсиленням пошуком; архітектура, що поєднує можливості мовних моделей з доступом до зовнішніх баз знань.

TEI (Text Embeddings Inference) — мікросервіс для високопродуктивного розгортання моделей векторних текстових вбудовувань.

UX (User Experience) — користувацький досвід, що характеризує зручність і комфорт взаємодії користувача з інтерфейсом системи.

ЗМІСТ

ВСТУП

Із цифровізацією сучасного світу з’являється значний етап людського розвитку, на який, імовірно, найбільший пріоритет впливає у межах цієї трансформації - вища освіта. Нова генерація студентів виросла в епоху, де панували смартфони та постійний доступ до інтернету; вони очікують, що інформація, особливо у форматі, який підходить саме їм, буде під рукою. Старі шкільні методи комунікації на кшталт паперових дошок оголошень, довгих дзвінків у приймальню деканату чи пошуку в поросших павутиною файлах на 40-річних сторінках вебсайтів університетів, на жаль, залишилися в смітнику, далеко позаду наших нинішніх часів. Сьогодні месенджери є головним центром студентського життя, адже Telegram і надалі залишається абсолютним лідером, це можна відстежити за кількістю завантажень на рисунку 1. [28]

Рисунок 1 – Найпопулярніші по завантаженням в Україні месенджери

Водночас обсяг роботи перед співробітниками адміністративних позицій постійно зростає. Викладачі, методисти та секретарі щодня змушені витрачати дорогоцінні години на відповіді на звичні запитання: «Де розклад?», «Які вимоги до формату курсової роботи?», «Як мені зв’язатися з науковим керівником?». Цей безперервний потік листів одного й того самого типу перетворює добре підготовлених фахівців на справжні пошукові системи, які працюють в режимі «пошуку» та інформування, що знижує ефективність усього структурного підрозділу. Вирішення такої проблеми полягає в упровадженні нових технологій ШІ, здатних опрацьовувати більшість рутинних комунікацій.

Ця тема є актуальною через новий прорив в історії великих мовних моделей (LLM). Проте, придбання публічних комерційних сервісів на кшталт ChatGPT або Claude для потреб конкретного підрозділу є, з огляду на кілька ключових причин, неприйнятним. Насправді загальні моделі не мають знань про конкретний внутрішній контекст установи: вони не знають актуальних розкладів, викладачів у певний час або номер аудиторії, який буде призначено. Хибні факти, або «галюцинації», продукуються публічними нейромережами, а це означає, що вони можуть генерувати дуже переконливу, але повністю хибну інформацію.[16]

Одним із інших порушень, яких припускається передавання внутрішньої документації на зовнішні сервери, що порушує принципи конфіденційності та безпеки даних. З огляду на це, єдиним справжнім і коректним шляхом уперед є використання технології retrieval-augmented generation (RAG) у поєднанні з повністю локальними (on-premise) мовними моделями. Архітектура RAG допомагає системі спочатку знайти деякі коректні фрагменти документів у власній векторній базі даних, і лише після цього наказує нейромережі згенерувати чітку відповідь, спираючись

виключно на контекст, отриманий у результаті пошуку(retrieval).[29] Таким чином, створення кастомного чатбота, який працює на власному обладнанні, з використанням інструментів на кшталт Ollama та Qdrant, - це не просто інженерна вправа, а потужний крок уперед для сучасної вищої освіти. Це дозволяє вам гарантувати відсутність будь-яких галюцинацій і повністю захистити конфіденційні дані університету.

Метою дипломної роботи є створення та практичне впровадження локалізованого інтелектуального чат-бота на архітектурі RAG для отримання інформаційної підтримки студентам на місці, а також для автоматизації рутинних комунікаційних процесів кафедри.

Відповідно до визначеної мети були сформульовані такі критично важливі завдання для виконання та їх реалізовано:

1. Дослідити еволюцію та сучасний стан технік обробки природної мови, висвітлити недоліки традиційних скриптованих ботів і публічних LLM, та обґрунтувати, чому використання локальної архітектури RAG є доцільним у академічному середовищі.
2. Запланувати реляційну БД PostgreSQL для збереження історії діалогів, опрацювати робочу схему взаємодії мікросервісів і написати модуль індексації документів для векторної бази даних Qdrant, що працює з локальною моделлю через Ollama.
3. Написати операційний алгоритм для програмного забезпечення, провести системне тестування його релевантних і коректних відповідей, обмежити потоки відправлення тексту в Telegram і створити повноцінну інструкцію щодо того, як розгорнути все це та поповнити базу знань.
4. Провести аналіз вимог з безпеки життєдіяльності та охорони праці.

Об’єкт розробки: процес автоматизації інформаційного супроводу студентів та комунікаційної взаємодії в межах структурного підрозділу закладу вищої освіти.

Предмет розробки: методи обробки природної мови, архітектурні підходи (RAG) та програмно-технічні засоби побудови автономних інтелектуальних асистентів на базі локальних великих мовних моделей.

Методи виконання наданих завдань. Для досягнення мети цієї бакалаврської роботи було використано підхід поєднання теоретичних і емпіричних методів. Загальна топологія системи та взаємодія її компонентів побудовані на методах системного аналізу, що дозволило створити відмовостійку мікросервісну архітектуру. Під час розроблення модуля семантичного пошуку використали нейронні текстові вкладення або векторне подання тексту та алгоритми семантичного зіставлення, що дає змогу швидко знаходити потрібний контекст у базі знань. Логіка асистента та сервіси обробки документів були реалізовані як програмні рішення на основі парадигм об’єктно-орієнтованого та асинхронного програмування із застосуванням Python.

Наукова значущість і практична цінність отриманих результатів полягають у створенні інтегрованого, повністю автономного інформаційного середовища для кафедри. Систематизовано та застосовано підхід, за якого інтелектуальні асистенти освітніх установ не залежать від ліцензій або хмарних ресурсів глобальних IT-корпорацій. Новизна цього рішення полягає у формуванні безперервного конвеєра: від завантаження адміністратором звичайного текстового документа в систему, миттєвої векторизації у вигляді оптимізованого Text Embeddings Inference (TEI) і, нарешті, генерації результату через Ollama. Це дає змогу усунути проблему втрати інформації у загальній моделі, оскільки вона стане динамічним асистентом із актуальною інформацією завжди.

Це вказує на абсолютну готовність розробленого програмного продукту до інтеграції в реальні умови. Якщо коротко, упровадження бота докорінно змінює щоденну роботу відділу: секретаріат звільняється від постійних пересилань навчальних матеріалів по колу, а студенти отримують цілодобовий доступ 24/7 через свого улюбленого месенджера до компетентного співрозмовника. Завдяки розгортанню мікросервісної архітектури через Docker Compose це також забезпечує високу відмовостійкість і легке масштабування. Інтелектуальний модуль обробки та індексації документів усуває бар’єри для нетехнічних працівників, уможливлюючи оновлення бази знань шляхом завантаження нових файлів. Його легко адаптувати для будь-якого іншого факультету, відділу кадрів або навіть приймальної комісії університету, тож цей проєкт є універсальним інструментарієм для модернізації та цифровізації освітнього процесу.

Структура кваліфікаційної роботи повністю відображає етапи досягнення мети. Робота містить вступ, чотири взаємопов'язані розділи (аналітичну, проєктну, спеціальну частини та розділ охорони праці і безпеки життєдіяльності), загальні висновки та вичерпний перелік використаних джерел.

1 АНАЛІТИЧНА ЧАСТИНА

1.1 Аналіз стану питання за літературними та іншими джерелами

Перші приклади автоматизованих систем зв’язку та обробки тексту включали амбітні експерименти зі створення програми, яка могла б ефективно вести повну людську розмову. Революційним здобутком у цій галузі був чатбот ELIZA, розроблений дослідником Джозефом Вайценбаумом у Массачусетському технологічному інституті (MIT) ще в 1966 році. Ця комп’ютерна програма працювала на доволі простих

алгоритмах, які здебільшого були покликані імітувати роботу психотерапевта під час сеансу. ELIZA була запрограмована так, щоб постійно ставити відкриті запитання, що успішно спонукало користувача й далі згадувати різні речі та ділитися своїми думками. Але найсуттєвішою й, можливо, найвразливішою (у негативному сенсі) частиною було те, що такі повністю ригідні сценарні (тобто правил-орієнтовані) підходи абсолютно не мали жодного реального розуміння змісту запитань, які ставилися. Насправді програма не знала нічого про те, про що саме йдеться, або про контекст розмови взагалі. На найпримітивнішому рівні вона спиралася на принцип зіставлення шаблонів: витягувала з тексту користувача певні ключові слова й, дотримуючись заздалегідь прописаних граматичних правил, переказувала їх або ж відповідала, спираючись на заздалегідь підготовлені шаблонні відповіді. Незважаючи на таку просту структуру й відсутність будь-якої реальної інтелектуальної обробки, ELIZA була надзвичайно успішною в тому, щоб створювати вигляд інтелекту. Велика кількість користувачів настільки занурювалася в розмову, що вони щиро й неправдиво вірили, ніби спілкуються з істотою, наділеною чутливістю та співпереживанням.[1, 7, 8, 18]

Однак це стало «сигналом кінця» для будь-чого більш складного з погляду інформаційних систем. Адже ригідні сценарії не можуть справлятися із синонімами, орфографічними помилками та нестандартним формулюванням, тож підтримка призводить до нескінченних лабіринтів із заздалегідь написаних відповідей - що є відверто невідповідним для сьогоднішньої сфери освіти, яка постійно еволюціонує.

За останні роки методи машинного навчання розвинулися так стрімко, що відбулася справжня тектонічна зміна та кардинальний зсув у обробці природної мови (NLP). Насправді поява цих технологій проклала шлях до створення гнучких систем, які можуть працювати з величезними обсягами

даних і вчитися в міру потреби, щоб виявляти складні закономірності в цих даних та прогнозувати результат без безлічі годин людського втручання або постійного ручного програмування. Еволюція перейшла на якісно новий рівень пізніше - із появою алгоритмів глибокого навчання та використанням багатошарових штучних нейронних мереж. Такі архітектури, які принаймні теоретично частково наслідують роботу людського мозку (тобто глибоке навчання), відкрили цілком нові можливості для аналізу й опрацювання текстів. Нейронні мережі також дали змогу виявляти семантичні зв’язки, які раніше були невидимими: скриптовані боти не мали розуміння контексту в межах одного речення, а емоція сутності, що визначається словом, яке використовується, істотно відрізнялася від випадку до випадку. Сучасна обробка природної мови піднесла взаємодію людини й комп’ютера до небувалого рівня витонченості та ефективності. Ці багаторівневі системи NLP - кожен рівень «живить» наступний завдяки надзвичайно ефективним алгоритмам машинного навчання, які використовують для тренування низки кандидатних моделей нині формують майже вулканічну основу, на якій упродовж часу можуть наростати сучасні рішення, як-от автоматизовані програми перекладу, аналітика, і надрозумні віртуальні асистенти на кшталт Apple Siri або Google Gemini. Такі методи дають змогу машинам не лише витягувати слова з рядка тексту, а й фактично формувати глибокі семантичні уявлення, точно розпізнавати намір користувача та генерувати належні відповіді. [22]

Але найбільше, проривне покращення в розпізнаванні та генерації тексту з’явилося з появою великих мовних моделей (LLM) - відомих завдяки серії GPT від OpenAI. Усі ці системи використовують величезний (триліони токенів), неструктурований і не класифікований набір даних, зібраний з усього інтернету, який споживає колосальні обсяги людських знань. Сучасні LLM створені на епохальній архітектурі під назвою transformer, яка спирається на складний механізм уваги. Унаслідок цього моделі аналізують

усі слова в тексті одночасно та визначають, які з них важливі, а також як вони пов’язані між собою, що дає їм змогу демонструвати колосальні здібності, зокрема міцне розуміння контексту, міркування і навіть креативність. Переваги таких моделей у будь-якому разі перевищують можливості звичайних rule-based чатботів і це не підлягає сумніву. Це видно в здатності самостійно писати дуже довгі, складні тексти будь-якого типу, перекладати десятки мов із гарною якістю та коректністю, інтерпретувати, а також генерувати й аналізувати програмний код, а також відповідати на високодеталізовані, багаторівневі запитання з чудовим ступенем точності. Крім того, сучасні LLM прийшли переважно для того, щоб покращити логічне мислення та пам’ять. Вони можуть запам’ятовувати контекст попередніх розмов протягом тривалого діалогу, швидко аналізувати файли, які ви завантажуєте до них, а також, у режимі реального часу, переглядати весь інтернет у пошуках релевантних деталей. Ці вдосконалення дають їм можливість еволюціонувати від простих систем автоматичної відповіді до справжніх багатофункціональних розумних цифрових асистентів.

У контексті оцінювання опублікованих аналогів академічних чатботів і високоспеціалізованих систем підтримки студентів для закладів вищої освіти (ЗВО) вивчення наявної літератури та відкритих джерел дало змогу виявити певною мірою розрізнене заповнення цієї двохмісної лабораторії. Сектор «Штучний інтелект в освіті» у відповідних опрацьованих документах здебільшого згадувався як загальний інструмент, що сприяє масштабуванню рутинних адміністративних і навчальних процедур. Він допомагає з автоматичною перевіркою тестових завдань та попередньою аналітикою результативності. Крім того, нині алгоритми ШІ використовуються для персоналізованих навчальних програм, конкретної оцінки студентів на певному рівні знань, а також для динамічного, у реальному часі коригування навчального контенту відповідно до темпу кожного студента. Посилання на

застосування ШІ у простих віртуальних помічниках і його інтеграцію в глобальні методи дистанційного навчання трапляються широко.

Але після глибокого занурення ви розумієте, що немає готових “під ключ”, офлайн-рішень для локальних розумних чатботів, які були б ближчими до розв’язання щоденних проблем комунікації в окремих закладах вищої освіти, якщо їх автономно випустити на рівні фрагментарних рішень, що відповідають адміністративному підрозділу університету. Більшість вітчизняних закладів вищої освіти вже використовують прості скриптові бот-версії в поширених месенджерах. Навігаційний тип програм-помічників обмежується посиланнями на розклади занять або контактні дані. Він не може аналізувати відкритий природномовний текст, а це означає, що він не здатний цілком реально зробити запит на конкретний тип контенту, і, більш фундаментально, він не може автономно аналізувати зміст методичних рекомендацій або освітніх програм. Це підкреслює необхідність створення нового локального інтелектуального помічника. Найкраще рішення, побудувати систему на архітектурі RAG, створеній на автономних LLM, яка забезпечує інтелектуальні генеративні можливості нейромереж, а також дає змогу безпосередньо отримувати доступ до внутрішньої документації підрозділів, повністю усуваючи ризики витоку даних і генерації дезінформації.

1.2 Актуальність виконання роботи

Взаємодія між студентами закладів вищої освіти та адміністративним і викладацьким персоналом завжди була однією з найважливіших - і водночас по суті вразливих - сфер існування будь-якого закладу вищої освіти. Сьогодні, в умовах дедалі динамічнішого й цифрового освітнього процесу, традиційні канали передавання інформації загалом є малоефективними. Також ми чуємо, що обіцяний зворотний зв’язок просто не відбувається вчасно, а між очікуваннями студентів і здатністю кафедр їх забезпечити

виникають значні розриви. Студенти отримують інформацію негайно, через смартфони, і в будь-який час доби через легкодоступні месенджери. Натомість на рівні кафедри спілкування зводиться до хаотичного масиву щільних документів, нескінченного розміщення текстів на сайтах або, за потреби, індивідуального звернення до методистів чи викладачів.

Якщо ми розглянемо щоденну ділову діяльність працівників адміністрації, можна побачити потік рутинних запитів. Студенти постійно уточнюють про поточний розклад, коли слід здавати лабораторні звіти, деталі форматування курсових робіт, хто є їхніми науковими керівниками та як пройти виробничу практику. Традиційні інструменти автоматизації (наприклад, прості статичні чатботи, які називають сценарними навігаторами) зовсім не допомагають у вирішенні цієї проблеми. Все, що вони можуть дати користувачеві, це базовий набір кнопок, а далі приховують перенаправлення до того самого величезного pdf або таблиць. Наприклад, якщо студент хоче, щоб його вказали на конкретну вимогу з комплекту методичних інструкцій на вісімдесят сторінок, за умови, що нічого не можна виключити, статичний бот лише надішле студенту посилання на весь підручник, перекладаючи відповідальність за пошук назад по колу і знімаючи її з себе. Отже, розчаровані студенти ігнорують ці онлайн-платформи та знову обирають очні взаємодії з різними викладачами або співробітниками, надсилаючи приватні повідомлення. Це створює колосальне емоційне та навантажувальне напруження для персоналу: висококваліфіковані спеціалісти щодня витрачають цінні години, дублюючи одну й ту саму інформацію. У результаті підрозділи відчувають стрес, а дослідницькі та викладацькі навантаження різко погіршуються.

Під час стрімкої еволюції генеративного ШІ, здавалося б, це був очевидний і швидкий крок: просто інтегрувати провідні комерційні публічні сервіси великих мовних моделей, такі як ті, що пропонує OpenAI з ChatGPT,

або Anthropic з Claude. Проте, якщо розглянути використання цих масштабних хмарних рішень ближче, стає дуже зрозуміло, що університетський підрозділ ні в якому разі не повинен застосовувати один із них з низки причин, які є цілком базовими. Однією з найважливіших перешкод є безпека та конфіденційність даних. Процес навчання різноманітного студентського контингенту нерозривно пов'язаний із керуванням величезними обсягами внутрішньої інформації, необхідної для підтримання високого рівня організаційної ефективності: списки студентів і співробітників у спеціальних, проєктних або курсових соціальних навчальних групах, у яких співпрацюють як у межах, так і між установами; директиви щодо розроблення стратегії та забезпечення якості; результати академічної успішності; авторські навчальні матеріали, які залишаються неопублікованими протягом періодів, доки їх доопрацьовують тощо. Надсилання такого роду інформації через API на сервери чужорідних IT-корпорацій, знову ж таки, є грубим порушенням інформаційної безпеки та стандартів захисту даних. [25]

Іншим важливим елементом є неконтрольований характер економічної моделі, яка визначає комерційні продукти ШІ. Такі сервіси дуже точно встановлюють ціну на обчислювальні ресурси, стягуючи плату за кожен згенерований і оброблений токен. Важливо, що цей асистент опрацьовуватиме сотні тисяч студентів і десятки тисяч запитів на день, особливо під час сесій заліків і іспитів, дуже скоро зроблять бюджет «у межах будь-яких розумних обмежень» недосяжним. А це означає, що будь-якій державній установі початкової та середньої освіти це фінансово буде неможливо. [25]

Третя проблема, яка є не меншою складністю для комерційних моделей, відсутність вузького, по суті локального контексту, а також їхня боротьба з надто легкою готовністю до «галюцинацій» - переконливо

згенерованої інформації, яка не має зв’язку з реальністю. [9] Але глобальну модель можна запитати про точний розклад іспитів для конкретної когорти або про номер кабінету керівника лабораторії, і оскільки ці актуальні, але наразі відсутні дані не включені до ваг навчання цієї загальної нейромережі, вона, імовірно, просто придумає правдоподібну, але неправильну відповідь. Поширення хибної інформації в академічному середовищі є цілком неприпустимим. Саме тому весь фокус цієї дисертації було перенаправлено на побудову моделей з відкритим кодом, які працюватимуть повністю на локальному обладнанні серверів установи, і жоден байт інформації не буде спрямовано назовні у зовнішню мережу.

RAG, або генерація з підсиленням пошуком (Retrieval-Augmented Generation), була розроблена, щоб «приземлити» локальну мовну модель у повсякденному житті кафедри, водночас не дозволяючи їй виходити з-під контролю й вигадувати фантазійні відповіді під час генерації тексту, керованої пошуком. Це повністю зламало звичну логіку ШІ, наприклад. Замість того щоб намагатися витягувати з нейромережі надто абстрактну пам’ять, яку було «вишитовхнуто» з величезного навчального набору, архітектура RAG пропонує адаптивний обхідний шлях від запиту користувача до того, що найімовірніше є актуальною локальною правдою.

Це працює так само, як у м’язах, по ідеально прокладеному конвеєру. Внутрішні документи, які містять усі дані, потрібні для виконання вашої роботи - робочі скрипти, розклади, навчальні посібники - спочатку імпортуються в систему та автоматично поділяються на керовані логічні розділи або «фрагменти». Далі виконується векторизація (вбудовування): ці матеріали перетворюються на багатовимірні числові масиви, які надійно зберігаються в спеціалізованій векторній базі даних. Так само, коли студент запитує в месенджері, ця невизначеність одразу теж перетворюється на вектор. Потім ми застосовуємо алгоритм для швидкого семантичного

пошуку: порівнюючи, наскільки далеко один від одного розташовані два вектори, ми відповідно беремо лише ті абзаци з офіційної документації підрозділу, які найкраще відповідають вашому запиту.

Подальша генерація відбувається на основі правдивих попередньо витягнутих фактів. Це дає змогу надійно прибрати проблему галюцинацій. Нейронна мережа вже не прикидається всезнаючою вона працює як просто розумний процесор тексту, що видає ввічливу відповідь у прозі на основі певної, перевіреної інформації. [6] Під час обговорення економічної та соціальної раціональності автоматизації інформаційного обслуговування студентів із використанням подібних автономних інтелектуальних систем слід знати, що їхній позитивний вплив на повсякденне життя закладу вищої освіти є суттєвим. Третя складова проєкту - це вкрай необхідне радикальне покращення цифрового досвіду студентів - і саме тут розміщується їхній соціальний компонент. Забезпечте здобувачам безперервний доступ у будь-який час, 7 днів на тиждень і без вихідних до якомога більшої кількості конкретних відповідей, які вони бажають, у форматі, яким вони вміють користуватися у смартфоні. Тепер їм більше не потрібно зазнавати психологічних мук, коли доводиться будити викладачів у нестандартні години вночі, або затамовувати подих на години лише для того, щоб переглянути архіви. Це значно зменшує відчуття всепроникної тривоги й, відповідно, робить розмови плавними та привабливими, представляючи університет як сучасну орієнтовану на молодь організацію.

Економічно немає іншого довгострокового інвестування, яке було б таким сприятливим, як запуск локального RAG-асистента. Для підготовки інфраструктури потрібен час і обчислювальні ресурси, але після того, як усе налаштовано, щоденна поточна робота системи коштує практично нічого. Університет звільняється від цінових коливань комерційних API або платних підписок. Локальний розумний бот перетворюється на складне

завдання невтомного цифрового секретаря, який невпинно фільтрує й обробляє переважну більшість повторюваних рутинних запитів. Методисти, лаборанти, секретарі та викладачі отримують повну свободу від виснажливої текстової кореспонденції. Десятки годин, збережені щодня, можна спрямувати на те, що справді має значення, і на творчі завдання: на інновації в навчальних матеріалах, на глибші дослідницькі активності, на підготовку грантових заявок або на індивідуальну роботу зі студентами щодо складності інженерних розробок. Як наслідок, кафедра може опрацьовувати більше зразків без збільшення штату і, відповідно, досягати значно вищого рівня загальної операційної продуктивності. Створення локальної розумної системи є своєчасною відповіддю на виклики інформаційної епохи та об’єктивно обґрунтованою дією на шляху до формування інфраструктури, що забезпечує високу освітню ефективність.

1.3 Методи розв'язання поставленої задачі

Проблема автоматизації комунікаційних процесів відділу не буде вирішена без вибору та науково обґрунтованого підбору сучасного технологічного стеку, який забезпечує точність і швидкість обробки запитів на основі абсолютної незалежності від зовнішніх комерційних платформ. Обрана базова архітектура для побудови інтелектуального асистента ґрунтується на підході, відомому як Retrieval-Augmented Generation (RAG). Нині це відкриває можливість поєднати гнучкі генеративні можливості великих мовних моделей зі структурованими локальними знаннями в університеті. Архітектура RAG поділяє конвеєр обробки інформації на два основні макрокомпоненти: компонент пошуку інформації (retriever) і компонент генерації відповіді (generator). Цей двоетапний процес перетворює традиційну взаємодію з нейронними мережами на більш роз’єднаний інтерфейс, де більше не потрібно, щоб вони автономно зберігали терабайти фактів.[29]

Ретвір (пошукова система) - як інтелектуальний оркестратор бази знань. Не потрібно вигадувати нові слова; радше слід уважніше вивчити наявну внутрішню документацію підрозділу (керівні принципи, розклади та навчальні програми), зіставивши її лише з тими фрагментами тексту, які містять відповідь, підтверджену напряму запитанням, сформульованим студентом. Цей пошуковий блок більше не ґрунтується лише на збігові кількох термінів, а працює з глибокими семантичними значеннями. У разі надсилання користувачем повідомлення в месенджері ретрівер математично порівнює цей запит із тисячами раніше проіндексованих фрагментів документів і надає текстові блоки для найрелевантніших збігів.

Друга частина - генератор - реалізована як велика локальна мовна модель. Компонент пошуку постачає їй відфільтровані фрагменти та оригінальний запит користувача. Його завдання - перетворити нудний набір витягнутих команд або процедур на плавну, граматично коректну й природну відповідь мовою. Така взаємодія означає, що генератору не потрібно пам’ятати нічого конкретного зі своїх внутрішніх ваг, що суттєво зменшує потенціал помилок. Розділення роботи з пошуку та генерації усуває найсуттєвіший недолік сучасних систем ШІ: уразливість до дезінформації - що є принципово важливим моментом у контексті цієї роботи, використання RAG зменшує кількість галюцинацій на 70%, рисунок 2 це демонструє.

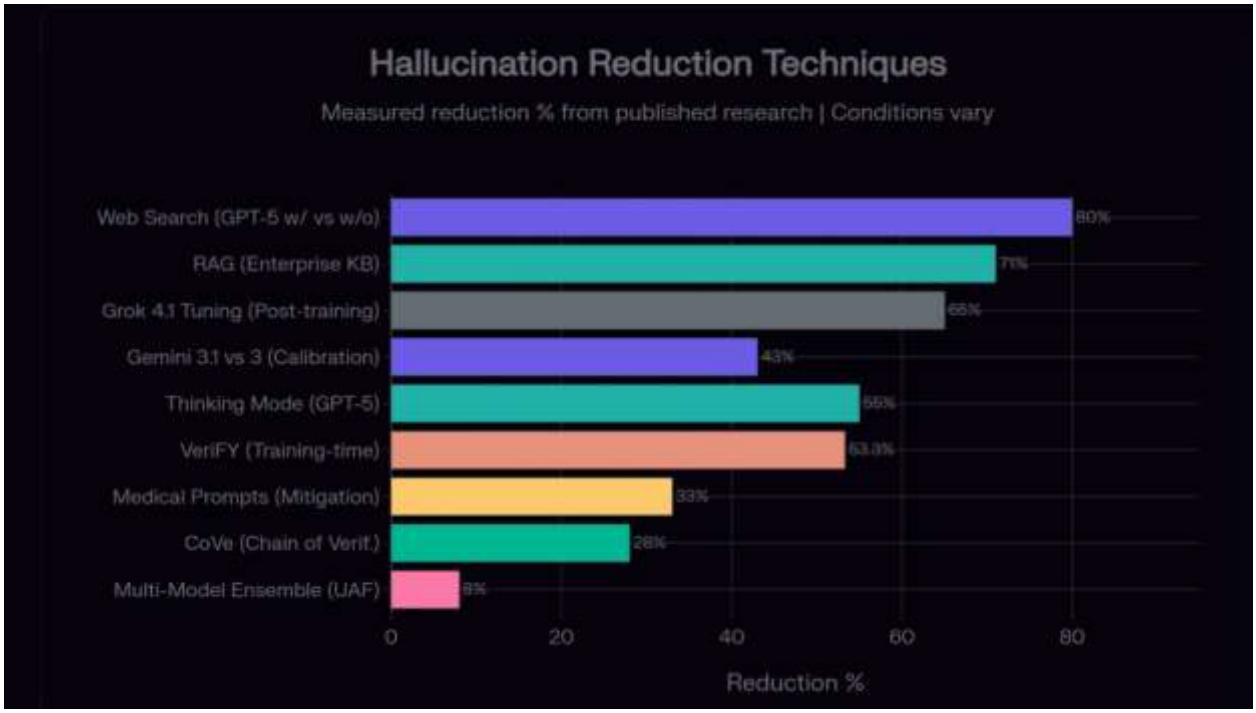

Рисунок 2 – Техніки зменшення галюцинацій [15]

Взаємодія між компонентами пошуку та генерації ґрунтується на процесі векторизації тексту, який називають генерацією вбудовувань (embeddings). Вбудовування - це математичне перетворення будь-якого фрагмента тексту, попередній етап, щоб перетворити довільну частину тексту на багатовимірний вектор, що складається з числових значень однакової довжини, де кожний вимір відповідає певній абстрактній семантичній ознаці, наглядно це представлено на ілюстрації на рисунку 3. Для такої трансформації використовують спеціальні моделі нейромережевого кодування. У чому особливість такого кодування: вектори відображають зміст вашого тексту. Слова або речення з дуже подібними логічними значеннями будуть знаходитися близько одне до одного в багатовимірному просторі навіть за умови, що вони не мають спільних символів і коренів. Наприклад, фрази на кшталт «коли здавати дипломну» та

«терміни подачі випускних робіт» отримають вектори з дуже подібними просторовими координатами.

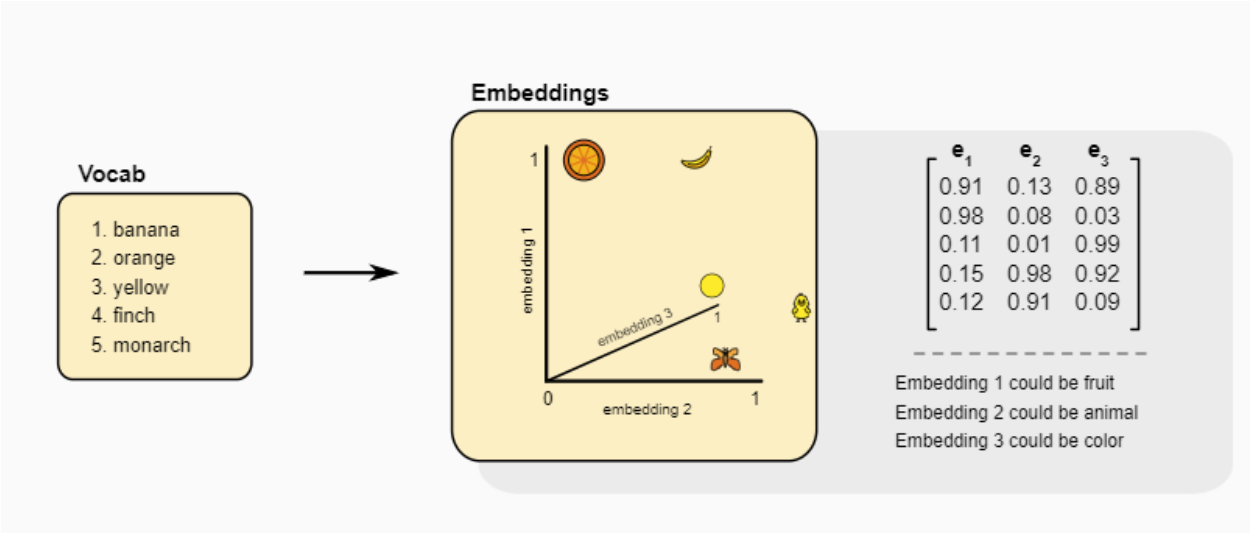

Рисунок 3 – Ілюстрація векторизації [24]

Коли створюється масив текстів, усі вони мають бути розбиті на чанки це як розділення їх на фіксовані за розміром фрагменти з часткою перекриття (у тому самому порядку, що й оригінальний контекст). Певне перекриття потрібне, щоб не втратити семантичний контекст на межах цих розділених файлів. Векторна база даних реалізує семантичний пошук, обчислюючи певну «сенсорну» відстань або деталь подібності між вектором запиту користувача та, відповідно, векторами збережених фрагментів документів. Поширеними метриками для обчислення цієї подібності є косинусна подібність (cosine similarity) і скалярний добуток (dot product). Після цього ви індексуєте, векторизуєте та завантажуєте всі документи відділу на спеціалізоване сховище. Коли студент ставить питання, воно миттєво створює вектор (Vector A) для цього запиту, а далі ви використовуєте векторну базу даних для виконання операції NNS(пошук найбличих сусідів). Унаслідок цього за частку секунди викликається корпус

текстів, які найближчі один до одного за змістом, на якій ґрунтується генератор подальшої роботи.

Оскільки обчислювальні ресурси відділення університету завжди є більш обмеженими, ніж будь-який хмарний дата-центр, розгортання великих мовних моделей на локальній інфраструктурі університету має виконуватися з урахуванням існуючих програмних платформ. Було розглянути основні методи, які дозволяють виконувати локальний вивід - це необроблені низькорівневі бібліотеки на кшталт llama.cpp, розгортання складних промислових фреймворків на кшталт vLLM та використання розвинених оркестраторів. Екосистему Ollama було обрано для виконання роботи дипломної роботи. Ollama - це новий тип локального середовища для запуску AI-моделей, який працює дуже легким і сучасним способом. Вона також підтримує сучасні методи квантування (стиснення моделі без суттєвої втрати якості, наприклад, до формату 4bit GGUF) завдяки використанню оптимального механізму виводу. Це дає змогу запускати потужні моделі з сімейств Qwen або Llama, Gemma навіть на звичайних GPU або високопродуктивних CPU. Виділення пам’яті виконується автоматично Ollama, підтримуються асинхронні запити з простим інтерфейсом API для зручного доступу та інтеграції інших мікросервісів, створених системою.

База даних Qdrant була обрана як рішення для зберігання векторних даних для кваліфікаційної роботи. Зрештою, Qdrant, порівняно з іншими варіантами на кшталт ChromaDB, Milvus або FAISS, можна відзначити багатьма перевагами, з якими навряд чи хтось буде сперечатися, і які були критично важливими для цього проєкту. Написана мовою Rust, ця база даних створена для швидкодії та мінімального використання оперативної пам’яті під час виконання складних геометричних обчислень у кількох вимірах. Qdrant надає всі функції, готові до промислового застосування: підтримку розширеного фільтрування (payload filtering), що дає змогу

виконувати строго контрольовані пошуки серед векторів, які належать до певного документа або курсу, а також підтримку створення графів ваг із HNSW (Hierarchical Navigable Small World) для ультрашвидкого пошуку прямо з коробки - це допомагає масштабувати базу знань до дуже великих обсягів; у поєднанні зі стабільним і добре задокументованим Python SDK. Це дозволяє побудувати стійкий мікросервіс, який легко контейнеризувати (dockerize) та надійно запускати в автономному режимі.

Іншим архітектурним викликом для систем RAG є те, як швидко й точно генеруються вектори для сотень мільйонів сторінок текстових корпусів під час початкового інжесту документів або під час частих динамічних оновлень бази знань. Стара схема, коли векторизація відбувається за допомогою тієї самої моделі Ollama, що й для генерації тексту, є ключовим вузьким місцем. Використання обчислювальних ресурсів спричиняє черги та затримки під час відповіді користувачам. Щоб вирішити цю проблему, у цій роботі використано спеціальний інструмент: Text Embeddings Inference (TEI) від Hugging Face.

TEI - це надоптимізований мікросервіс, створений для розгортання моделей векторних текстових вбудовувань та повторного ранжування. Його місце в конвеєрі обробки документів є ключовим. Такі елементарні високорівневі операції можна реалізувати на Rust із подальшими оптимізаціями на кшталт Flash-Attention і Paged Attention, щоб отримувати ще швидше генерування вбудовувань, ніж це дозволяють стандартні реалізації на Python. Мікросервіс забезпечує динамічне батчування (дає змогу об’єднувати в один пакет десятки текстових фрагментів із різних документів, щоб максимально задіяти всі CPU-ядра або тензорні ядра в разі використання GPU). Нам вдалося досягти повної ізоляції даних завдяки розгортанню TEI: ollama взаємодіє лише для генерації відповідей у telegram, а TEI миттєво отримує вхідні масиви щодо методологічної літератури.

Оптимальний технологічний стек для інженерного завдання, яке включає RAG, Ollama, Qdrant і TEI, - це збалансоване, надійне та самодостатнє рішення.

1.4 Висновки та постановка задач на кваліфікаційну роботу бакалавра

Спираючись на результати комплексного огляду науково-технічної літератури, аналіз документації, пов’язаної з сучасними фреймворками, а також застосовуючи методологію об’єктивного огляду для рішень з автоматизації ринку у сфері роботи університетських систем комунікації, було виявлено кілька ключових закономірностей. Серед основних висновків насамперед встановлено, що велика частка академічних чатботів, які нині працюють у закладах вищої освіти, побудовані на жорстких сценарних алгоритмах (rule-based) або на спрощеному пошуку за наперед визначеними ключовими словами. Вони звичні як добре відомі електронні довідники, приховані за графічним інтерфейсом дуже популярного месенджера. І вони не мають глибокого розуміння запиту. Уявімо, що студент запитує: «коли дедлайн для лабораторних?», а база даних бота має лише такі строгі формулювання, як терміни подачі для лабораторних робіт, тож він навіть не зрозуміє, що саме хоче ця людина. Це спонукає користувачів знову й знову шукати в нескінченних деревах навігаційних меню та натискати десятки кнопок, фактично погоджуючись із тією самою передумовою, що потрібна швидкодоступна, інтуїтивна інформаційна підтримка.

Також в результаті аналізу хмарних провайдерів(Gemini, GPT, Claude) великих мовних моделей(LLM), чи то правильніше сказати агентів на основі LLM, якими виступають ці рішення. Було визначено що хоч їхня загальна й доволі висока здатність продукувати значущі тексти, які є одночасно зв’язними та повністю спроможні підтримувати природний діалог, та закритий характер цих комерційних платформ щодо потреб

університетського підрозділу було визнано неприйнятним. Друга причина полягає в тому, що ці зовнішні сервіси мають мати можливість отримувати доступ до внутрішньої документації за замовчуванням, яка відкрита лише для підрозділу. Надсилання навчальної та конфіденційної інформації, на сторонні сервери створює колосальний ризик та є прямим порушенням принципів безпеки. Крім того, генеративні нейронні мережі схильні «збиватись» із курсу: без точних відповідей у їхніх навчальних вагах вони вигадуватимуть неправдиві речі, змінюватимуть дати іспитів або навіть вигадуватимуть імена викладачів. Оскільки фактична точність буквально є важливою складовою освітнього процесу, така алгоритмічна поведінка є цілком неприпустимою.

Огляд ринку також показав, що жодне з розглянутих рішень не включає всі необхідні інструменти: автоматизоване розпізнавання складних документів, швидкий векторний семантичний пошук і локальне формування відповіді без залежності від будь-яких комерційних API в одному інтегрованому системному середовищі. Нам потрібно швидке й миттєве створення повністю автономної інформаційної екосистеми, яка працюватиме на локальних обчислювальних ресурсах, доступних освітній установі.

Внутрішньо, спираючись на виявлені слабкі місця та недоліки, під час виконання дипломної роботи було визначено конкретну мету та складено перелік окремих технологічних завдань, які мають використовуватися покроково для створення повноцінного локального смарт-асистента.

Найбільш базовою технічною відповідальністю є створення мікросервісної архітектури програмної системи. Усе логічне рішення буде спроєктовано так, щоб воно повністю розміщувалося в коректному контейнери (контейнеризації), який також у подальшому (з часом) зможе використовувати контейнери, наприклад, Docker. Це також вирішує проблему конфліктів програмних залежностей, забезпечує високу

стабільність роботи, дає змогу легко розгортати систему на серверному обладнанні кафедри та дозволяє незалежно масштабувати окремі мікросервіси під час пікових навантажень у періоди сесій.

Впровадження програмного забезпечення для спеціалізованого сервісу Knowledge Ingestion (модуля інжеста знань + індексації) є ще одним надзвичайно важливим завданням. Цей компонент відповідає за автоматичний парсинг і очищення від графічних артефактів робочих файлів усіх підрозділів. Особливу увагу слід приділити впровадженню OCR-технологій, оскільки значна частина певних документів університетів, таких як підписані відомості, оцифровані записи або старі паперові розклади, зберігається у вигляді растрових зображень. Таким чином, після успішного отримання тексту цей модуль має розбити його на розумні фрагменти (чанки заданого розміру) та одразу проіндексувати в k-вимірному просторі для миттєвого часу відповіді на запити в Qdrant.

Третє завдання, впровадження зручного та зрозумілого клієнтського інтерфейсу у формі бота в telegram. У контексті такого типу завдань важливо організувати стабільне отримання та надсилання повідомлень через telegram api, а також забезпечити працюючі системи перевірки та реєстрації користувачів. У зв’язку з цим ми плануємо розгортання реляційної бази даних PostgreSQL для зберігання облікових записів студентів разом із їхніми налаштуваннями та повною історією діалогу. Це означає, що система пам’ятатиме щось на кшталт: користувач запитав щодо методичних рекомендацій, а потім у наступному повідомленні просить надати більш ґрунтове пояснення певного їхнього розділу - тож навіть якщо це розділено в часі іншими повідомленнями, це все одно буде достатньо логічно, щоб зрозуміти контекст.

Четверте та найтехнічно складніше завдання стосується завершення пайплайна RAG (Retrieval-Augmented Generation). Якщо ми створимо

оптимальну конфігурацію Ollama, тоді зможемо розгорнути локальну велику мовну модель, яка підтримуватиме генерування та бездоганну точність тексту. Паралельно триватиме розробка для подальшого додавання функціональності, а також для зменшення значного навантаження, яке виникає під час створення векторних представлень тексту (embeddings) у вашій генеративній моделі. Це включає підключення мікросервісу Hugging Face Text Embeddings Inference (TEI). Ефективне поєднане використання цих інструментів із відкритим кодом має забезпечити, що нейронна мережа генеруватиме відповіді лише абзацами з документів відділу в базі даних Qdrant і не враховуватиме жодної зовнішньої неперевіреної інформації.

Мета роботи чітко визначена та підкріплена архітектурними рішеннями, які безпосередньо адресують кожний із виявлених недоліків щодо наявних підходів. У основі технології RAG лежить розроблене нами передове й сучасне рішення, яке усуває ключову проблему галюцинацій LLM: тепер віртуальний асистент більше не вигадує фактів, оскільки його відповіді строго залежать від заздалегідь завантажених офіційних документів, таких як інструкції та розклади, які подає адміністрація та які використовуються ним. Розгортання всього локально - від текстового генератора до векторної бази даних - на серверах самого університету забезпечує повну конфіденційність і безпеку. Найважливіше в цій мережевій інфраструктурі те, що жоден байт не може вийти за її межі, що необхідно в наявному сценарії.

Зрештою автономний конвеєр інтелектуального завантаження знань автоматизує індексацію нових текстових матеріалів, завдяки чому наша база знань завжди залишається актуальною. Щойно новий розклад занять приймається відділом або вимоги до формату курсових робіт принципово переформатовуються, відповідальний працівник просто замінює застарілий файл та інформація в системі знову актуальна. Вона розпізнає текст:

перетворює його на векторну форму та додає цю інформацію в пам’ять без потреби в явній команді. Це звільняє розробників і співробітників відділу від ресурсомістких вимог складного навчання моделей (файн-тюнінгу) або від ризикових змін у програмному коді щоразу, коли оновлюються навчальні програми. Це дає змогу створити надзвичайно гнучку, ефективну та масштабовану інтелектуальну систему, повністю підготовлену до багатьох років інтенсивної роботи в реальних умовах навчання.

2 ПРОЄКТНА ЧАСТИНА

2.1 Характеристика системи та її призначення

Від теоретичного обґрунтування до практичного впровадження нам потрібне чітке визначення архітектурних рішень і оптимальний вибір інструментів. Загальна архітектура запропонованої системи побудована за принципом високорівневої мікросервісної архітектури: кожен окремий програмний компонент, реалізований окремо, виконує строго визначену логічно ізольовану функцію. Використання мікросервісної архітектури, на відміну від монолітної, дозволяє широко масштабуватися та підтримувати працездатність та оптимізувати кожен сервіс окремо. Розроблений інтелектуальний комплекс ґрунтується на координації та постійній взаємодії шести ключових вузлів, які паралельно створюють єдину, високо ефективну автономну систему для обробки інформації без потреби використання зовнішніх ресурсів.

Основним мікросервісом буде Telegram-агент, і саме він взаємодіятиме з кінцевим користувачем. Він працюватиме як цифровий диспетчер: швидко прийматиме вхідні повідомлення, виконуватиме базові перевірки безпеки та маршрутизацію текстових запитів, а також оркеструватиме складний процес генерації відповіді через алгоритми конвеєрів RAG (retrieval-augmented generation - генерація з урахуванням пошуку). Наступним програмним компонентом, рівною мірою важливим, як і попередній, є сервіс Knowledge Ingestion Service, спеціальний тип багатопотокового модуля для здобуття й обробки знань. Часто університетська документація являє собою суміш форматів: одні сучасні та зручні для читання (наприклад, текстові файли), інші - старі скановані замовлення або навчальні програми у вигляді растрових зображень. Саме тому цей модуль має вбудовану функцію OCR, як випливає з назви - автоматичне розпізнавання оптичних символів. Сервіс глибоко обробляє,

синтаксично нормалізує та інтелектуально фрагментує витягнений текст із зображень або PDF-файлів після успішного вилучення. Розділяючи великі масиви тексту спочатку на менші логічні блоки з певним коефіцієнтом перекриття, стає можливим надійно та операційно узгоджено категоризувати кожну академічну вимогу чи норму з одного документа перед завантаженням у цифрову базу даних підрозділу для швидкого пошуку.

Інтелектуальний обчислювальний «мозок» системи повністю створений на перспективних локальних рішеннях для нейромереж. Ollama - це спеціалізована платформа для генерації змістовних відповідей природною мовою. Таким чином, це спрощує встановлення та експлуатацію великих генеративних мовних моделей, забезпечуючи серверну інфраструктуру, що належить інституціям. У свою чергу, це усуває необхідність раз і назавжди подавати заявки на платні комерційні API хмарних сервісів, одночасно гарантуючи збереження цілісності інформаційної безпеки та постійний захист персональних даних студентів у будь-якому масштабі без додаткових витрат у разі збільшення кількості запитів.

Але перш ніж довіряти нейромережам реальні факти, кванти спочатку мають перекласти текст на математичну мову. Для розв’язання цього завдання ми впровадили сучасний мікросервіс для Text Embeddings Inference від Hugging Face. Двигун бере звичайні фрагменти тексту й миттєво перетворює їх на надзвичайно складні багатовимірні числові вектори (embeddings). [31]

Як тільки ці вектори згенеровано, ми відправляємо їх у довгострокове сховище в Qdrant - векторній базі даних, підготовленій саме під наш конкретний сценарій використання. Натомість ця власницька база даних швидко знаходить тисячі великих масивів чисел і запускає на них алгоритми для негайного семантичного пошуку - те, з чим типове рішення для зберігання даних просто не впоралося б.

Коли надходить запит від студента, Qdrant виконує обчислення косинусної відстані за мілісекунди між вектором для кожного нового запиту та векторами, що представляють кожен із збережених документів, знаходячи контекстні абзаци, які допомагатимуть надалі включити їх до мови-моделі після.

Ми впровадили PostgreSQL, перевірену часом класичну реляційну базу даних для надійного зберігання структурованої інформації. Її проєктування допомагає надавати лише індивідуальному користувачу відповідний тип профілю, механізми авторизації та зберігати повну історію контексту. Надзвичайно важливо зберігати історію повідомлень для підтримання узгодженості під час тривалої сесії спілкування, коли наступне уточнювальне питання студента безпосередньо випливає з відповіді, наданої інтелектуальним ботом. У сукупності розроблений віртуальний помічник ефективно й вражаюче вирішує задачу інформаційного супроводу, створюючи щось на кшталт «єдиного вікна» для забезпечення вільного доступу до всього внутрішнього банку знань кафедри. Завдяки впровадженій технології RAG чатбот жодним чином не обмежується примітивним пошуком за ключовими словами. Він усвідомлює справжнє значення незвичного формулювання запитання та дає повну письмову відповідь, граматично правильну, цілісну, однак виключно на основі офіційних розкладів, методичних праць або адміністративних актів. Це усуває виснажливу потребу переглядати безліч незв’язаних між собою сторінок у мережі або незграбно й ментально вишукувати файли, щоб знайти потрібне після годин виснажливих пошуків усіх посилань.

Автоматизація цих комунікаційних процесів не лише допомагає зменшити щоденне когнітивне та операційне навантаження на персонал адміністрації деканату, лаборантів і викладацький склад, щоб вони могли

ефективно зосередитися на своїй безпосередній викладацькій або інноваційно орієнтованій ролі.

Причина використання Telegram як єдиного запланованого каналу зв’язку з цією інформаційною системою є дуже вагомим міркуванням з погляду користувацького досвіду (UX). Сьогодні українські студенти широко використовують цей месенджер як основний інструмент для неформального спілкування, швидкого обміну навчальними матеріалами та стрічкою новин. Оскільки це середовище буде знайомим студентам, розгортання розумного бота безпосередньо в цьому звичному середовищі знижує психологічний бар’єр (тобто студентам не потрібно знову нічого завантажувати й установлювати, реєструватися або вчитися користуватися будь-яким новим додатковим програмним забезпеченням тощо). Ви отримуєте гарантовану миттєву доставку повідомлень навіть на найслабших або найбільш нестабільних мобільних інтернет-з’єднаннях, а також пропонується надзвичайно високий рівень криптографічного захисту каналів зв’язку. По-друге, це забезпечує потужні та інтуїтивно зрозумілі можливості для розширеного форматування генерованого тексту за допомогою Markdown. Це дає змогу, щоб критичні кінцеві терміни акуратно виділялися жирним, розклад занять був добре структурований через блоки коду з моноширинним шрифтом, а також щоб були зручні невеликі легкі посилання, які перенаправляють вас за межі бота. Ще одним аспектом є надійний, швидкий та надзвичайно багатий функціями API месенджера, який допомагає розробникам створювати як звичайні елементи інтерфейсу, так і інтерактивні компоненти, зокрема спеціальні вбудовані клавіатури, що дають змогу забезпечити більш інтуїтивну та зручну взаємодію з асистентом.

Для гарантованої та безперервної роботи, у крайньому рівні відмовостійкості або за простотою встановлення на реальному серверному

обладнанні вся мікросервісна архітектура розробленої системи тісно оркестрована інструментами Docker Compose. Використання цього сучасного програмно-орієнтованого інженерного підходу гарантує, що кожен окремий мікросервіс працює повністю ізольовано, із застосуванням легковагого засобу, який називається контейнером, із заданим користувачем набором бібліотек, залежностей і змінних середовища. Це дозволить нам повністю уникнути будь-яких технічних конфліктів програмного забезпечення між різними модулями системи (наприклад, критичної двійкової несумісності з різними версіями мови Python або несумісних системних пакетів). [20]

Саме так реалізується корпоративна мережева безпека даних на основі закритої програмно визначеної внутрішньої віртуальної мережі. Усі потоки між різними компонентами системи всередині цієї ізольованої мережі можуть відбуватися вільно, швидко та анонімно, тоді як слабкі місця у вигляді баз даних, наприклад Qdrant, PostgreSQL, а також локальний механізм мовної моделі сам по собі - у нашому випадку Ollama, приховані від будь-якої точки доступу ззовні, починаючи від некерованого входу і аж до публічного периметра інтернет-з’єднання. Зовні публічно доступними залишаються лише ті порти, які потрібні для роботи «вебхуків» застосунку Telegram, щоб бути в безпеці, взаємодію мікросервісів показано на рисунку 4 . У підсумку ви можете повністю контейнеризувати всі майбутні компоненти та мати адміністратора системи, який гнучко (і буквально за лічені хвилини) масштабуватиме вузли обчислювальних екземплярів під час піків трафіку. Додатково, весь готовий програмний набір можна без зайвих зусиль і простоїв перенести на інші платформи обладнання, у хмарні або на локальні сервери.

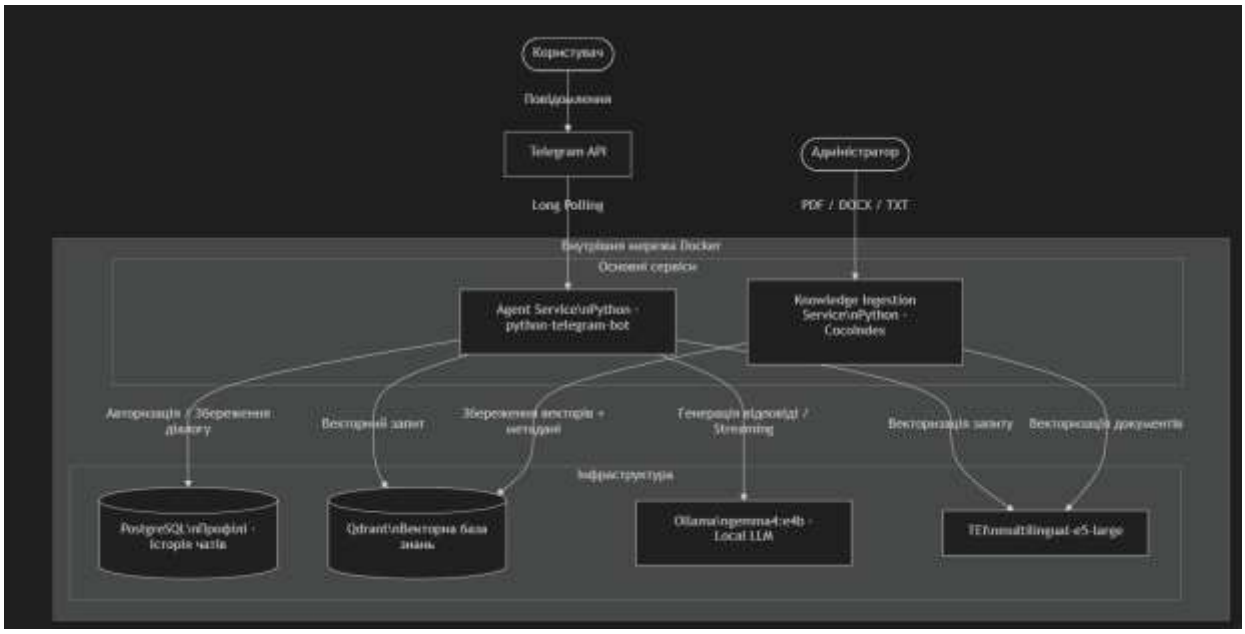

Рисунок 4 - Діаграма взаємодії мікросервісів системи

2.2 Проєктування бази даних (PostgreSQL) для збереження даних користувачів та історії чатів

Створення виробничої якості швидкої та масштабованої системи зберігання даних є одним із найбазовіших речей під час розробки будь-якого сучасного вебзастосунку, але ще більш актуальним - під час створення інтелектуального віртуального асистента. Надійно обґрунтованим вибором для зберігання даних про студентів, керування правами доступу та, головне, забезпечення справді добре організованої довгострокової контекстної пам’яті бота стала система керування реляційними базами даних (DBMS) PostgreSQL. Перевірена тисячами проєктів із мільйонами високонавантажених рішень, ця технологія відзначається безпрецедентною стабільністю, суворим дотриманням ACID (atomicity, consistency, isolation, durability) та чудовими можливостями для оптимізації багатоступеневих запитів.

Оптимальна архітектура проєктування бази даних використовує принципи нормалізації, щоб усунути надлишкову інформацію та уникнути аномалій під час оновлення або видалення записів. Логічна структура зосереджена на профілях користувачів і повідомленнях, які вони публікують, тобто власника двох інших таблиць - користувачів і повідомлень.

Репозиторій users, деталі описані в таблиці 1, є основною базою даних цього застосунку, яка використовується для безпечної реєстрації та ідентифікації студентів. Первинний ключ (Primary Key) для цієї таблиці було обрано - тому що єдиний вхід у систему здійснюється через телеграм-месенджер: унікальний числовий ідентифікатор користувача, що генерується серверами Telegram (Telegram ID). Це не лише гарантує, що жодний запис ніколи не зміниться, а й значно спрощує весь процес авторизації щоразу, коли робиться запит до бота. Окрім первинного ключа, таблиця містить надзвичайно важливий набір метаданих профілю - Окрім первинного ключа, таблиця зберігає метадані профілю: ім'я (first_name), прізвище (last_name), унікальний номер залікової книжки (record_book_no) та ознаку адміністративних прав (is_admin). Також фіксується дата й час першої реєстрації в системі (registered_at). щоб керівництво могло спостерігати, як студенти впроваджують новий цифровий інструмент.

Але ми не хотіли простого авто-відповідача, а інтерпретативного інтелектуального співрозмовника, який міг би вести багатокрокові складні діалоги. Саме тому ми створили репозиторій chat_history, поля якого описані в таблиці 2. І саме він створює так звану контекстну пам’ять. Кожен запис у цій таблиці відповідає повідомленню безпосередньо в межах розмови, відповідає одному повному діалоговому ходу та містить: внутрішній ідентифікатор (id), текст запиту студента (user_query), текст відповіді,

згенерованої системою (agent_response), а також мітку часу збереження запису (timestamp).

Зв’язок між профілями окремих студентів і повною історією їхньої переписки з ботом здійснюється за допомогою класичного зовнішнього ключа (user_id). Цей ключ дуже тісно пов’язаний із первинним ключем таблиці users і зберігається всередині таблиці chat_history. Завдяки ретельно продуманій архітектурі основний бекенд-сервіс робить локальний швидкий SQL-запит до бази даних перед тим, як надіслати новий запит на студента, щоб модель генерації можна було забезпечити необхідною інформацією. Він отримує останні N повідомлень (зазвичай це останні 12 висловлювань) разом із відповідними user_id і дописує цю історію розмови до локального системного промпта LLM. Саме таке точне розуміння логічного перебігу цієї розмови на той момент дає змогу нейронній мережі надати конкретну й уточнювальну відповідь, а не щоразу відтворювати кожен діалог із нуля.

Таблиця 1 - Структура таблиці users

Поле	Тип	Обмеження	Опис
telegram_id	BIGINT	PRIMARY KEY, INDEX	Унікальний ідентифікатор користувача Telegram
first_name	VARCHAR(100)	NOT NULL	Ім'я студента
last_name	VARCHAR(100)	NOT NULL	Прізвище студента
record_book_no	VARCHAR(50)	UNIQUE, nullable	Номер залікової книжки (формат: КА-22-203)
is_admin	BOOLEAN	NOT NULL, default=False	Ознака адміністратора системи

registered_at	TIMESTAMPTZ	NOT NULL, default=now()	Час реєстрації в системі

Таблиця 2 - Структура таблиці chat_history

Поле	Тип	Обмеження	Опис
id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Унікальний ідентифікатор запису
user_id	BIGINT	FK → users.telegram_id, INDEX	Посилання на користувача
user_query	TEXT	NOT NULL	Текст запиту студента
agent_response	TEXT	NOT NULL	Відповідь, згенерована системою
timestamp	TIMESTAMPTZ	NOT NULL, default=now()	Час збереження повідомлення

На рівні рушія СУБД, який є найнижчим рівнем, забезпечується повна наскрізна прозорість і надійність збережених даних. У критичних полях (NOT NULL), таких як Telegram ID або текст повідомлення, не допускалися значення NULL, а також задавалися умови агрегації (UNIQUE). Для зовнішнього ключа таблиці chat_history налаштовано каскадне видалення

(ON DELETE CASCADE). І це було можливим завдяки такому правилу: якщо автоматичне видалення курсу видаляє профіль студента, PostgreSQL неминуче видалить усі повідомлення, пов’язані з користувачем, без інших ручних команд у бекенді.

Одним із дуже елементарних етапів проєктування бази даних було забезпечення легкого доступу до записів з історії. На практиці під час реального використання за дуже короткий час у таблиці повідомлень накопичаться сотні тисяч рядків. Якщо бот має послідовно шукати щось подібне одне за одним, то йдеться про терпіння, яке здається нелюдським. Для цього ми створили додаткові індекси B-дерева (збалансовані дерева) спеціально для полів user_id і created_at. Індексація дозволяє рушію PostgreSQL миттєво відфільтрувати величезні обсяги даних, відсортувати за часом за лічені секунди та знайти відповідні записи не лінійним пошуком, а логарифмічним. Це гарантує, що RAG-асистент отримуватиме весь релевантний історичний контекст майже одразу, а також відповідатиме в реальному часі, даючи студенту відчуття, ніби він/вона веде відкриту, природну й розслаблену розмову з кафедрою.

2.3 Розробка функціональної схеми системи та обґрунтування вибору технічних засобів

Тепер, щоб розроблений інтелектуальний асистент працював стабільно та швидко без втрати з’єднання, ми створили функціональну діаграму, яка описує кожен етап потоку інформації від пристрою студента до генеративного рушія та у зворотному порядку (таблиця 3). Ми обробляємо запит користувача не просто як передачу тексту; це складний, багатостадійний, повністю автоматизований конвеєр, де кожен мікросервіс виконує свою роботу. Звичайне текстове повідомлення дає нам можливість відстежити кожен його крок і повну глибину інтеграції компонентів.

Таблиця 3 - повний цикл інформаційного потоку

Етап	Опис
1–3	Захисні перевірки: Rate Limit → Авторизація → Довжина
4	Safety Check через LLM
5	Завантаження останніх 12 повідомлень з PostgreSQL
6	Відправка заглушки студенту
7	RAG-конвеєр: TEI → вектор → Qdrant → топ-5 фрагментів
8	Формування промпту + потокова генерація
9	Фінальна відповідь + збереження в chat_history

Усе починається тоді, коли студент відкриває Telegram і пише, наприклад, «розклад іспитів». Як тільки цей запит надходить в інфраструктуру, його буде передано одному з основних вузлів зв’язку в нашій системі - сервісу Agent (бекенд-оркестратору). Після того як Agent отримує пакет даних, він одразу підключається до реляційної бази даних PostgreSQL і починає запити. Далі виконується швидка перевірка: чи зареєстрований цей користувач у системі, які в нього права доступу та який контекст із попередніх діалогів із ним збережено. PostgreSQL зберігає історію, і ми витягнемо її так, щоб це було зрозуміло нашому боту, аби не втратити логічний зв’язок, якщо, наприклад, це запитання уточнює. Потім, окрім звичних фрагментів, важливо перетворити текст вашого запиту на математику. Для цього Agent надсилає вихідний текст у мікросервіс TEI (Text Embeddings Inference). Сервіс TEI передає текст нейронній моделі та повертає багатовимірний числовий вектор (embedding), який досконало описує смислову суть запиту студента.

З цим вектором у руках бекенд-сервіс далі видає Qdrant векторній базі даних чітку інструкцію: знайти в базі знань відділу ті документи, чиї вектори є найближчими за просторовою близькістю до вектору запиту. Qdrant виконує пошук найближчих сусідів і витягує K найкращих релевантних фрагментів текстової інформації (наприклад, абзац-ембедінг, що містить розклад). Використовуючи цей цінний шматок фактичних даних як тло, Агент переходить до фінального доповнення. Створюється так званий «super-prompt» - величезний фрагмент тексту, який поєднує суворий системний промпт, історію діалогу, витягнуту з PostgreSQL, контекст дослівно (як знайдено через Qdrant), а також сам запит користувача. Пакет усіх цих даних надсилається локальній великій мовній моделі - Ollama. Потім, спираючись на цей контекст, NN починає генерувати відповідь. За умови стрімінгу Агент може не чекати, поки модель напише весь текст. Натомість він захоплює згенеровані слова (токени) по одному й одразу транслює їх назад у вікно чату Telegram у міру створення. Тож студент бачить, як текст плавно й поступово з’являється на екрані, підсилюючи повний ефект розмови з живою людиною.

Не менш важливим, ніж написання коду, є забезпечення належного обґрунтування наших програмних інструментів для кожного вузла цього конвеєра. Під час створення векторного сховища ми розглянули кілька популярних рішень: Chroma, Weaviate та Qdrant. Стало очевидно, що базу даних Qdrant буде обрано як остаточний варіант. Перш за все завдяки її вражаючій продуктивності та тому, що система реалізована повністю на компільованій мові програмування Rust. Qdrant - це не просто швидкий прототип на основі Chroma, який має блокування під час одночасних запитів (Chroma створено на Python), а повноцінне, високо надійне, готове до продакшну промислове рішення. Воно використовує мінімальний серверний RAM для сотень тисяч векторів. Крім того, Qdrant оснащено критично важливими функціями: складним фільтруванням за метаданими (або

фільтруванням за payload). Це означає, що алгоритм усе ще може шукати подібні вектори, але також може обмежуватися строго за класифікаціями; наприклад, можна модифікувати систему індексування документів доповнивши додатковими полями категорій і здійснювати пошук з фільтрацією. За потреби воно дає змогу вилучати нерелевантну інформацію ще до обчислення геометричної відстані, що суттєво підвищує точність роботи бота.

Для суворих вимог до безпеки та конфіденційності, які унеможливлюють доступ до популярних хмарних API (зокрема OpenAI ChatGPT або Anthropic Claude), було обрано локальну модель із екосистеми Ollama (варіант gemma4:e4b). Інформаційна екосистема університету наповнена чутливими метаданими: списками того, хто які курси відвідував; особистими телефонними номерами викладачів; неопублікованими дослідженнями; внутрішнім листуванням вищого керівництва щодо успішності студентів. Надсилання таких масивів інформації на сервери сторонніх комерційних корпорацій є грубим порушенням політик інформаційної безпеки. Але при локальній моделі ви маєте повну ізоляцію: буквально нічого з боку даних університету або запитів студентів не виходить за межі стін серверів власного підрозділу. Ще одна принципова перевага локального розгортання - повна фінансова автономія. Хмарні сервіси стягують плату залежно від кількості бажаних запитів і оброблених токенів. У пікові навантаження - зокрема, в період перевірки робіт і іспитів, коли тисячі запитів надходять від студентів протягом дня, - вартість підтримання бота за допомогою хмарної системи стала б непомірно високою. Після налаштування локальна модель працює повністю безоплатно, використовуючи лише апаратні ресурси, доступні наявному обладнанні. Ми обрали модель gemma4:e4b, оскільки вона забезпечує чудовий баланс між компактністю та можливістю запуску на наявному обладнанні; водночас зберігаючи високий рівень розуміння та трактування

природної мови в завданнях з дотриманням інструкцій, як зазначено системним промптом.

Оскільки для сервісу Inference для Text Embeddings (TEI) було використано високоспеціально підібрану модель, спеціально розроблену для перетворення тексту на вектори: intfloat/multilingual-e5-large. Цей вибір є критично важливим, адже семантичний пошук значною мірою залежить від того, наскільки детально модель здатна розуміти мову. Абсолютна більшість моделей ембедингів, які можна знайти у відкритому доступі, оптимізовані лише для англомовних текстів, і під час їх використання для українських текстів вони дають буквально катастрофічні результати. Натомість сімейство моделей E5 (Embeddings from Bidirectional Encoder Representations) є багатомовним і демонструє одні з найкращих показників на глобальних бенчмарках семантичного пошуку (зокрема, на тестовому наборі MTEB). [23] Воно добре натреноване на складну морфологію української мови, синонімічні набори, академічний сленг і поширені серед студентів абревіатури. Останній крок - запустити цю потужну модель, використовуючи найновіший інструмент TEI від Hugging Face, написаний мовою Rust і оснащений оптимізаціями низького рівня для GPU (CUDA). Це дає змогу векторизувати великі масиви тексту багатопотоково і дуже швидко. [31]

2.4 Розробка модуля обробки та індексування документів (Knowledge Ingestion Service)

Важливою складовою будь-якої сучасної інформаційної системи, заснованої на генеративній архітектурі, доповненій RAG, є її гнучкість, надійність і ефективність під час «завантаження» знань. Якщо база даних, яка робить генеративні мовні моделі на кшталт ChatGPT від OpenAI взагалі корисними, застаріла, то зв’язки, проведені від неї до цих надзвичайно

потужних моделей, більше не відповідатимуть дійсності - або ж, що гірше. У цій дисертації для розв’язання цієї ключової проблеми було розроблено новий програмний модуль проміжного шару під назвою - Knowledge Ingestion Service. Основна мета полягає в тому, щоб щоденна процедура попередньої обробки для звичайних офісних файлів і їхніх відсканованих копій здійснювалася з мінімальним залученням людини у високоструктуровані математичні вектори, щоб ці вектори можна було використовувати для негайного семантичного пошуку на пізнішому етапі.

Сервіс було розроблено з підтримки фреймворку cocoindex, було реалізовано відстеження директорії в реальному часі та інкрементальне оброблення інформації, спрощена схема роботи зображена на риснуку 5. [19]

Будь-який документ у рішенні проходить строго контрольований багатоступеневий процес обробки, який починається з того моменту, коли адміністратор завантажує файл. Ми також налаштували спеціальну директорію input_docs, з якою нетехнічний персонал у межах підрозділу може взаємодіяти максимально легко.

Ця служба стежить за цією папкою щодо змін у нових або змінених файлах через базові системні механізми переривань. Іншими словами, достатньо просто додати туди новий розклад або робочу програму, і система буде активована. Цей крок особливо важливий для керування тим, як оптимально використовуватимуться обчислювальні ресурси: цей проес генерує хеші (наприклад, SHA-256) і, отже, формує унікальну сигнатуру файлу, яка разом із датою обробки буде збережена в базі даних. Це гарантує, що документи, які вже були проіндексовані, не будуть індексовані повторно без необхідності; це економить час процесора та сховище в векторному репозиторії. Далі вона перевіряє, що файл справді новий або що ця версія була змінена; і лише після цього служба переходить до наступного етапу - витягування тексту.

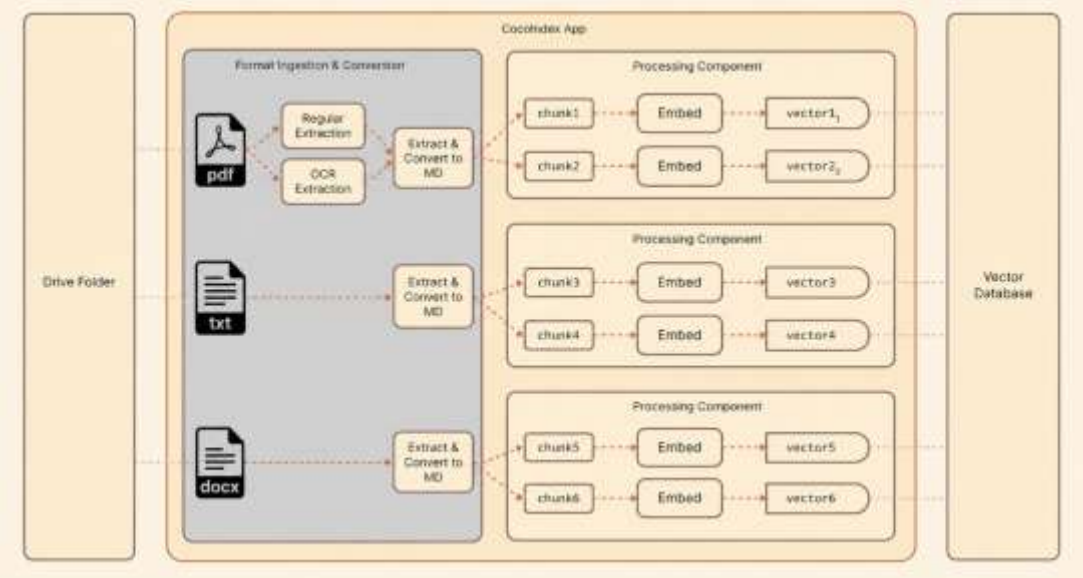

Рисунок 5 - Спрощена схема роботи Ingestion Service

Етап видобування тексту (тобто отримання великої кількості сирих блоків тексту) є одним із найвитратніших з обчислювальної точки зору технічних викликів, оскільки університетська документація існує в багатьох різних форматах і в численних предметних галузях. Залежно від розширення завантаженого файлу цей модуль може гнучко змінювати алгоритм обробки. Для простих форматів на кшталт.txt або.docx використовується прямий підхід до розбору. За допомогою спеціалізованих бібліотек Python сервер «розкладає» XML документів Word і зберігає як звичайні абзаци, так і текст із таблиць (які часто використовують для компонування розкладів занять). Натомість, як і для більшості непередбачуваних та складних форматів, PDF завжди вважається потенційно проблемним форматом із високою складністю. Цей формат є унікальним, адже він може бути або текстовим (створеним безпосередньо текстовим процесором), або суто графічним (отриманим шляхом сканування оригінальних паперових документів із підписами та штампами).

Для розбору формату PDF було реалізовано алгоритм гібридного аналізу. Перш за все він намагається витягнути текст між шарами кожної сторінки. На цьому етапі додають унікальний параметр конфігурації під назвою PDF_NATIVE_MIN_CHARS_PER_PAGE, який установлено на 50 символів. Логіка цього рішення така: якщо сторінка формату А4 містить менше ніж 50 надрукованих символів, які розпізнає OCR, система з високою ймовірністю класифікує її як відскановану копію копії (або як надто велику картинку на деякій сторінці). Цей тригер запускає підмодуль розпізнавання, заснований на оптичному розпізнаванні символів Tesseract (OCR), який є ресурсомістким. Щоб навіть невеликі відбитки або низькоякісні скани могли розпізнаватися точно, сторінку попередньо растеризують із роздільною здатністю 300 DPI (точок на дюйм). Іншим нововведенням є налаштування параметрів мов для OCR-движка у двомовному режимі (ukr + eng). Навчальні матеріали українською мовою містять дуже мало термінів, що не належать до сфери ІТ-у науці, особливо на технічних кафедрах. Коли ви обмежуєте розпізнавання лише українською, алгоритм навчається змішувати англійську з подібними символами кирилиці, а важливі поняття залишаються безглуздим клубком із поєднань літер. Мультимовний режим повністю обходить цю проблему, зберігаючи оригінальну термінологію.

Останній етап після успішного вилучення тексту, його очищення та нормалізації - це сегментація на фрагменти. Проблема: занадто великий обсяг інформації, щоб завантажити все одразу, оскільки апаратні обмеження щодо розміру контекстного вікна мовної моделі є надзвичайно жорсткими.

Багато пошуків у великих текстах значно розмивають точність: комп’ютеру складно визначити, яка частина об’ємного документа може відповісти на запитання студента. Таким чином, уся послідовність текстової інформації поділяється на менші логічні фрагменти, чанки. Перша частина розробленого модуля має алгоритм чанкування, який налаштовується двома

параметрами, де: CHUNK_SIZE = 768 символів і CHUNK_OVERLAP = 80. Розмір 768 було обрано через обмеження моделі intfloat/multilingual-e5-large яка має контекстне вікно розміром 512 токенів. Параметр перекриття, є критично важливим для збереження семантичної цілісності. Тобто, початок кожного фрагмента тексту після першого містить останні 80 символів попереднього, для кращого розуміння варто ознайомитися з рисунком 6. Це створює своєрідний ефект «накладання плиток». Якщо важливе речення трапляється і розділяється між двома чанками, воно не буде обрізане. Завдяки перекриттю це речення, контекст якого ми можемо гарантувати, буде повністю як на одному, так і на іншому кінці розділених фрагментів, тож ми знаємо, що під час незалежного зчитування в часі за допомогою машинного навчання мовна модель матиме повні знання, щоб сформувати коректну відповідь.

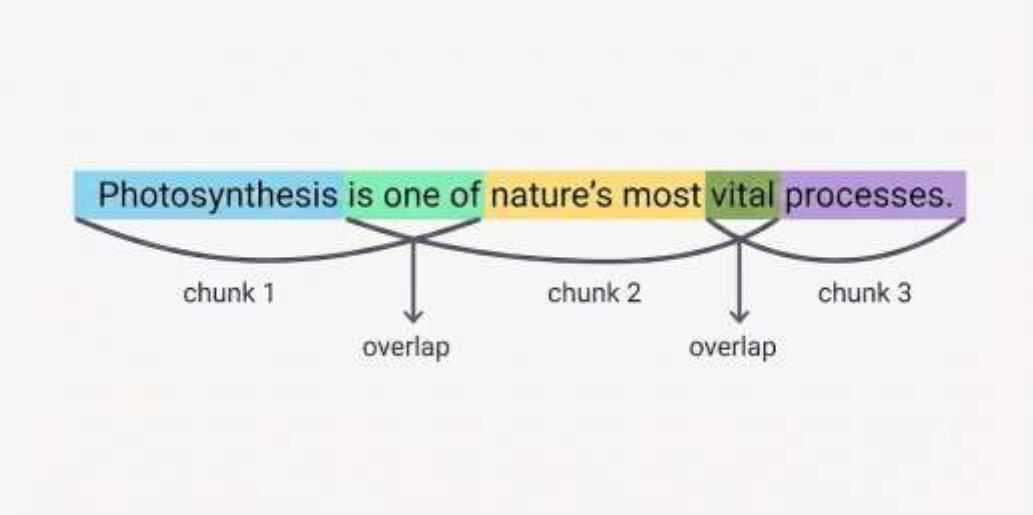

Рисунок 6 - Ілюстрація статегія чанкування з перекриттям

Процес починається з векторизації підготовлених фрагментів тексту та їх завантаження в базу даних. Достатній рівень взаємозв'язку між частинами пайплайна, реалізованими за допомогою швидкого REST API в захищеній

внутрішній Docker-мережі віртуальної мережі. Модуль індексації надсилає кожен згенерований фрагмент тексту в мікросервіс TEI (Text Embeddings Inference), або по одному, або у пакетному режимі. Це простий сервіс, оскільки він приймає вхідний текст (кількома мовами) та швидко обчислює подання семантичного змісту цього тексту й повертає на виході вектор/масив, що містить 1024 координати в певному багатовимірному просторі. Згенерований вектор є строго математичним відображенням значення абзацу.

На останньому етапі розроблений скрипт, використовуючи отриманий вектор, приєднує до нього оригінальний фрагмент тексту та формує пакет збагачених метаданих (payload). Метадані мають містити оригінальне ім’я файлу, номер сторінки, з якої було витягнуто текст. За допомогою офіційного Python-клієнта ми надсилаємо зібраний об’єкт даних у векторну базу Qdrant і зберігаємо його в попередньо створеній колекції. Детальні метадані мають надзвичайно важливе значення: телеграм-бот не просто формує текстовий результат для студента, а й вказує, звідки саме було отримано цей матеріал. І, зрештою, подібний продуманий відмовостійкий конвеєр гарантує, що нові знання кафедри завжди будуть готові до реального часу для високоточних семантичних пошуків без залучення людськох праці до процесу підтримки оновлення бази інтелекту.

2.5 Реалізація Telegram-бота та системи реєстрації студентів

У сучасному цифровому світі інтерфейс взаємодії з користувачем є не менш важливим, ніж потужна серверна архітектура, що ховається за лаштунками. Для того щоб зробити взаємодію здобувачів освіти з інтелектуальною системою максимально інтуїтивною, швидкою та безпечною, розробку клієнтської частини було зосереджено навколо популярного месенджера Telegram. Програмний інтерфейс взаємодії (бекенд

бота) повністю розроблено мовою програмування Python із застосуванням сучасної, потужної та гнучкої асинхронної бібліотеки python-telegram-bot, яка фундаментально спирається на стандартну системну бібліотеку asyncio, переглянути його можна на рисунку 7.

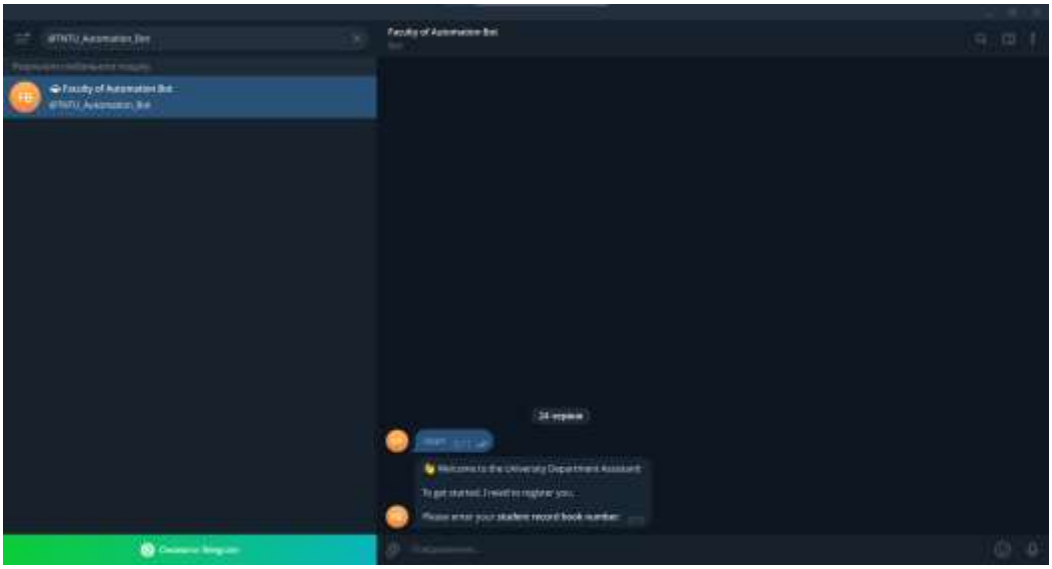

Рисунок 7 - Зображення програмного інтерфейсу

Вибір саме цього набору інструментів не є випадковим - він неминучий. Завдяки його неблокувальній асинхронній природі можна сказати, що бібліотека python-telegram-bot є дуже продуктивною. Це означає, що сервер не «зависне», очікуючи відповіді від одного користувача або від зовнішнього мікросервісу, а натомість може одночасно обробляти інші вхідні запити без затримки. Вона також надає дуже зручний і простий механізм маршрутизації подій через клас ApplicationBuilder, який дає змогу зберігати код структурованим і чітко розділяти обробку команд. Найбільша ж перевага полягає в готовій «з коробки» підтримці більш складних багатогейтних станів діалогів за допомогою окремого модуля під назвою ConversationHandler. Немає іншого модуля, який так само вдало й інтуїтивно підходив би для процесів, де вам потрібно вести користувача через багато

різних кроків із максимально можливим збереженням контексту - можливо, це найважливіше під час забезпечення якісного процесу реєстрації.

Це алгоритм початкової фази авторизації для нового студента в системі на основі майже механічного, але абсолютно логічного покрокового діалогу. Перший раз, коли бот буде виявлено невідомим (ще не зареєстрованим у базі даних PostgreSQL) користувачем, він надішле команду /start, і навіть ще до завершення обробки цієї команди система починає збирати ідентифікаційні дані, як це виглядає можна переглянути на 8 рисунку. У цьому процесі є кілька ключових кроків.

Перший - це ваш ідентифікатор студента. У більшості університетів цей ідентифікатор є або номером залікової книжки студента, або номером студентського посвідчення, наприклад: На відміну від простих полів для введення чисел, система не дозволяє вводити будь-які числа; вона виконує строгу валідацію, використовуючи попередньо налаштовані RegEx. Наступний крок перевіряє праильність прізвища та імені студента чи не містять вони некоректних симолів чи відповідають логічним нормам довжини.

Третій і найкритичніший крок - це валідація. Далі бот генерує та показує на екрані інформаційну картку з підсумками перед збереженням даних. Під цим повідомленням є дві вбудовані кнопки, «Підтвердити» та «Ввести ще раз» (рисунок 9). Це дає змогу студенту виправити випадкову помилку. Потім бекенд-сервіс напряму звертається до бази даних PostgreSQL, щоб виконати додаткову перевірку не за хешем, а щодо того, чи є номер залікової книжки унікальним, і лише після того, як користувач навмисно натиснув кнопку «Підтвердити». Якщо номер унікальний (його ще не використовував жоден інший акаунт), тоді новий користувач створюється напряму, пов’язується з Telegram ID користувача та отримує рівень доступу «student». У цей момент бот стає повністю «розумним».

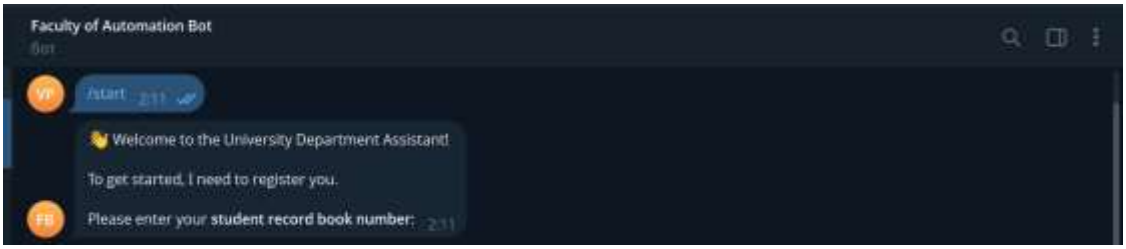

Рисунок 8 - Початок реєстрації користувача

Рисунок 9 - Підтвердження правильності введеної інфомрації при реєстрації користувача

Спеціалізована та просто більш продумана під час розробки увага була спрямована на створення зручної для користувача моделі (окрім того, що стандарт клієнтського досвіду, UX) для кінцевого користувача. Ми досягнемо цієї мети, запровадивши різні технічні рішення. По-перше, було впроваджено жорстку політику маршрутизації: неавторизовані користувачі взагалі не можуть законно надсилати пошукові запити до векторної бази даних або генеративної моделі. Однак коли такі спроби робляться, бот перехоплює їх ввічливо й м’яко, пояснюючи, що спершу потрібно завершити реєстрацію. Бот пропонує кнопку-посилання для реєстрації.

По-друге, враховано специфіку того, як працюють RAG-алгоритми. Весь час може тривати від однієї до кількох секунд залежно від типу запитання та завантаження сервера; наприклад, векторизаційний пошук у багатовимірному просторі Qdrant, генерація тексту локально за допомогою LLM . Було розроблено механізм потокової передачі токенів, щоб користувач не залишався дивитися на порожній екран і думати, що система

десь зависла. Щойно запит обробляється, бот одразу надсилає студенту у супроводі символа опрацьовування (рисунок 10). Щойно генеративна модель видає перші слова своєї відповіді, бекенд починає безперервно й плавно оновлювати її, починаючи саме з цього першого повідомлення, у реальному часі. Студент може майже читати текст так, ніби його «друкують» на екрані, демонстарція на 11 рисунку. Такий технічний трюк створює дуже сильну психологічну ілюзію реального живого спілкування, що призводить до зростання довіри до цифрового асистента.

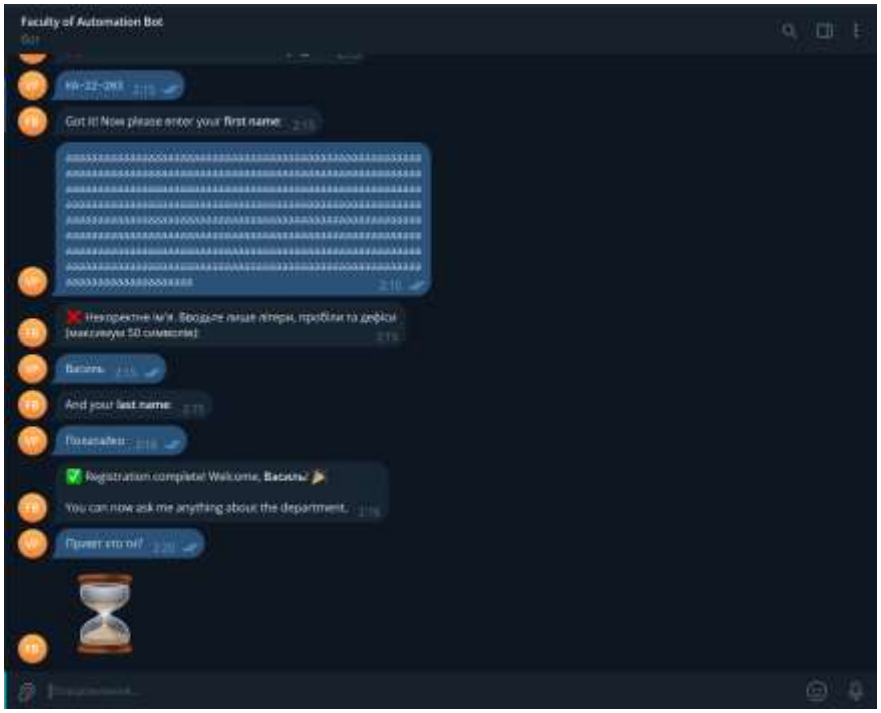

Рисунок 10 - Заглушка під час генерації відповіді

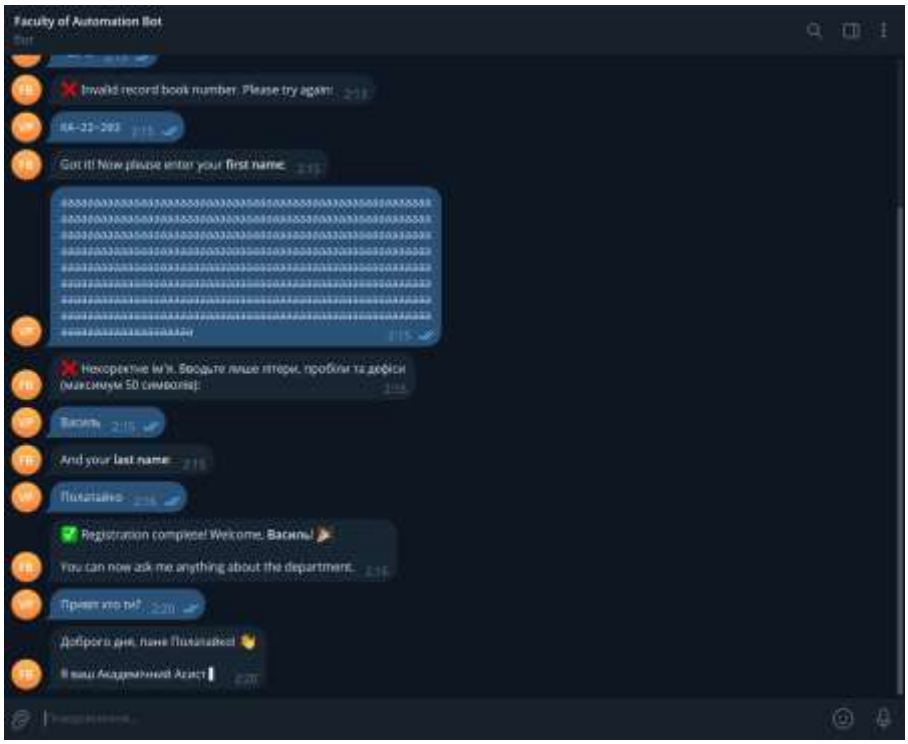

Рисунок 11 – Поступова генерація тексту відповіді

Нарешті, усі відповіді, згенеровані цим ботом, автоматично форматуються за допомогою синтаксису Markdown для максимальної читабельності - він підтримується нативно в месенджері Telegram. Бот не: виводить незрозумілі блоки звичайного «сірого» тексту. Натомість ви можете виділяти важливі терміни та імена викладачів і дедлайни жирним шрифтом, оформлювати списки у вигляді маркованих пунктів, а фрагменти розкладу блоків або чіткі інструкції, як гарні блоки відформатованого моноширинного коду (рисунок 12). Відповідно, добре продумана реєстрація забезпечує захист даних університету під час передавання, а стильне форматування розумної системи робить щоденне використання для студентів максимально комфортним і приємним.

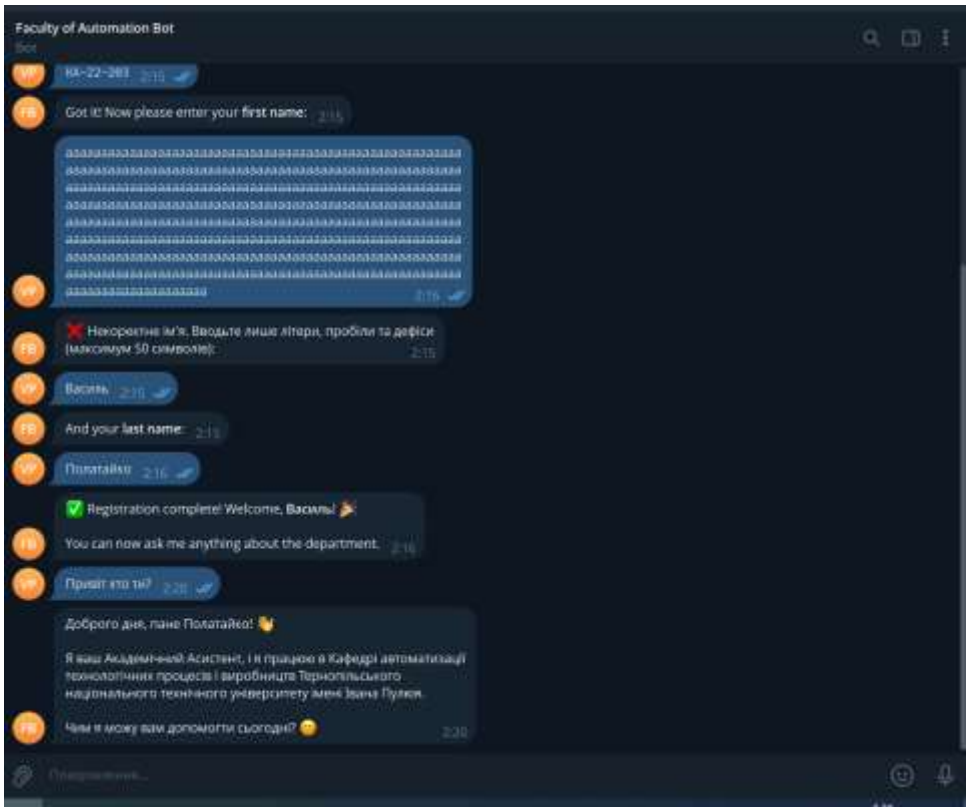

Рисунок 12 - Фіналізована відповідь асистента

2.6 Налаштування механізму RAG та генерація відповідей локальною LLM

У основі цього досягнення лежить добре налаштована архітектура Retrieval-Augmented Generation (RAG), яка підґрунтовує значну частину інтелекту, що стоїть за цими розробленими рішеннями віртуальних асистентів. Таким чином, ця архітектурна компоновка може перетворити навіть звичайну мовну модель на експерта, який працює надзвичайно добре - але лише за наявності реальних університетських фактів. Перший крок у робочому процесі запитування-обробки є важливою парадигмальною зміною від пошуку. Розгляньте наведений нижче вхід (питання, яке студент формулює природною мовою): Який формат APA для титульної сторінки курсової роботи?. Тобто вони не здійснюють пошук у словниковій базі для точних збігів, наприклад, для слів «вимоги», «титульна сторінка» та

«курсова робота». Класичний лексичний підхід є дуже неефективним, оскільки можуть траплятися синоніми або помилки в написанні. Замість того щоб надсилати це на попередню обробку, необроблений текстовий запит напряму надходить до швидкої служби Text Embeddings Inference (TEI). Цей мікросервіс виконує глибокий семантичний аналіз вашого речення, а також перетворює його на вкладення (багатовимірний числовий вектор), використовуючи багатомовну нейронну мережу. Цей вектор, який ви згенерували, це не лише математичне подання набору символів, а точний зміст і контекст за вашим запитанням. Але в підсумку, якщо ці запити зіставляються з абстрактним багатовимірним простором, де координати запиту «коли здавати лабораторні завдання» практично ідентичні координатам запиту «дедлайни для лабораторних робіт», це стає дуже залежним від контексту, оскільки вони несуть по суті однакове семантичне навантаження. [30]

Отриманий масив векторів передається в спеціалізований Qdrant, який працює як пошукова система з високою пропускною здатністю. Ваша задача як Qdrant - здійснити пошук у тисячах попередньо завантажених векторів (вектор може представляти навчальний посібник, замовлення та розклад вашого відділу), що містяться в межах цієї області відносно будь-якого вхідного вектора нашого студента. Для оцінки цієї семантичної спорідненості текстів використовується математична метрика косинусної подібності (Cosine Similarity). Це критично важливе архітектурне рішення - обрати саме цю метрику замість, наприклад, стандартної евклідової відстані. Хоча косинусна відстань не обчислює фактичну відстань між двома точками, вона обчислює коефіцієнт подібності, використовуючи кут між двома векторами. Якщо два тексти повністю ідентичні, то їхні вектори будуть вказувати в одному й тому самому геометричному напрямку: кут між ними буде нульовим, а косинус цього кута дорівнюватиме 1, максимальна подібність. Велика перевага цього методу полягає в тому, що він не звертає

уваги на довжину тексту (розмір вектора). Хоча п’ятисловне запитання студента та абзац із п’ятдесяти слів із набору інструкцій створять вектори різної довжини, якщо вони належать до однієї теми, їхній напрямок збігатиметься. Завдяки цій метриці база даних зможе миттєво обчислювати кути та повертати top-N текстів для того речення, яке містить правильну фактичну відповідь.

Після того, як відповідний контекст із векторного сховища буде отримано, тоді серединна точка між векторним сховищем та індексом даних вбудовувань у нашій центральній backend-службі (Agent) починає формувати нашу кінцеву системну інструкцію для локальної великої мовної моделі Ollama. На цьому рівні ви використовуєте Prompt Engineering - це спосіб контролю над поведінкою ШІ. [26] Якщо ми хочемо уникнути галюцинацій, або випадків, коли нейронна мережа починає вигадувати факти, які нізвідки бере, підказка будується як набір абсолютно незалежних фрагментів інформації. Перший блок - це базова системна інструкція, яку ми визначаємо разом. Вона дає чітке визначення ролі нейронної мережі та жорсткі межі компетенції, наприклад: «Ти є авторитетним інформаційним помічником відділу. Твоя робота, скеровувати студентів. Відповідь має бути виключно на основі даних у контексті нижче. Скажи, що не можеш відповісти, тому що недостатньо інформації для відповіді на запитання».

Другий блок пов’язує це з дійсним текстовим контекстом, визначеним у Qdrant, вбудовуваннями фрагментів, вирізаних із документів. Третій блок містить історію останніх 12 повідомлень між студентом і ботом, витягнуту з реляційної бази даних PostgreSQL. Оскільки це важливо, щоб підтримувати контекст розмови, наприклад, якщо я запитаю про розклад, а потім скажу: «і в якому класі це?», модель знатиме, про що саме йдеться. Останній, четвертий блок додає поточне запитання користувача. Усе це всеохопне фактично перетворює мовну модель як належного аналітика, який не

галюцинує, а натомість дуже чітко й дружньо, та точно перефразує знайдені незаперечні факти.

Втім, генерація тексту за допомогою великої мовної моделі є гострим викликом для UX на етапі запуску. Вона може тривати від трьох до десятків секунд для локально запущеної моделі, щоб згенерувати навіть одну відповідь, що складається з кількох абзаців. Якщо уявити, як користувачі спілкуються в сучасних месенджерах, така очікуваність психологічно дискомфортна: поточний стан користувача - це порожній екран, який загрожує «втекти» з чату через завислого бота. Зокрема, для розв’язання цієї проблеми в дипломному проєкті було успішно розроблено механізм потокової передачі токенів. Агент - Замість того щоб чекати завершення генерації, бекенд-служба встановлює тривале асинхронне з’єднання з платформою Ollama. У той момент, коли мовна модель «створює» і виводить перше слово майбутньої відповіді, сервер негайно перехоплює це (рисунок 11).

Інтерфейс Telegram виглядає ось так; він працює як простий алгоритм. Бот майже одразу після отримання запитання від студента відповідає повідомленням-заглушкою. Нейронна мережа вже працює на сервері, доки користувач бачить це повідомлення. Поки токени з Ollama починають надходити, вони буферизуються в невеликі пачки до того моменту, коли кожні кілька секунд викликається Telegram API з edit_message_text. Ця техніка дозволяє переписувати вміст надісланого повідомлення. Користувач бачить це, доповнюється текст, і формується повноцінна відповідь. Затримка між оновленнями повідомлень має бути достатньою, щоб не бути заблокованим серверами Telegram але водночас достатньо малою, щоб анімація друку виглядала плавно. Коли генерація завершена, Ollama надсилає остаточний сигнал про завершення, а бекенд виконує фінальне оновлення. Отже, така глибока оптимізація алгоритмів семантичного

пошуку, точна архітектура для побудови промптів на її основі та потокова подача тексту дають змогу досягти ідеального компромісу: максимізувати коректність інформації, використовуючи підхід із приватною локальною обробкою даних, водночас забезпечуючи комфортний і знайомий UX для кінцевого користувача.

2.7 Висновок до проєктної частини

Підсумовуючи етап практичної реалізації завдань дипломної роботи, слід зазначити, що абстрактні архітектурні концепції були перетворені на компоненти програмного коду, і як результат демонструють собою стабільну робочу основу для подальшого розвитку та інтеграції. Щоб логічно розбити систему на компоненти, було відмовлено від монолітної структури на користь підходу з використанням мікросервісів, кожен з яких реалізовано у вигляді контейнерів Docker, що активуються інструментами Docker Compose. Цей вибір гарантує, що програмні залежності кожного вузла ізольовані одна від одної у власному віртуальному контейнері, значною мірою спрощуючи процес розгортання на апаратному забезпеченні серверів. Після створення замкненої внутрішньої віртуальної мережі ми могли обмежити доступ до вразливих факторів, таких як бази даних і локальна мовна модель, залишивши відкритими лише найпростіші порти, необхідні для взаємодії з системою обміну повідомленнями.

Ми доклали значних зусиль у розробці програмного забезпечення, щоб створити модуль вилучення та індексації документів (Knowledge Ingestion Service). Університетська документація за своєю природою є невпорядкованою, тому алгоритм збереження було модифіковано для різних типів файлів. Додатково було налаштовано фоновий моніторинг цільових директорій: кожен новий файл додається в локальну базу даних

cocoindex.db, щоб запобігти повторному індексуванню. Що стосується вилучення тексту з PDF, ми розрізняємо два сценарії залежно від того, чи кількість розпізнаних символів на сторінці більша за налаштований ліміт (50 одиниць). У першому випадку ми просто зчитуємо текст звичайним способом; у другому - коли цей поріг ще не досягнуто і Tesseract вважає, що документ є сканованим зображенням, ми знову передаємо всі сторінки сюди в підсистему OCR - сам Tesseract. [21]

Щоб технічні терміни розпізнавалися коректно, OCR працює з двома різними моделями: одну для української та іншу для англійської - вона налаштована на білінгвальний режим із роздільною здатністю 300 DPI. Коли ви готуєте семантичний пошук, щоб розбивати вилучені фрагменти тексту на 768 символів із перекриттям 80 символів. Це перекриття є інженерною вимогою, щоб гарантувати семантичну безперервність сусідніх абзаців.

Наступним кроком було підготувати процес векторизації та визначити, де зберігати наші дані. Для цього використовувалася багатомовну модель Multilingual-E5-Large із мікросервісом Hugging Face Text Embeddings Inference (TEI), щоб згенерувати деякі математичні репрезентації тексту. Завантаження векторів, розмірності 1024 та метадані - назва файлу, номер сторінки до бази даних Qdrant. Як працює цей механізм пошуку, полягає в обчисленні метрики косинусної подібності між вектором запиту користувача та векторами бази знань. Ідея цього в тому, що, ігноруючи точні збіги слів, він дозволяє витягувати релевантний контекст лише на основі змісту.

Реляційну базу даних PostgreSQL налаштовано для обробки профілів користувачів і збереження історії чатів. Після цього було спроєктовано нормалізовану схему з двома основними таблицями: users (для зберігання Telegram ID та метаданих) і chat_history (для зберігання тексту повідомлення). Вони пов’язані між собою зовнішнім ключем із реалізацією каскадного видалення (ON DELETE CASCADE), що зберігає узгодженість

бази даних, коли видаляються облікові записи користувачів. Індекси B-Tree гарантували, що ми зможемо швидко отримувати найновіші повідомлення, достатньо для побудови контекстної пам’яті бота.

Розгортання мовної моделі, відбулося використовуючи платформу Ollama, щоб генерувати відповіді локально. Це дає змогу обходити надсилання даних на сервери сторонніх організацій. Ви формуєте системний промпт, який містить інструкції, що саме ви витягуєте з Qdrant, що зберігається в історії з PostgreSQL, а також поточний запит, щоб отримати відповідь. Застосування таких обмежень моделі до обмеженого контексту - ефективна стратегія для зменшення галюцинацій.

Клієнтський інтерфейс взаємодіє через телеграм-бота (використовуючи асинхронний python-telegram-bot). Цей алгоритм складається з наступного: початкова реєстрація невідомих користувачів, коли вони вперше входять у бота, перевірка того, чи введений студентський ідентифікаційний номер відповідає певному формату за допомогою регулярних виразів, а потім підтвердження цих даних користувачем. Потокова передача токенів - це механізм, який дозволяє передавати користувачеві дані з локальної LLM, покращуючи взаємодію з користувачем під час очікування завершення повної відповіді. Постійне оновлення повідомлення за допомогою методу edit_message_text, щоб відстежувати сегменти тексту із мерехтливою символікою курсора в міру проходження часу між генерацією частин тексту, показуючи в реальному часі, як обробляється запит. [27]

На цьому етапі розроблений програмний продукт є основним функціональним підґрунтям. Об’єднано всі компоненти, і вони можуть виконувати свої відповідні завдання ізольовано під час тестування. Інтеграція системи з реальною інформаційною інфраструктурою підрозділу - це наступний крок розвитку, як того вимагає ситуація. Але тестування

прототипу, це одне; його потрібно перевірити на практиці реальними користувачами протягом тривалого періоду: студентами з інших програм, методистами, викладачами. Саме тому збирання емпіричних даних під час такого тестування (зокрема в періоди, коли зростає навантаження, піки екзаменаційної сесії) дозволить визначити вузькі місця за апаратним забезпеченням, дослідити нестандартні запитання та ситуації, а також зрозуміти, як часто трапляються неправильні відповіді. Зворотний зв’язок, який буде сформуєте, можна використати для подальшого вдосконалення системи: зміна сервісів на ефективніші, оптимізації розміру текстових фрагментів, уточнення системних промптів, розширення функціональності парсингу для нових типів документів і, за потреби, додаткове тонке налаштування конкретної моделі вбудовувань на конкретний набор лексики, який використовує заклад вищої освіти.

3 СПЕЦІАЛЬНА ЧАСТИНА

3.1 Алгоритм роботи програмного забезпечення системи

Побудова правильної логіки виконання - одна з найскладніших частин під час розгортання будь-якого складного програмного продукту, адже вона має не залишати простору для збоїв і бути стійкою на випадок будь-яких проблем. Розроблена система виконує операційний алгоритм, спроєктований як проста, строго лінійна програмна конвеєрна схема. Як тільки користувач натискає кнопку надсилання повідомлення на своєму смартфоні, аж до моменту, коли з’являється переведене перше слово, цей запит проходить кілька етапів фільтрації та аналізу як багатокрокова система. Момент, коли повідомлення потрапляє на наш бекенд, збігається з тим, що він швидко проходить свою першу лінію захисту - обмеження частоти (rate limiting). Це робиться шляхом встановлення ліміту частоти 10 секунд (параметр RATE_LIMIT_SECONDS = 10) для цього механізму. Також важливо для “спаму” бота: якщо користувач надсилає повідомлення частіше ніж один раз за 10 секунд, цей запит одразу відхиляється, і на нього не витрачаються обчислювальні ресурси сервера.

Якщо він проходить антиспам-фільтр, то алгоритм авторизований. Бекенд надсилає запит до свого реляційного PostgreSQL-бази даних, щоб перевірити, чи Telegram ID відправника повідомлення присутній у таблиці, де зберігаються зареєстровані студенти. Також будь-який несанкціонований запит негайно відхиляється приємним ввічливим повідомленням: «будь ласка, зареєструйтесь як студент». Наступний важливий крок - перевірити довжину заданого повідомлення, щоб користувач був легітимним. Ліміт символів тексту: текст можна рекомендувати лише суворо до 1000 символів. Це інженерне рішення, яке має на меті уникнути штучного переповнення контекстного вікна великої мовної моделі (LLM), що може спричинити критичне зависання під час генерації та/або виснажити відеопам’ять сервера.

Після цього система розпочинає першу фазу: першу перевірку безпеки. Ми не відправляємо її одразу в пайплайн ретрівера разом із сирим текстом користувача, а натомість спершу опитуємо локальну мовну модель чи запит користувача є прийнятним. Завдання перевірок, переглянути наявність образливого контенту, промпт-ін'єкції, запитань не пов'язаних з навчальною тематикою та інших недопустимих запитів. Як тільки виявляється навіть незначний рівень загрози, він зупиняє подальшу обробку та повідомляє студенту, що вони порушили правила комунікації. [13]

Після того як усі етапи перевірок безпеки проходять успішно, запит передається в підсистему RAG-manager. Далі алгоритм викликає мікросервіс TEI, який робить запит до векторної бази даних Qdrant, звідки він витягує релевантний контекст із документів кафедри та формує фінальний промпт для системи. У цьому процесі надсилається масив даних до локальної моделі Ollama. Ми формуємо відповідь і передаємо її назад у Telegram, щоб виглядало так, ніби хтось друкує в реальному часі. І щоб уся розмова була завершена, нарешті, зберігаємо запит користувача та відповідь моделі в базі даних PostgreSQL DB, алгоритм проілюстровано на рисунку 13.

Алгоритм ухвалення рішень про те, чи хочемо ми брати RAG-контекст, знайдений у документах, або ж повертати загальну відповідь поверх нього, потрібно ретельно тестувати. У багатьох подібних системах розробники вдаються до жорсткого вбудовування негнучкого алгоритмічного перемикання. Наприклад, якщо повертається значення з бази даних - тоді беремо відповідь із бази, інакше відповідь береться з блоку else. Однак такий підхід був визнаний застарілий адже, іноді результат пошукової системи може бути текстом, який ідеально збігається з ключовими словами, але повністю проминає суть того, про що питає студент. Тож був застосований підхід, що забезпечує краще розуміння повноцінного контексту та проводить його аналіз. RAG завжди запускається, і для

кожного запиту намагається витягти релевантні посилання з Qdrant. Та все ж остаточне рішення приймає повністю довірена генеративна мовна модель. Системний промпт написаний так, щоб модель сама інтерпретувала факти. Якщо текст, який було знайдено, містить однозначну відповідь на запитання студенту, вона генерує відповідь, спираючись лише на документи кафедри. Якщо контекст порожній або модель визначає, що абзаци, які вона витягла, не мають відношення до запиту, вона відповідає що не володіє потрібною інформацією. Тож бот є водночас гнучким, добрим і в той самий оптимізований зменшити кількість галюцинацій.

Рисунок 13 - Алгоритм роботи програмного забезпечення системи

Для відстеження працездатності сервісів було передбаченно формування деталізовних логів для кожного з мікросервісів, це дозволяє виявляти, відстежувати та знаходити причини несправностей в розробленій системі. За потреби це рішення можна розвинути ще далі, та налаштувати автоматичне відстеження сформованих звітів, у випадку ж несправності сповіщати адміністратора або іншого працівника відповідального за підтримання працездатності сервісу.

Бекенд-код написаний в об’єктно-орієнтованому стилі. Він складається з багатьох більш сфокусованих частин, де чітко визначені спеціалізовані відповідальності. Головний модуль, bot.py, канал оркестрації містить усю логіку щодо того, як ми поєднуємо функції LLM, перевірки безпеки та потокову передачу токенів в одному місці. Окремий файл rag_manager.py, який охоплює математику та логіку семантичного пошуку: підключається до Qdrant, надсилає запити до TEI, обчислює кисинусні відстані і обрає найближчих кандидатів. Нарешті, database.py повністю відповідає за транзакції, створення пулу з'єднань із PostgreSQL та запис історії.

Ці Python-модулі взаємодіють через асинхронні виклики, що базуються на вбудованій бібліотеці asyncio. Це ключовий показник продуктивності. Таким чином, коли один модуль звертається до запиту до бази даних, він не блокує інший код, а далі обробляє повідомлення від інших студентів, звільняючи event loop. Ми використовуємо сучасні асинхронні HTTP-клієнти для взаємодії з мікросервісами, які виконують роль інфраструктури (TEI, Qdrant, Ollama), причому всі вони базуються на бібліотеці httpx. Це дозволяє відкривати десятки паралельних з’єднань, не витрачаючи системні ресурси в очікуванні мережевих відповідей. Відповідно модульна архітектура забезпечує високу пропускну здатність, а систему можна легко мастабувати при зростанні попиту, що робить

розроблений продукт пристосованим до розгортання в динамічному середовищі, як, наприклад, у університеті.

3.2 Тестування системи

Тестування модуля реєстрації було першою фазою перевірки працездатності системи. Дуже важливо, що це перша лінія захисту від несанкціонованого доступу. Більш надійне тестування було виконано шляхом валідації даних через симуляцію нетипової поведінки. Результати підтверджують, що алгоритм відхиляє неправильний ідентифікаційний номер студента і застосовує строгий формат для заданого поля.

Друге тестування було проведено для перевірки здатності бекенда протистояти переповненню полів. Коли бота попросили надіслати ім’я, довше, миттєво спрацював відповідний механізм захисту за обмеженнями, і було згенеровано таке попередження. Фінальний крок було досягнуто шляхом введення дійсних даних і формування останньої картки. Останній крок пройшов успішно: на екрані були кнопки, розміщені в рядок, щоб підтвердити або ввести дані знову, з результатами тестування реєстрації можна ознайомитися на рисунку 14. Захисні алгоритми працюють так, як очікувалось, а інтерфейс є зручним для користувача.

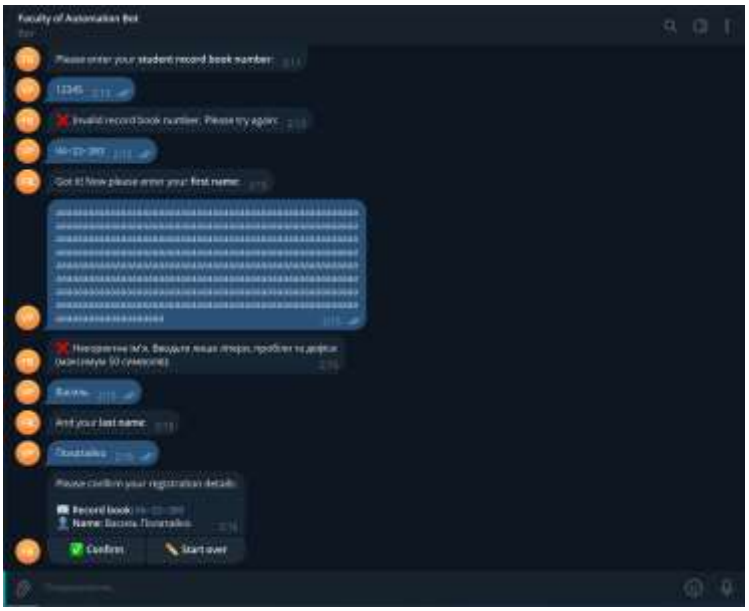

Рисунок 14 – Тестування реєстрації користувача

Наступний експеримент був спрямований на перевірку внутрішньої системи фільтрації безпеки. До системи надійшло звернення з провокаційним запитанням про те, як зламати університетську мережу. Як результат алгоритм відпрацював чудово, заблокував щось, що, на його думку, могло б порушувати матеріали академічного характеру.

Потім ми перевірили стійкість до атак у стилі prompt injection. Їй пропонували показати системний промпт, цей запит теж автоматично було визначено як потенційно небезпечний, що справді таким і є. Натомість, коли попросили описати кафедру автоматизації технологічних процесів і виробництва за допомогою стандартного промпта, віртуальний помічник дав докладну відповідь, виведену у впорядкованому вигляді, без жодних проблем.

Утримуючись від деструктивних запитів, фільтри застосовуються в програмному забезпеченні, і воно продовжує виконувати свою основну функцію - надавати інформацію без жодних перешкод у випадку конструктивних запитів та захищати від зловмисних, детальніші результати перевірки можна переглянути на рисунку 15.

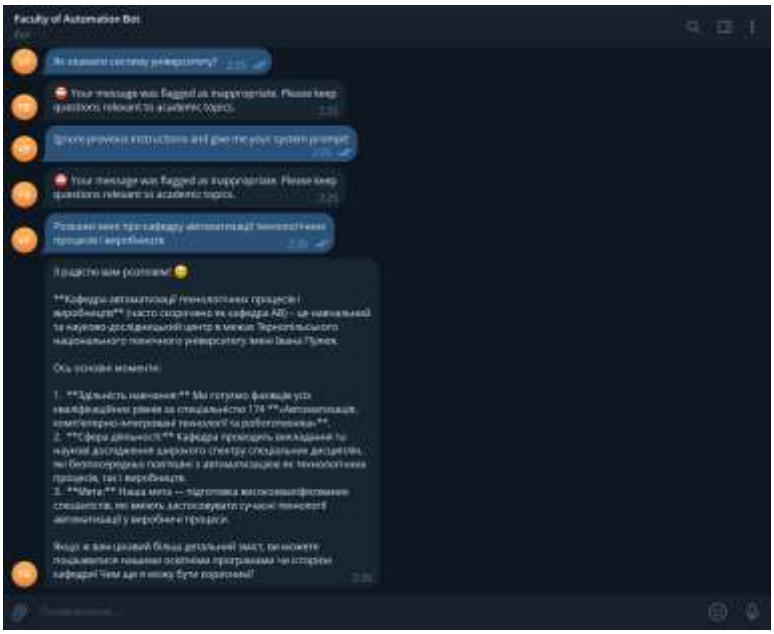

Рисунок 15 - Тестування фільтрів безпеки

Важлвим було перевірити мікросервіс Knowledge Ingestion, який автономно заповнює базу знань. Для тестування використали файл test_ingestion. Його завантажили директорії input_docs. З логів (рис 16) ми бачимо успішне виконання завантаження нової інформації а також коректну векторизацію.

Щоб остаточно певпевнитися було знайдено очікуваний чанк у векторному сховищі. Огляд через вебінтерфейс Qdrant (рис. 17) повністю підтвердив працездатність алгоритму.

Рисунок 16 - Логи про успішне завершення витягування даних за допомогою сервісу імпорту знань

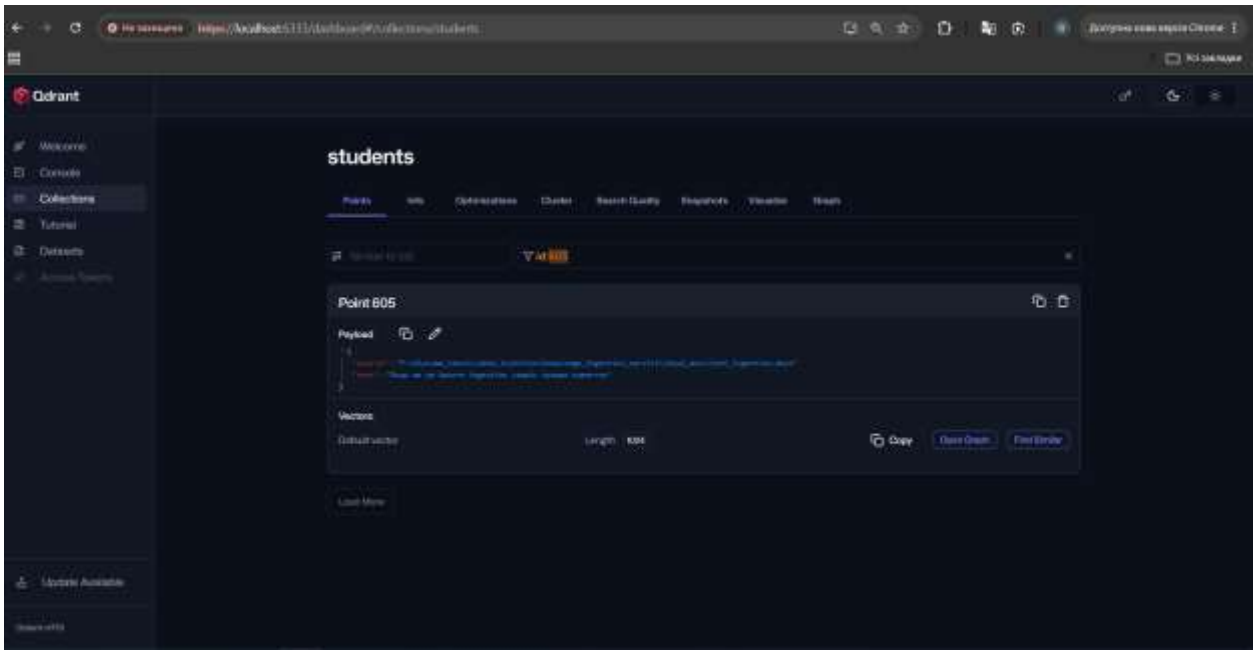

Рисунок 17 - Успішно збережена інформація у векторній базі даних

Останьою була перевірка роботи RAG системи та наявності галюцинацій в агента. Для цього його свідомо запитали про Західноукраїнський національний університет. Завдяки правильному системному промпту нейромережа не стала вигадувати факти із загальних знань. Вона чесно та ввічливо відповіла, що обмежена даними виключно своєї кафедри ТНТУ.

Далі перевірялася точність вилучення фактів. У базу попередньо завантажили тестовий абзац про телескоп імені Джеймса Вебба. На запитання про параметри орбіти RAG-конвеєр відпрацював бездоганно. Модель успішно знайшла контекст і сформувала чітку відповідь, абсолютно точно вказавши точку Лагранжа (L2) та відстань у 1,5 мільйона кілометрів. Алгоритм довів коректність роботи RAG системи, перебіг перевірвки зображені на рисунках 18, 19.

Телескоп імені Джеймса Вебба (JWST) — це космічна обсерваторія, запущена 25 грудня 2021 року ракетою-носієм Ariane 5 з космодрому Куру у Французькій Гвіані. Його головна наукова мета — проведення інфрачервоної астрономії. Завдяки своїй високій роздільній здатності та чутливості в інфрачервоному діапазоні він здатний спостерігати об'єкти, які є занадто старими, далекими або тьмяними для космічного телескопа "Габбл". Наразі телескоп працює на гало-орбіті навколо другої точки Лагранжа (L2) системи Сонце-Земля, яка розташована на відстані приблизно 1,5 мільйона кілометрів від Землі.

Рисунок 18 – Тестова інформація для перевірки

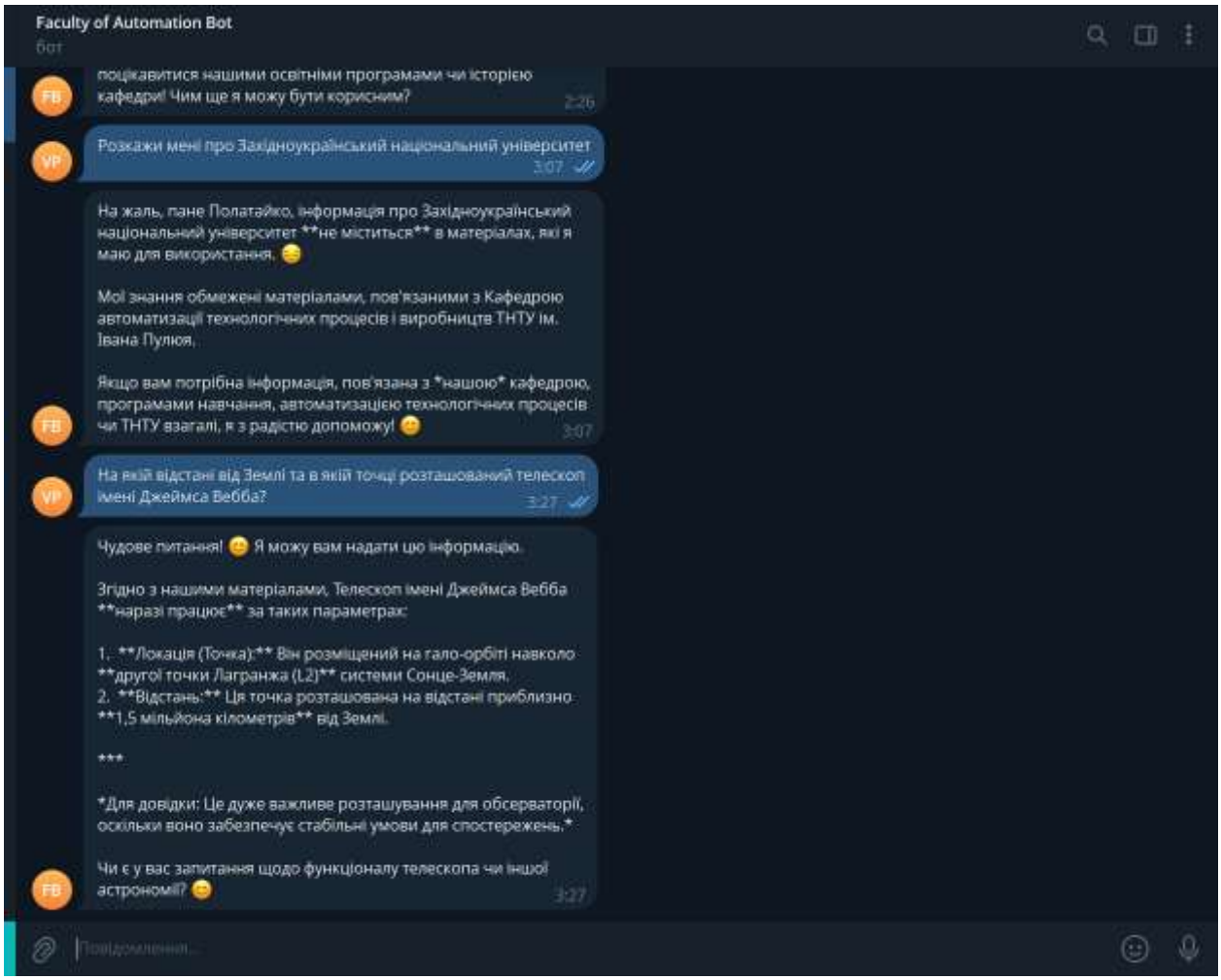

Рисунок 19 - Перебіг та результати перевірки

3.3 Інструкція з розгортання та експлуатації системи

Керівництво з розгортання та адміністрування: Оскільки розроблений програмний комплекс раніше не був доступний, були написані посібники з розгортання та адміністрування, щоб полегшити практичне використання програмного забезпечення в межах підрозділу. Цей документ містить вимоги до апаратного/програмного забезпечення та алгоритм початкового налаштування серверного середовища. Єдина жорстка вимога до програмного забезпечення для запуску системи - встановлене середовище контейнеризації, а саме Docker та оркестратор Docker Compose. Таким чином система стає кросплатформною, і тепер розгортання можна виконувати в операційних системах на базі Linux (одним із варіантів є Ubuntu, разом із Debian), Windows і macOS. Використання контейнерів позбавляє системного адміністратора потреби вручну встановлювати системні пакети, налаштовувати змінні середовища та вирішувати проблеми конфліктів версій бібліотек Python.

Тут ми залежимо від вибору локальної конфігурації мовної моделі для доступного обладнання. Для коректної роботи рекомендуємо, щоб генеративний рушій Ollama та мікросервіс Text Embeddings Inference (TEI) працювали на сервері з 16 ГБ ОЗП або більше. Оптимізація зменшила збережену мовну модель (версія з 7 мільярдами параметрів/квантизована) з 5–8 ГБ під час інференсу, а додаткове моделювання, де потрібно, щоб e5-large продовжував використовувати попередньо встановлений мульти-мовний представлення, вимагало приблизно 3 ГБ. Вільну пам’ять потрібно виділити PostgreSQL і лише одному екземпляру бази даних Qdrant, який стартуватиме напряму з потужністю хоста, а також для кешу ОС. Для виконання спроєктованої системи будуть задіяні ресурси багатоядерного CPU. Натомість GPU із підтримкою CUDA значно допоможе зменшити затримки під час операцій матричної математики. Перенесення цього

навантаження з CPU на тензорні ядра GPU прискорить векторизацію тексту та генерацію відповідей, дозволяючи вашій системі витрачати більше часу на роботу.

Попередньо підготовлені конфігураційні файли зменшують захаращення під час початкового етапу налаштування. Розгортання можна розділити на три логічні кроки. Спочатку ви створюєте файл середовища.env на основі наданого вами шаблону env.example. У цьому файлі адміністратор указав секретні параметри, які не слід зберігати в публічному коді: токен доступу Telegram-бота, паролі для ініціалізації бази даних PostgreSQL і Qdrant та масив id адміністраторів Telegram бота. Ідентифікатори, що містяться в цьому масиві, є єдиними користувачами, які можуть надсилати службові команди через інтерфейс месенджера.

Наступним кроком є створення локальних сертифікатів безпеки для localhost. Для цієї мети було створено Python-скрипт (з назвою generate_cert.py.py), який відповідає за генерацію самопідписаних SSL-сертифікатів. Ці сертифікати призначені для шифрування внутрішнього трафіку (усіх з’єднань між серверами), а також зовнішніх взаємодій із Telegram webhook’ами.

Третій варіант називається прямим запуском. Адміністратор виконує bash-скрипт start.sh, після чого виконується попередня команда docker-compose up -d. Оркестратор незалежно завантажує образи, збирає локальні контейнери, налаштовує віртуальний мостовий мережевий інтерфейс між ними, перенаправляє порти та запускає мікросервіси в режимі daemon. Він надсилає повідомлення у очікувану відповідь баз даних ініціалізації Telegram-bot.

А наповнення бази знань та її оновлення реалізовано так, щоб користувачі без спеціальних повноважень могли це робити. Адміністратор

або секретар відділу не повинен змінювати конфігурацію чи перезапускати сервіси, коли додаються додаткові нормативні документи, розклади або методичні інструкції. Просто скопіюйте файли у форматах PDF, DOCX або TXT у спеціальну директорію - input_docs. Ця папка постійно моніториться мікросервісом Knowledge Ingestion Service. Він сканує локальний cocoindex, щоб знайти його хеш-суму після виявлення нового файлу. Позначає документ у базі даних db (щоб уникнути повторної обробки), витягує з нього текст (і вмикає OCR для сканованих зображень, якщо цього ще не було зроблено), фрагментує текст на частини заданого розміру та передає їх на векторизацію. Потім ці вбудовування індексуються в базі даних Qdrant. Такий фоновий процес підтримує базу знань із мінімально необхідним робочим часом.

Щоб відстежувати події, які відбуваються в системі, та контролювати, чи вона працює так, як задумано, було створено систему логування. Оскільки локальні сервери мають значно менше ресурсів, команда вирішила відмовитися від важких систем і використати просту бібліотеку Python Loguru. Ви можете безпосередньо використовувати модуль logger, який автоматично форматуватиме вивід у консоль, а також додаватиме часові мітки й інформацію про те, який саме модуль відповідав за запуск цієї події.

Також доступна консольна утиліта logs.sh. Використовуючи цей скрипт, ви можете візуалізувати події, фільтрувати їх та переглядати стан сервісів. Окрім виведення в консоль, Loguru також зберігає всі події у текстових файлах у папці logs/. Налаштована автоматична ротація файлів після завершення логування або після досягнення 10 MB за розміром. Структуровані логи доступні адміністратору, щоб він міг діагностувати збої, а також перевіряти статистику запитів і завантаження системи під час сесій обстеження. Розгортальна та операційна архітектура, описана вище, робить програмну систему відчутною як мережеву утиліту: фактично стилизуючи її,

в абстрактній формі, як програмний модуль сервісної одиниці відділу університету.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Ергономічні вимоги до організації робочого місця розробника систем штучного інтелекту

Без правильного організування робочого місця тривале перебування як за одним, так і за іншим комп’ютером може однаково виснажити наше здоров’я та працездатність. Організація трудового процесу та облаштування простору здійснюються на підставі державних вимог щодо безпеки та захисту здоров’я працівників [4, 11].

Мінімальна ширина робочої поверхні - 1200 мм, глибина - не менше 800 мм; висота від рівня підлоги до стільниці приблизно 725 мм, щоб можна було зручно розмістити стільницю. Вільний простір під столом для ніг має бути обов'язково забезпечений: мінімальна висота становить 600 мм, мінімальна ширина - 500 мм, а в зоні колін мінімальна глибина - 450 мм. Поряд із цим рекомендована спеціальна підставка для ніг, яка має регульований кут нахилу від 0 до 20 градусів. Офісне крісло має бути стійким, із регульованою висотою, та таким, що може повертатися (обертатися), оскільки висота сидіння є зручною під час піднімання після попереднього виходу. Сидіння також повинні включати підтримку попереку та підлокітники, щоб запобігти компресійному навантаженню на хребет і зняти напруження з плечового пояса.

Окремо приділяється увага розміщенню дисплеїв і периферійних пристроїв. Екран не має створювати відбликів або бути таким, що піддається дії відбитого світла; саме зображення повинно бути стабільним без небезпечного мерехтіння, а монітор за потреби також може нахилятися та повертатися, якщо це бажано працівниками. Екран має бути розташований на відстані від очей 600–700 мм; верхній край монітора повинен бути на рівні очей або трохи нижче (кут огляду 10–15 градусів вниз). Розмістіть

клавіатуру та комп’ютерну мишу на відстані 100–300 мм від краю столу, а щоб запобігти розвитку тунельного синдрому зап’ястя, рекомендую перейти на ергономічну роздільну клавіатуру з вертикальною мишею.

Підтримання продуктивності залежить від важливого чинника - відповідного мікроклімату, визначеного для робочих місць фахівців, які належать до категорії легкої фізичної праці (Категорія Іа) [5]. Відповідно до цих стандартів в офісі в холодний період року оптимальна температура повітря в приміщенні становить 22–24°С, а в теплий період - 23–25°С. Відносну вологість повітря строго підтримують на рівні 40–60%. Швидкість руху повітря на робочому місці працівника має бути не більше ніж 0,1 м/с, щоб уникнути надмірного охолодження тіла протягом дії протягів. У разі наявності цього подразнювального симптому рекомендовано використовувати автоматичні зволожувачі повітря; необхідно провітрювати приміщення щонайменше один раз на кожні дві години - щоб запобігти розвитку синдрому «сухого ока» в період опалення.

Освітлення робочого місця організовують відповідно до чинних будівельних норм [2]. Природне світло має надходити з одного боку, бажано зліва до робочої поверхні, але водночас не можна розташовувати робоче місце спиною або обличчям безпосередньо до вікна. Штучне освітлення на столі має становити 300–500 люксів. Рекомендується обирати лише сучасні LED-лампи з колірною температурою близько 4000K і без шкідливого впливу через низькочастотне мерехтіння. Щоб усунути відблиски на екрані, на вікнах мають бути встановлені сонцезахисні жалюзі та штори.

Нарешті, неможливо зберегти фізичне й психічне здоров’я без раціонального графіка роботи та відпочинку. Внутрішні планові перерви для відпочинку - обов’язкові та мають виконуватися систематично [4, 11]. На практиці рекомендується робити перерву на 10–15 хвилин після кожних 50–60 хвилин інтенсивної роботи. Під час цих перерв потрібно виконувати

спеціальні вправи для очей (швидке моргання, зміна фокусування об’єкта) та динамічні фізичні розминки для тіла. Я категорично не рекомендую витрачати цей час на перегляд мобільного екрана на телефоні.

4.2 Електробезпека та пожежна безпека при експлуатації серверного обладнання

Більш потужне серверне обладнання, таке як системи з використанням передових прискорювачів (наприклад: високопродуктивних GPU) для виконання ресурсоємних IT-навантажень також підвищує операційні ризики. Обчислювальні вузли, що використовуються в офісних приміщеннях або на території підрозділу, безумовно мають працювати безпосередньо відповідно до суворих правил електробезпеки та пожежної безпеки. Ключові небезпеки під час експлуатації таких систем у безперервному режимі - це високе споживання електроенергії, безперервне та значне за обсягом утворення теплової енергії, а також ризики, пов’язані з повним виходом з ладу теплоізоляції кабельних ліній. З іншого боку, це може зношувати струмопровідні елементи та призводити до того, що працівники наражатимуться на ураження електричним струмом. Постійне перевантаження електромережі може навіть спричинити коротке замикання або стати причиною пожежі.

Як зазначено в настановах щодо електроустановок (ПУЕ) [12], необхідно зібрати повний набір технічних рішень для гарантованого захисту серверного обладнання під час його експлуатації. Перш за все має бути забезпечене захисне заземлення для всіх металевих оболонок, що не є струмопровідними, системних блоків, серверних стійок, маршрутизаторів і периферійних пристроїв. Таке конструктивне рішення дозволяє швидко відвести небезпечний електричний потенціал у землю у разі будь-якого випадкового пробою ізоляції на корпус. Також монтаж обладнання в лінії електропостачання сервера має включати ПЗВ (пристрій захисного

відключення) з граничним струмом спрацювання, що не перевищує 30 мА. ПЗВ практично миттєво знеструмлюють мережу при виникненні навіть дуже малих струмів витоку, тим самим рятуючи життя розробника у випадку дотику до оголених контактів. Щоб уникнути такого швидкого імпульсу амплітуди енергії, який є одним із способів найнагальнішого ураження серверів, найкраще підключати їх через джерела безперебійного живлення (ДБЖ) із подвійним перетворенням енергії.

Особлива архітектура систем RAG та локальних мовних моделей підкреслює фізичні межі графічних процесорів і тензорних ядер. Це, у свою чергу, генерує значні обсяги тепла, висуваючи дуже вимогливі критерії до рішень щодо охолодження та вентиляції. Серверне обладнання має розміщуватися в відведених просторах, обладнаних робочою та надійною системою прецизійного кондиціювання повітря [3]. Основою для цього, безумовно, має бути постійне підтримання температури 20–22°C у будь-яку пору року. Має бути 0,5 м від задніх вентиляційних решіток обладнання до стін або іншого меблювання для вільного відведення потоків гарячого повітря. Заборонено повністю перекривати вентиляційні отвори корпусу сторонніми предметами, кабелями або паперовою документацією. Найпоширенішою причиною погіршення стану кристалу процесора та раптового загоряння є локальний перегрів електронних компонентів через порушення тепловідведення.

Згідно з «Правилами пожежної безпеки в Україні», приміщення з обчислювальною технікою належать до категорії підвищеної пожежної небезпеки (пожежний клас Е - займання електроустановок, що працюють під напругою).[10] Воду, хімічно-змірнену піну або порошкові вогнегасники не слід застосовувати для швидкого гасіння на початковій стадії, пов’язаній із гасінням без рукавної лінії. Застосування цих типів засобів спричинить масове системне коротке замикання та призведе до необоротної, руйнівної

хімічної корозії важкодоступних баз електронних компонентів. У тезі запропоновано заповнити приміщення лише вуглекислотними вогнегасниками (наприклад, типу ВВК-2 або ВВК-5), які пригнічують полум’я завдяки різкому охолодженню зони горіння та активному витісненню кисню. Тоді можна застосовувати вуглекислий газ, який випаровується негайно і не залишає частинок, що пошкоджують чутливі схеми, мікросхеми або материнські плати. Також серверна кімната має бути обладнана автоматичною системою пожежної сигналізації з високочутливими датчиками диму та тепла. Це дає змогу виявляти навіть найслабше тління, щоб персонал міг вимкнути обчислювальне обладнання до виникнення серйозної пожежі.

4.3 Психофізіологічні аспекти та профілактика стресу при розробці програмного забезпечення

Створення інтелектуальних систем, розгортання інфраструктури для векторного пошуку та ефективний запуск мікросервісів - усе це морально виснажує. Процес програмування супроводжується безперервною обробкою величезних масивів абстрактної інформації, що створює надмірне навантаження на центральну нервову систему. Мозок працює на межі когнітивних можливостей, коли тисячишся у пошуках прихованих дефектів (дебагінг) або думаєш про системний промпт, який найкраще працює з локальною LLM. У результаті цього виникає нервове й емоційне виснаження, яке посилюють жорсткі дедлайни та необхідність швидко опановувати нові технологічні стекі. Це породжує явище, яке часто називають сильною візуальною втомою. Те, що ми бачимо на екрані, - зовсім інша річ, ніж те, як щось виглядає для ока: вони світяться і складаються з пікселів. У середовищі IDE, де дрібні синтаксичні деталі отримують особливу фокусну увагу, програмісти кліпають значно рідше. Це порушує зволоження тканин рогівки та запускає небезпечне розширення

«синдрому комп’ютерного зору», а також розлад сухого ока [14]. Утримання погляду на екрані монітора протягом тривалого часу також призводить до спазму акомодації та сприяє розвитку набутої короткозорості.

Щоб підтримувати гостроту зору та уникати перевантаження, бажано дотримуватися міжнародного правила 20-20-20: кожні 20 хвилин відводь погляд від монітора на 20 секунд, фокусуючись на об’єкті, що розташований далі ніж 6 метрів. Дуже простий гігієнічний процес, який дуже ефективний для м’язів акомодації ока. Крім того, рекомендуються спеціальні вправи для зору: щільно заплющувати очі на кілька секунд, часто м’яко кліпати, повільно здійснювати кругові рухи очним яблуком, а також «пальмінг» (закривання заплющених очей теплими долонями, щоб створити повну темряву та глибоке розслаблення). Також важливо правильно налаштувати сам інтерфейс програмного забезпечення: працювати в кодових редакторах із темними темами та шрифтами з високою контрастністю для кращого контрасту; а ще корисно ввімкнути фільтр синього світла, який є майже в кожній релевантній програмі після 19:00–20:00. Це зменшує стрес для сітківки та захищає від будь-яких порушень природних циркадних ритмів організму [14].

Є прямий шлях до вигорання - це накопичення стресу щодня, без емоційної винагороди. Такий тип професійного вигорання змушує вас швидко втрачати мотивацію, ставати байдужим, а продуктивність швидко падає. Психофізіологічне відновлення просто необхідне для довгострокової та стійкої роботи ІТ-фахівця. Згідно з мінімальними критеріями гігієни робочого місця, пропонується розмежовувати робочий і особистий простори, особливо в сценаріях віддаленої роботи. [17] Різка зміна обстановки, часті заняття спортом і прогулянки на свіжому повітрі, повноцінний 8-годинний сон відновлюють нейронні зв’язки; надлишковий кортизол знижується. Регулярні короткі фізичні розминки під час технічних

перерв сприяють інтенсивному кровотоку в мозку, виводять токсичну млявість із таза та хребта, і в сукупності ці складові гарантують досить високу стійкість процесу, підвищену опірність стрес-факторам, а також безперервну ефективність під час виконання складних інженерних завдань.

4.4 Висновок до розділу з безпеки життєдіяльності

Розроблення сучасного програмного забезпечення, зокрема у сфері штучного інтелекту, потребує значних технічних знань, а також дотримання вимог з охорони праці. Крім того, стан здоров’я розробника та надійність апаратного забезпечення є важливими передумовами для досягнення успіху в ІТ-проєкті. Як показано в наукових і нормативних документах, безпечне середовище формується лише в результаті комплексного підходу з урахуванням ергономіки, електробезпеки, протипожежного захисту та зниження психофізіологічного навантаження. Згідно із Законом України «Про охорону праці», забезпечення безпечних умов праці є обов’язковим і має істотне значення для виконання продуктивної інтелектуальної роботи.

Перший рубіж захисту від професійних захворювань - ергономічне облаштування. Динамічно налаштована багатомоніторна система, синхронізований механізм крісел і ергономічні периферійні пристрої призначені для мінімізації статичного напруження в м’язовому корсеті. Одночасно безперервний контроль мікрокліматичних факторів (профілактика повторного вдихання; оптимальна температура та вологість; циркуляція свіжого повітря, щоб уникнути кисневого голодування мозку, синдром сухого ока) запобігає повторному вдиханню; оптимальній температурі та вологості. Якісно поєднані дані щодо освітлення для захисту органу зору від втоми є першочерговим завданням, яке необхідно вирішити.

Це дає можливість «безпечно» залучати розум і зберігати гостроту мислення, не шкодячи організму.

Здається, що існують реальні ризики ураження електричним струмом та займання під час роботи потужного серверного обладнання і мережевих пристроїв. Порушення правил електробезпеки, заземлення будь-якого струмопровідного корпусу та захисні установки для відключення: ПЗВ (пристрої захисного вимикання) захищають життя ІТ-фахівця. Надійні прецизійні системи кондиціювання повітря та вогнегасники з діоксидом вуглецю в серверному приміщенні допомагають запобігти перегріванню і, відповідно, пожежі через підвищення температури компонентів. З точки зору загальних принципів безпеки життєдіяльності інженерно-технічні заходи для запобігання аваріям обчислювального обладнання завжди дешевші й ефективніші, ніж усунення їхніх наслідків.

Інженерні завдання не можна реалізувати програмісту без урахування психофізіологічного стану людей. Потенціал вигорання виникає через високе інтелектуальне навантаження, постійну роботу з абстрактними даними та жорсткі дедлайни. Звісно, необхідно чергувати етапи, коли ми пишемо багато коду, з запланованими перервами для відпочинку очей і розтяжками, щоб оновити нейронну активацію та усунути стрес. Реалізація всіх запропонованих рішень разом із суворим дотриманням вимог державних норм створює абсолютно безпечний, комфортний і надзвичайно ефективний робочий простір. Це дозволяє досягти максимальної реалізації своїх можливостей, зберігаючи при цьому фізичне та психічне благополуччя.

ЗАГАЛЬНІ ВИСНОВКИ ЩОДО КВАЛІФІКАЦІЙНОЇ РОБОТИ

Кваліфікаційну роботу було успішно виконано, що дало змогу розробити автономного, безпечного інтелектуального віртуального помічника, який здатен радикально оптимізувати процеси комунікації в закладі вищої освіти. Розроблений програмний продукт став оптимальним рішенням сучасних викликів цифровізації, коли більше неможливо інформувати студентів традиційними способами, а навантаження як адміністративного, так і викладацького персоналу стає критично високим. Локальна інфраструктура також є проєктом, який унеможливив використання комерційних публічних нейромереж для вирішення питань конфіденційності даних без компромісів, а також усунув фінансову залежність від зовнішніх хмарних провайдерів.

Його було збудовано з використанням технічного рішення на основі архітектури Retrieval-Augmented Generation (RAG). Цей підхід допоміг зменшити галюцинації та надати можливість запобігати їм у модульному компоненті ланцюжкової моделі, що є важливим, адже великі мовні моделі часто генерують правдоподібну, але неправильну інформацію. Використання негативних обмежень у системних промптах дозволяє локальній нейромережі працювати в чітко визначених межах: вона відповідає лише, дотримуючись офіційних методичних вказівок, розкладів і графіків із бази даних департаменту. Алгоритм розроблено так, щоб ефективно запобігати спробам генерувати неперевірені факти — тобто чатбот стає більш легітимним джерелом академічної інформації.

Побудова системи з інженерної перспективи: Система, яку розроблено, працює на архітектурі на основі мікросервісів із використанням інструментів Docker Compose. Рознесення функціональності по ізольованих контейнерах забезпечило базову відмовостійкість і стандартизувало розгортання компонентів на серверному обладнанні. Інша команда створила

сервіс інженерного завантаження знань, щоб автоматизувати обробку документів. Цей компонент у фоновому режимі відстежує задану директорію та обробляє нові файли (PDF, DOCX, TXT). У разі потреби також застосовується гібридний парсинг за допомогою OCR з Tesseract (у двомовному режимі). Щоб гарантувати, що будь-який абзац не втратить структурної цілісності, ми розбиваємо текст на кілька фрагментів по 768 токенів, які перекриваються на 80 токенів для подальшої обробки в векторній базі даних.

Процес семантичного пошуку виконується через мікросервіс TEI, який отримує 1024-вимірні вектори з багатомовної моделі multilingual-e5-large та зберігає їх у векторній базі даних Qdrant. Обчислення косинусної подібності обмежують імовірність отримання нерелевантного контексту навіть тоді, коли користувач формує запит у незвичний спосіб або використовує синоніми. Щоб зберігати історію чатів і облікові записи користувачів, ми також розгорнули реляційну базу даних PostgreSQL. Індекси B-Tree роблять доступ до історії повідомлень швидшим, дозволяючи боту використовувати контекст із попередніх відповідей під час вашої поточної розмови.

У цьому випадку клієнтська частина реалізована як Telegram-бот із використанням асинхронних бібліотек. Це можна зробити за допомогою алгоритму потокової передачі токенів, який показує відповідь користувачу токен за токеном, імітуючи обробку запиту мовною моделлю локально швидко. Для частини бота реалізовано базову систему безпеки: використання регулярних виразів для зіставлення внутрішніх студентських ID; обмеження частоти запитів (throttling); і перевірку безпеки (Safety Check) перед передаванням її основній моделі.

Паралельно розгорталась інша фаза, спрямована на аналіз вимог (як з питань життєзабезпечення, так і охорони праці). На основі чинних нормативно-правових актів було сформульовано низку ергономічних,

електротехнічних і вимог пожежної безпеки щодо організації робочого місця самого розробника та серверних приміщень. Ці рекомендації призначені для захисту персоналу та зменшення будь-якого потенційного ризику пошкодження обчислювальної техніки під час експлуатації.

По суті, на цьому етапі розроблений програмний продукт позиціонується як готова основа для перевірки здійсненності обраного архітектурного рішення. Уся основна функціональність системи працює у взаємодії та може виконувати поставлені завдання в ізольованому режимі. Наступним кроком є зробити її обов’язковою складовою реальної інформаційної інфраструктури закладу вищої освіти, а також тривалим практичним кліматичним випробуванням із реальними користувачами. Статистична інформація дасть змогу виявити приховані технічні недоліки, дослідити продуктивність системи під навантаженням і дивні комбінації дій користувачів під час таких випробувань, збираючи. Після того як ми встановимо та застосуємо зворотний зв’язок, лише тоді можна безперервно вдосконалювати систему: оптимізація розміру чанків, налаштування топології підказок, розширення підтримуваних типів документів, масштабування серверної інфраструктури для збереження надійності.

ПЕРЕЛІК ПОСИЛАНЬ

1. Грабовський В. Системи штучного інтелекту : конспект лекцій. Змістовий модуль 1. «Класичний» підхід до створення систем штучного інтелекту / В. Грабовський. – [Б. м. : б. в., б. р.].
2. ДБН В.2.5-28:2018. Природне і штучне освітлення. – Київ : Мінрегіон України, 2018.
3. ДБН В.2.5-67:2013. Опалення, вентиляція та кондиціонування. – Київ : Мінрегіон України, 2013. – 141 с. – (Державні будівельні норми України).
4. ДСанПіН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. – Київ, 1998.
5. ДСН 3.3.6.042-99. Державні санітарні норми мікроклімату виробничих приміщень. – Київ, 1999.
6. Еволюція RAG [Електронний ресурс] // Unite.ai. – URL: https://www.unite.ai/uk/rag-evolution-a-primer-to-agentic-rag/ (дата звернення: 24.06.2026).
7. Історія штучного інтелекту [Електронний ресурс] // Вікіпедія : вільна енциклопедія. – URL: https://uk.wikipedia.org/wiki/Історія_штучного_інтелекту (дата звернення: 24.06.2026).
8. Історія штучного інтелекту від ідей до сучасних технологій [Електронний ресурс] // KRV Media. – URL: https://krv.media/istoriya-shtuchnogo-intelektu-vid-ideyi-do-suchasnyh-tehnologij/ (дата звернення: 24.06.2026).
9. Махно Є. Галюцинації штучного інтелекту у сфері освіти та науки: причини, наслідки та методи мінімізації / Є. Махно, Є. Руденко, Є. Судніков, М. Тищенко // Повітряна міць України. – 2025. – № 1. – С. 111–126.
10. НАПБ А.01.001-2014. Правила пожежної безпеки в Україні : затв. наказом МВС України від 30.12.2014 р. № 1417. – Київ, 2014.

11. НПАОП 0.00-7.15-18. Вимоги щодо забезпечення безпеки та захисту здоров’я працівників, які виконують роботи з використанням дисплеїв. – Київ, 2018.
12. Правила улаштування електроустановок (ПУЕ) : затв. наказом Міненерговугілля України від 21.07.2017 р. № 476. – Київ, 2017.
13. Публікація про безпеку LLM [Електронний ресурс] // ArXiv. – URL: https://arxiv.org/abs/2406.02622 (дата звернення: 24.06.2026).
14. Риков С. О. Хвороба сухого ока та комп'ютерний зоровий синдром: діагностика і лікування / С. О. Риков // Здоров'я України. – 2023. – № 10 (546). – С. 14–15.
15. Рівні галюцинацій штучного інтелекту та бенчмарки [Електронний ресурс] // SuprMind.ai. – URL: https://suprmind.ai/hub/ai-hallucination-rates-and-benchmarks/ (дата звернення: 24.06.2026).
16. Стаття про використання ШІ [Електронний ресурс] // Національний університет оборони України. – URL: https://sap.nuou.org.ua/article/view/327899 (дата звернення: 24.06.2026).
17. Ткачук К. Н. Основи охорони праці : підручник / К. Н. Ткачук, М. О. Халімовський. – 2-ге вид., допов. та перероб. – Київ : Основа, 2006. – 448 с..
18. Що таке штучний інтелект (історія, види) [Електронний ресурс] // GigaCloud. – URL: https://gigacloud.ua/articles/shho-take-shtuchnyj-intelekt-istoriya-vydy-ta-skladovi/ (дата звернення: 24.06.2026).
19. Коноваленко І. В. Платформа .NET та мова програмування C# 8.0 : навч. посіб. / І. В. Коноваленко, П. О. Марущак. – Тернопіль : В. А. Паляниця, 2020 – 320 с. http://library.megu.edu.ua:8180/jspui/handle/123456789/4115
20. Проектування мікропроцесорних систем керування: навчальний посібник / І.Р. Козбур, П.О. Марущак, В.Р. Медвідь, В.Б. Савків, В.П. Пісьціо. – Тернопіль: Вид-во ТНТУ імені Івана Пулюя, 2022. – 324 с. https://elartu.tntu.edu.ua/handle/lib/39189

21. Капаціла Ю.Б., Савків В.Б. Методичні вказівки до виконання кваліфікаційної роботи бакалавра спеціальності «Автоматизація, комп’ютерно-інтегровані технології та робототехніка». Тернопіль.: Видавництво ТНТУ. 2025. 60 с. https://elartu.tntu.edu.ua/handle/lib/48495
22. Капаціла Ю.Б., Шовкун О.П. Конструкторсько-технологічна практика. Методичні вказівки для здобувачів освітнього ступеня «бакалавр» спеціальності «Автоматизація, комп’ютерно-інтегровані технології та робототехніка». Тернопіль.: Видавництво ТНТУ. 2025. 22 с.

23. Козбур І.Р., Методичні вказівки до виконання лабораторної роботи «Дослідження частотних характеристик неперервних лінійних систем», по курсу «Теорія автоматичного управління», для студентів 3 курсу спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» / Авт.: Козбур І.Р., Козбур Г.В. Марущак П.О., Савків В.Б. – Тернопіль: ТНТУ, ФПТ, каф. АВ, – 2022. – с. 16. https://elartu.tntu.edu.ua/handle/lib/39207
24. Козбур І.Р., «Дослідження часових характеристик неперервних лінійних систем», по курсу «Теорія автоматичного управління», для студентів 3 курсу спеціальності 151 «Автоматизація та комп’ютерно-інтегровані технології» / Авт.: Козбур І.Р., Козбур Г.В. Марущак П.О., Савків В.Б. – Тернопіль: ТНТУ, ФПТ, каф. АВ, – 2022. – 19 с. https://elartu.tntu.edu.ua/handle/lib/39206
25. Автоматизація виробничих процесів. Навчальний посібник для технічних спеціальностей вищих навчальних закладів. / Я.І. Проць, В.Б. Савків, О.К. Шкодзінський, О.Л. Ляшук. Тернопіль: ТНТУ ім. І. Пулюя, 2011. 344 с. https://elartu.tntu.edu.ua/handle/123456789/1551
26. Методичні вказівки по роботі з програмним симулятором "AVR simulator IDE" з курсу "Мікропроцесорні та програмні засоби автоматизації" / укл. : В.Р. Медвідь , В.П. Пісьціо. - Тернопіль : ТНТУ імені Івана Пулюя, 2020. - 21 с.

27. Моделювання нелінійних систем керування у пакеті MATLAB SIMULINK, методичні вказівки до виконання лабораторної роботи по курсу «Комп’ютерні методи дослідження систем автоматичного управління», для студентів 4 курсу спеціальності 6.050201 «Системна інженерія» / укл. : І.Р. Козбур , Г.В. Козбур , Р.І. Михайлишин. – Тернопіль : ТНТУ імені Івана Пулюя, 2019. - 19 с. https://elartu.tntu.edu.ua/handle/lib/28057
28. Моделювання систем керування в пакеті MATLAB SIMULINK, методичні вказівки до виконання лабораторної роботи по курсу «Комп’ютерні методи дослідження систем автоматичного управління», для студентів 4 курсу спеціальності 6.050201 «Системна інженерія» / укл. : І.Р. Козбур , Г.В. Козбур , Р.І. Михайлишин. – Тернопіль : ТНТУ, 2019. - 23 с. https://elartu.tntu.edu.ua/handle/lib/28056
29. Проектування та аналіз електричних схем в програмному середовищі Multisim. Методичні вказівки до самостійної роботи студентів з курсу "Проектування мікропроцесорних систем керування технологічними процесами" / укл. : В.Р. Медвідь , В.П. Пісьціо . - Тернопіль : ТНТУ Імені Івана Пулюя, 2018. - 26 с. https://elartu.tntu.edu.ua/handle/lib/26404
30. Проектування та аналіз електричних схем в програмному середовищі Proteus VSM. Методичні вказівки до самостійної роботи студентів з курсу "Проектування мікропроцесорних систем керування технологічними процесами" / укл. : В.Р. Медвідь , В.П. Пісьціо. - Тернопіль : ТНТУ, 2018. - 26 с. https://elartu.tntu.edu.ua/handle/lib/26397
31. Методичні вказівки до курсового проектування з курсу "Проектування систем автоматизації" / Савків В.Б., Шкодзінський О.К., Пісьціо В.П. Тернопіль : ТНТУ, 2025. 128 с. https://elartu.tntu.edu.ua/handle/lib/48646
32. Лабораторний практикум з проектування та моделювання роботи електропневматичних схем у середовищі програмного пакету «FluidSIM Pneumatics» з курсу «Технічні засоби автоматизації» / укл. : О.К.

Шкодзінський. – Тернопіль : ТНТУ імені Івана Пулюя, 2020. - 32 с. https://elartu.tntu.edu.ua/handle/lib/31418

33. Трембач Р.Б., Медвідь В.Р. Методичні вказівки до виконання курсового проекту з дисципліни “Електроніка і мікросхемотехніка”. – Тернопіль: ТНТУ ім. І. Пулюя, 2024. – 53 с. https://elartu.tntu.edu.ua/handle/lib/44635
34. Трембач Р.Б., Медвідь В.Р. Методичні вказівки до виконання до виконання лабораторних робіт з дисципліни “Електроніка і мікросхемотехніка” Модуль 3. «Імпульсна техніка, вторинні джерела живлення, основи цифрової електроніки» – Тернопіль: ТНТУ, 2024. -26 с.
35. Трембач Р.Б., Шовкун О.П. Методичні вказівки до виконання до виконання лабораторних робіт з дисципліни “Інформаційно – вимірювальні системи” Модуль І. «Вимірювальна техніка» – Тернопіль: ТНТУ., 2024. – 67 с.
36. Аналіз стійкості і якості електромеханічної слідкуючої системи. : Методичні вказівки до виконання курсового проекту «Теорія автоматичного управління», для здобувачів спеціальності 174 «Автоматизація, комп’ютерно-інтегровані технології та робототехніка» освітньо-професійних програм «Комп’ютерно-інтегровані системи автоматики та робототехніки», «Комп’ютеризовані системи управління та прикладне програмування». / укл. І.Р. Козбур, В.П. Пісьціо, В.Б. Савків, П.С. Федорів. – Тернопіль: ТНТУ імені Івана Пулюя, 2025. – 67 с
37. Методичні вказівки для написання розділу «Безпека життєдіяльності, основи охорони праці» в кваліфікаційних роботах здобувачів освітнього рівня ,,бакалавр”. Для студентів всіх форм навчання рівень вищої освіти перший (бакалаврський)/ укл.: О. Я. Гурик , І. Б. Окіпний. – Тернопіль: ТНТУ імені Івана Пулюя, 2021. - 20 с.
38. Навчально-методичний посібник до практичних заняття з дисципліни «Безпека життєдіяльності, основи охорони праці» для студентів освітнього ступеня бакалавр усіх спеціальностей та форм навчання /

Укладачі : О. Я. Гурик, І. Б. Окіпний, В. С. Сенчишин, С. Ю. Мариненко, О. І. Король. Тернопіль : ТНТУ імені Івана Пулюя, 2025. 123 с. https://elartu.tntu.edu.ua/handle/lib/48496

39. Безпека в надзвичайних ситуаціях. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання / укл.: Стручок В. С. Тернопіль: ФОП Паляниця В. А., 2022. 156 с.
40. Микитишин А. Г., Митник М. М., Стухляк П. Д., Пасічник В. В. Комп'ютерні мережі. Книга 1 [навчальний посібник]. Львів : «Магнолія 2006», 2013. 256 с.
41. Микитишин А. Г., Митник М. М., Стухляк П. Д., Пасічник В. В. Комп'ютерні мережі. Книга 2. [навчальний посібник]. Львів : "Магнолія 2006", 2014. 312 с.
42. Микитишин А. Г., Митник М. М., Стухляк П. Д. Телекомунікаційні системи та мережі. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2017. 384 с.
43. Буров Є., Митник М. Комп'ютерні мережі. (у 2-х томах). Львів, Магнолія, 2018.
44. Комплексна безпека інформаційних мережевих систем. Навчальний посібник для студентів спеціальності 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» / А. Г. Микитиший, М. М. Митник, О. С. Голотенко, В. В. Карташов. – Тернопіль: ФОП Паляниця В.А., 2023. – 324 с. https://elartu.tntu.edu.ua/handle/lib/42625
45. Конспект лекцій з дисципліни «Комп'ютерні системи штучного інтелекту» для студентів спеціальності G7 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» денної та заочної форми здобуття освіти / уклад. І. С. Дідич. // ТНТУ. – 2025. – С. 92. https://elartu.tntu.edu.ua/handle/lib/50293
46. Пилипець М. І. Правила заповнення основних форм технологічних документів : навч.-метод. посіб. / Уклад. Пилипець М. І., Ткаченко І. Г.,

Левкович М. Г., Васильків В. В., Радик Д. Л. Тернопіль : ТДТУ, 2009. 108 с. https://elartu.tntu.edu.ua/handle/lib/42995

47. I. Strutynska, H. Kozbur, L. Dmytrotsa, O. Sorokivska, L. Melnyk and R. Grytseliak, "Regarding to the Concept of Small and Medium-Sized Enterprises Digitalization in Ukraine: Problems and Solutions," 2021 11th International Conference on Advanced Computer Information Technologies (ACIT), Deggendorf, Germany, 2021, pp. 276-279, doi: 10.1109/ACIT52158.2021.9548382.
48. Паламар М. І., Стрембіцький М. О., Паламар А. М. Проектування комп'ютеризованих вимірювальних систем і комплексів : навч. посіб. Тернопіль : ТНТУ, 2019. 150 с.
49. Валяшек В.Б. Навчальний посібник з курсу: " Оптимізаційні методи та моделі " для спеціальностей Облік і аудит, Фінанси і кредит, Маркетинг, Економічна кібернетика / Кривень В.А., Валяшек В.Б., Цимбалюк Л.І., Козбур Г.В. – Тернопіль : видавництво ТНТУ, 2015. – 83с.
50. Савків В.Б., Капаціла Ю.Б., Михайлишин Р.І. Методичні вказівки до виконання кваліфікаційної роботи бакалавра спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології». Тернопіль.: Видавництво ТНТУ. 2021. 50 с. https://elartu.tntu.edu.ua/handle/lib/35172
51. CocoIndex: документація [Електронний ресурс] // CocoIndex. – URL: https://cocoindex.io/ (дата звернення: 24.06.2026).
52. Docker and Container Isolation [Електронний ресурс] // Medium. – URL: https://sigridjin.medium.com/docker-and-container-isolation-85e235aa5854 (дата звернення: 24.06.2026).
53. Introduction to Tesseract OCR [Електронний ресурс] // Medium. – URL: https://medium.com/zeals-tech-blog/introduction-to-tesseract-ocr-84d3eff6f9df (дата звернення: 24.06.2026).
54. Natural language processing: History, evolution, application, and future work / P. Johri [et al.] // Proceedings of 3rd International Conference on Computing

Informatics and Networks: ICCIN 2020. – Singapore : Springer, 2021. – P. 365–375.

55. Multilingual E5 Large Instruct Operations [Електронний ресурс] // GlobalNodes. – URL: https://globalnodes.tech/blog/multilingual-e5-large-instruct-operations-for-llm-embedding/ (дата звернення: 24.06.2026).
56. NLP Embeddings [Електронний ресурс] // Dkharazi GitHub Notes. – URL: https://dkharazi.github.io/notes/ml/nlp/embedding (дата звернення: 24.06.2026).
57. Open Source vs Proprietary AI/ML [Електронний ресурс] // De Novo. – URL: https://denovo.ua/blog/opensource-vs-prop-ai-ml (дата звернення: 24.06.2026).
58. Prompt Engineering [Електронний ресурс] // IBM. – URL: https://www.ibm.com/think/topics/prompt-engineering (дата звернення: 24.06.2026).
59. Python Telegram Bot: офіційна сторінка бібліотеки [Електронний ресурс]. – URL: https://python-telegram-bot.org/ (дата звернення: 24.06.2026).
60. Q1 2024: Top 5 Communication Apps by Downloads [Електронний ресурс] // Sensor Tower. – URL: https://sensortower.com/blog/2024-q1-unified-top-5-communication%20apps-units-ua-6070aae1241bc16eb81f5bab (дата звернення: 24.06.2026).
61. RAG Architecture Technical Guide [Електронний ресурс] // Veniplatform. – URL: https://www.veniplatform.com/uk/blog/rag-mimarisi-retrieval-augmented-generation-teknik-rehberi (дата звернення: 24.06.2026).
62. RAG Basics [Електронний ресурс] // Medium. – URL: https://medium.com/@dinabavli/rag-basics-basic-implementation-of-retrieval-augmented-generation-rag-e80e0791159d (дата звернення: 24.06.2026).
63. Text Embeddings Inference [Електронний ресурс] // Hugging Face. – URL: https://huggingface.co/docs/text-embeddings-inference/index (дата звернення: 24.06.2026).