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

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Дослідження та розробка AI-асистента на основі моделі Mistral для середовища університету

Виконав: студент VI курсу, групи СНм-61
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

_____ Попович В.В.
(підпис) (прізвище та ініціали)

Керівник _____ Готович А.В.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Никитюк В.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Ясній О.П.
(підпис) (прізвище та ініціали)

Тернопіль
2025

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

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
« 17 » листопада 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)
за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)
Студенту Поповичу Валерію Валерійовичу
(прізвище, ім'я, по батькові)
1. Тема роботи Дослідження та розробка AI-асистента на основі моделі Mistral для середовища університету

Керівник роботи Готович Володимир Анатолійович, к.т.н., доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «_27_» _листопада ___ 2025 року № _*4/7-1042*_
2. Термін подання студентом завершеної роботи 22 грудня 2025 р.
3. Вихідні дані до роботи Наукові праці в галузі штучного інтелекту та великих мовних моделей, документація бібліотек NLP, а також масив PDF-документів з репозиторію ELARTU для наповнення та тестування бази знань системи.
4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз предметної області та методів побудови інтелектуальних пошукових систем. 2. Проєктування та розробка архітектури AI-асистента. 3. Реалізація та експериментальне дослідження системи. 4. Охорона праці та безпека в надзвичайних ситуаціях. Висновки. Перелік джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1 Титульна сторінка. 2 Мета та задачі. 3 Об'єкт та предмет дослідження. 4 Наукова новизна та Практичне значення. 5 Аналіз проблеми «галюцинацій» у LLM. 6 Концепція архітектури RAG 7 Обґрунтування технологічного стеку. 8 Структурна схема системи. 9 Методика обробки даних (Chunking). 10 Алгоритм семантичного пошуку. 11 Математична модель семантичного пошуку. 12 Програмна реалізація алгоритму сегментації. 13 Реалізація логіки гібридного ранжування (RAG). 14 Експериментальний набір даних. 15 Програмна реалізація та інтерфейс. 16 Інтерфейс користувача та демонстрація роботи. 17 Порівняльний аналіз якості (A/B тестування). 17 Висновки. 18 Завершальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Гурик О.Я., канд. техн. наук, доц.		
Безпека в надзвичайних ситуаціях	Теслюк В.М., проректор з адміністративно-господарської роботи та будівництва		

7. Дата видачі завдання 17 листопада 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	17.11.2025	*Виконано*
2.	Аналіз науково-технічних публікацій та збір даних по	17.11.2025-24.11.2025	*Виконано*
	темі кваліфікаційної роботи		
3.	Виконання дослідження згідно мети кваліфікаційної	25.11.2025-08.12.2025	*Виконано*
	роботи		
4.	Оформлення розділу «Аналіз предметної області та	09.12.2025-11.12.2025	*Виконано*
	методів побудови інтелектуальних пошукових		
	систем»		
5.	Оформлення розділу «Проєктування та розробка	09.12.2025-11.12.2025	*Виконано*
	архітектури AI-асистента»		
6.	Оформлення розділу «Реалізація та експериментальне	09.12.2025-11.12.2025	*Виконано*
	дослідження системи»		
7.	Виконання завдання до підрозділу «Охорона праці»	12.12.2025-15.12.2025	*Виконано*
8.	Виконання завдання до підрозділу «Безпека в	12.12.2025-15.12.2025	*Виконано*
	надзвичайних ситуаціях»		
9.	Оформлення кваліфікаційної роботи	15.12.2025	*Виконано*
10.	Нормоконтроль	17.12.2025	*Виконано*
11.	Перевірка на плагіат	18.12.2025	*Виконано*
12.	Попередній захист кваліфікаційної роботи	19.12.2025	*Виконано*
13.	Захист кваліфікаційної роботи	22.12.2025	*Виконано*

Студент _____ Попович В.В,
(підпис) (прізвище та ініціали)

Керівник роботи _____ Готович В.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дослідження та розробка AI-асистента на основі моделі Mistral для середовища університету // Кваліфікаційна робота освітнього ступеня «Магістр» // Попович Валерій Валерійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНм-61 // Тернопіль, 2025 // С. 85, рис. – 8, табл. – 8, додат. – 3, бібліогр. – 69.

Ключові слова: штучний інтелект, велика мовна модель, Mistral, RAG, обробка природної мови, семантичний пошук, векторна база даних, AI-асистент.

Кваліфікаційна робота присвячена розробці автономного інтелектуального асистента для університетського середовища, який використовує технологію Retrieval-Augmented Generation (RAG) та локальну мовну модель Mistral для надання точних відповідей на основі внутрішньої бази знань.

В першому розділі кваліфікаційної роботи описані сучасні підходи до організації інформаційного пошуку та принципи функціонування великих мовних моделей. Висвітлено проблему виникнення «галюцинацій» у генеративних системах та методи їх мінімізації. Розглянуто архітектуру RAG як оптимальний метод поєднання генеративних можливостей нейромереж із точністю пошукових систем. Проаналізовано переваги локального розгортання моделей для забезпечення конфіденційності даних.

В другому розділі кваліфікаційної роботи обґрунтовано вибір технологічного стеку, що включає модель Mistral 7B, платформу Ollama та векторну базу даних ChromaDB. Досліджено методи попередньої обробки неструктурованих PDF-документів та розроблено алгоритм сегментації тексту методом ковзного вікна. Подано структурну схему системи та математичний опис алгоритму гібридного семантичного пошуку.

В третьому розділі кваліфікаційної роботи описано програмну реалізацію прототипу системи мовою Python та процес формування унікального експериментального корпусу даних на основі магістерських робіт.

Проаналізовано ресурсомісткість системи та підтверджено ефективність методу квантування. Проведено порівняльне тестування якості відповідей, яке продемонструвало суттєве підвищення фактологічної точності при використанні розробленої архітектури.

Об’єкт дослідження: процеси інтелектуального пошуку та обробки природної мови для спеціалізованих базах знань.

Предмет дослідження: методи та засоби побудови діалогових систем на основі архітектури Retrieval-Augmented Generation з використанням локальних обчислювальних ресурсів.

ANNOTATION

Research and Development of an AI Assistant Based on the Mistral Model for the University Environment // The educational level "Master" qualification work // Popovych Valerii // Ternopil Ivan Pulyuy National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, SNm-61 group // Ternopil, 2025 // P. 85, fig. – 8, tables – 8, annexes – 3, ref. – 69.

Key words: artificial intelligence, large language model, Mistral, RAG, natural language processing, semantic search, vector database, AI assistant.

The Master's thesis is dedicated to the development of an autonomous intelligent assistant for a university environment, which utilizes Retrieval-Augmented Generation (RAG) technology and the local Mistral language model to provide accurate responses based on an internal knowledge base.

The first chapter of the thesis describes modern approaches to organizing information retrieval and the operating principles of large language models. The problem of "hallucinations" in generative systems and methods for their minimization are highlighted. The RAG architecture is considered as an optimal method for combining the generative capabilities of neural networks with the precision of search engines. The advantages of local model deployment for ensuring data privacy are analyzed.

In the second chapter, the choice of the technology stack, including the Mistral 7B model, the Ollama platform, and the ChromaDB vector database, is substantiated. Methods for preprocessing unstructured PDF documents are investigated, and a text segmentation algorithm using the sliding window method is developed. The structural scheme of the system and a mathematical description of the hybrid semantic search algorithm are presented.

The third chapter describes the software implementation of the system prototype in Python and the process of forming a unique experimental data corpus based on

Master's theses. The resource intensity of the system is analyzed, and the effectiveness of the quantization method is confirmed. Comparative testing of response quality was conducted, demonstrating a significant increase in factual accuracy when using the developed architecture.

Object of research: processes of intelligent search and natural language processing in specialized knowledge bases.

Subject of research: methods and tools for building dialogue systems based on the Retrieval-Augmented Generation architecture using local computing resources.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – База даних.

ВНЗ – Вищий навчальний заклад.

ОС – Операційна система.

ПЗ – Програмне забезпечення.

ПК – Персональний комп’ютер.

ШІ – Штучний інтелект.

AI (англ. Artificial Intelligence) – штучний інтелект.

ANN (англ. Approximate Nearest Neighbors) – наближений пошук найближчих сусідів.

API (англ. Application Programming Interface) – прикладний програмний інтерфейс.

CLI (англ. Command Line Interface) – інтерфейс командного рядка.

CPU (англ. Central Processing Unit) – центральний процесор.

GPU (англ. Graphics Processing Unit) – графічний процесор.

HNSW (англ. Hierarchical Navigable Small World) – ієрархічний навігаційний «дрібний світ» (алгоритм пошуку).

HTML (англ. HyperText Markup Language) – мова розмітки гіпертексту.

HTTP (англ. HyperText Transfer Protocol) – протокол передачі гіпертексту.

JSON (англ. JavaScript Object Notation) – текстовий формат обміну даними.

LLM (англ. Large Language Model) – велика мовна модель.

NLP (англ. Natural Language Processing) – обробка природної мови.

PDF (англ. Portable Document Format) – формат переносних документів.

RAG (англ. Retrieval-Augmented Generation) – генерація, доповнена пошуком.

RAM (англ. Random Access Memory) – оперативна пам'ять.

REST (англ. Representational State Transfer) – передача репрезентативного стану (архітектурний стиль).

SQL (англ. Structured Query Language) – мова структурованих запитів.

VRAM (англ. Video Random Access Memory) – відеопам'ять.

ЗМІСТ

ВСТУП

Актуальність теми. В умовах стрімкої цифровізації освітнього процесу університети накопичують значні обсяги неструктурованої інформації: від нормативно-правових актів та навчальних планів до кваліфікаційних робіт здобувачів освіти. Традиційні засоби пошуку в таких масивах даних, що базуються на співпадінні ключових слів, втрачають свою ефективність, оскільки не враховують семантичний контекст запиту та морфологічну варіативність природної мови.

Комерційні продукти на основі великих мовних моделей (LLM), такі як ChatGPT, демонструють високу ефективність в обробці текстів, проте їх використання в корпоративному середовищі університету обмежене політиками конфіденційності, вартістю доступу до API та залежністю від зовнішніх серверів. На рівні локального розгортання відкритих моделей (Open Source LLM) доступно менше готових рішень, адаптованих під специфіку української мови та академічної термінології. Тому дослідження та розробка автономної інформаційної системи на основі технології Retrieval-Augmented Generation (RAG) та моделі Mistral є актуальним напрямком сучасних наукових досліджень.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи ступеня «Магістр» є підвищення ефективності інформаційного пошуку та автоматизації консультаційної підтримки в університетському середовищі шляхом розробки AI-асистента на основі локальної моделі Mistral.

Для досягнення поставленої мети потрібно виконати ряд завдань, зокрема:

– Проаналізувати стан досліджень в області обробки природної мови (NLP) та архітектур великих мовних моделей.

– Дослідити існуючі на даний час методи векторного представлення текстів та організації семантичного пошуку.

– Проаналізувати методи мінімізації «галюцинацій» генеративних моделей за допомогою підходу RAG.

– Виконати порівняння існуючих відкритих LLM для локального використання та обґрунтувати вибір моделі Mistral.

– Розробити методику попередньої обробки та сегментації академічних текстів у форматі PDF.

– Розробити програмний прототип AI-асистента з використанням векторної бази даних та гібридного механізму пошуку.

– Провести експериментальне дослідження продуктивності системи та релевантності відповідей.

Об’єкт дослідження: процеси інтелектуального пошуку та генеративної обробки природної мови для спеціалізованих баз знань.

Предмет дослідження: методи та засоби побудови діалогових систем на основі архітектури Retrieval-Augmented Generation з використанням локальних обчислювальних ресурсів.

Наукова новизна одержаних результатів кваліфікаційної роботи полягає у тому, що:

– отримав подальший розвиток метод семантичної сегментації україномовних академічних текстів, який, на відміну від стандартних підходів, враховує логічну структуру наукових робіт для збереження контексту;

– вдосконалено підхід до побудови локальних RAG-систем шляхом впровадження гібридного алгоритму ранжування джерел, що дозволяє динамічно балансувати між загальними знаннями про університет та специфічними даними студентських робіт.

Практичне значення одержаних результатів. Виконано макетування та прототипування AI-асистента для середовища ТНТУ, який здатний функціонувати на персональних комп’ютерах без передачі конфіденційних даних третім сторонам. Створено інструментарій для автоматизованого збору та векторизації магістерських робіт з репозиторію університету.

Апробація результатів магістерської роботи. Основні результати проведених досліджень обговорювались на VI міжнародній студентській науково-технічної конференції «Теорія модернізації в контексті сучасної світової науки» міжнародного центру наукових досліджень (м. Івано-Франківськ, 19 грудня 2025 р.) та XIII науково-технічній конференції

«Інформаційні моделі, системи та технології» Тернопільського національного технічного університету імені Івана Пулюя (м. Тернопіль, 18 грудня 2025 р.).

Публікації. Основні результати кваліфікаційної роботи опубліковано у двох працях конференції (додаток Б).

Структура й обсяг кваліфікаційної роботи. Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку літератури з 69 найменувань та 3 додатків. Загальний обсяг кваліфікаційної роботи складає 85 сторінок, з них 68 сторінок основного тексту, який містить 8 рисунків та 8 таблиць.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ ПОБУДОВИ ІНТЕЛЕКТУАЛЬНИХ ПОШУКОВИХ СИСТЕМ

1.1 Аналіз існуючих підходів до організації інформаційного пошуку

Сучасний етап розвитку інформаційних систем в освітніх закладах характеризується стрімким зростанням обсягів накопичених даних. Університетські репозиторії, електронні бібліотеки та системи управління навчанням (LMS) містять тисячі документів: від адміністративних наказів та навчальних планів до кваліфікаційних робіт студентів та наукових статей викладачів. За оцінками експертів [1], понад 80% цієї інформації зберігається у неструктурованому вигляді (текстові файли форматів PDF, DOCX, TXT), що створює значні труднощі для ефективного пошуку та отримання знань.

Проблема інформаційного пошуку (Information Retrieval – IR) полягає у задоволенні інформаційної потреби користувача шляхом видачі релевантних документів із наявної колекції. Історично склалися два принципово різні підходи до вирішення цієї задачі: лексичний (пошук за ключовими словами) та семантичний (пошук за змістом).

1.1.1 Обмеження та недоліки традиційних систем пошуку за ключовими словами

Традиційні пошукові системи, які десятиліттями домінували в галузі, базуються на лексичному співпадінні термінів запиту та термінів документу. В основі таких систем лежить структура даних, що називається інвертованим індексом (Inverted Index).

Інвертований індекс являє собою словник унікальних слів (термінів), де для кожного слова зберігається список документів, у яких воно зустрічається, а також позиція слова в документі [2].

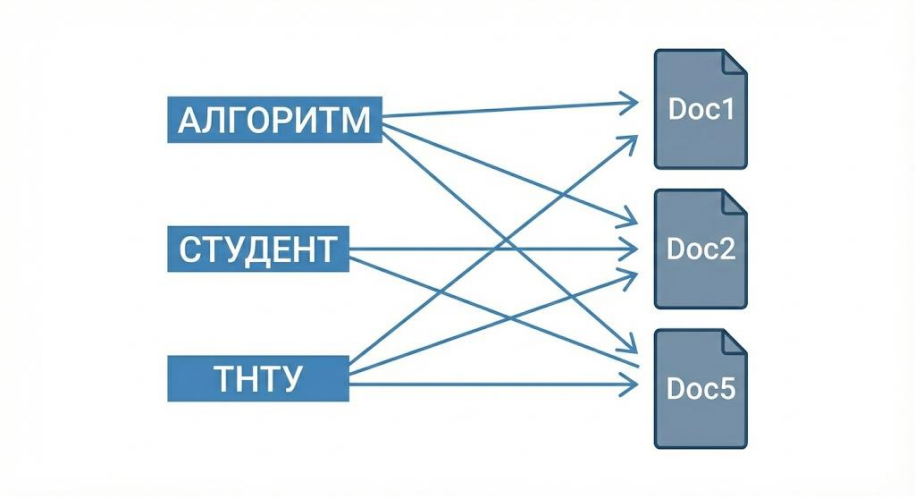

Рисунок 1.1 – Схема інвертованого індексу

Для оцінки важливості документа відносно запиту в таких системах використовуються статистичні міри. Найбільш поширеною є метрика TF-IDF (Term Frequency – Inverse Document Frequency), яка дозволяє оцінити вагу слова в контексті конкретного документа та всієї колекції [3].

Вага терміну t у документі *d* розраховується за формулою (1.1):

$$W(t,d) = TF(t,d) \cdot IDF(t), \tag{1.1}$$

де TF(*t*,*d*) - частота входження терміну в документ (Term Frequency), яка показує, наскільки часто слово зустрічається в тексті;

IDF(*t*) – обернена частота документа (Inverse Document Frequency), яка зменшує вагу загальновживаних слів (наприклад, прийменників або сполучників). IDF*IDF* обчислюється як логарифм в

IDF обчислюється як логарифм відношення загальної кількості документів *N* до кількості документів, що містять термін *t:*

$$IDF(t) = \log \frac{N}{df(t)}. \tag{1.2}$$

Більш досконалі системи, такі як ElasticSearch або Apache Solr, використовують імовірнісну модель BM25 (Best Matching 25), яка є

модифікацією TF-IDF і враховує довжину документа та насичення частоти термінів [4].

Незважаючи на високу швидкість роботи та простоту реалізації, традиційні підходи мають ряд фундаментальних недоліків, які стають критичними при побудові інтелектуальних асистентів для університету. Основні проблеми наведено в таблиці 1.1.

Таблиця 1.1 – Недоліки лексичного підходу до пошуку

Проблема	**Опис проблеми**	**Приклад з предметної області**
Синонімія	Різні слова можуть мати однакове значення. Система не знайде документ, якщо в запиті використано синонім, якого немає в тексті.	Запит: *"Правила вступу"*. Документ: *"Умови прийому на навчання"*. Результат**:** Документ не знайдено.
Полісемія	Одне слово може мати різні значення залежно від контексту.	Запит: *"Потік"*. Значення 1: *"Потік студентів"*. Значення 2: *"Потік виконання (Thread)"*.
Морфологічна варіативність	Для флективних мов (як українська) зміна закінчень слова робить його "іншим" для простого пошуку без лематизації.	Запит: *"кафедрою"*. Документ: *"кафедри"*. Без стемінгу (stemming) збіг не фіксується.
Ігнорування контексту	Пошук розглядає запит як набір окремих слів ("мішок слів"), ігноруючи їх порядок та логічний зв'язок.	Запит: *"Хто є ректором університету?"*. Система шукає будь-які документи зі словами *"хто"*, *"ректор"*, *"унів ерситет"*.

Особливо гостро ця проблема постає при роботі зі студентськими кваліфікаційними роботами. Теми досліджень часто формулюються складною науковою мовою, і студент, який шукає схожі роботи для огляду літератури, може не знати точної термінології, використаної попередниками. Лексичний розрив (lexical gap) між мовою запиту користувача та мовою документів призводить до низької повноти пошуку (Recall).

1.1.2 Переваги семантичного підходу до обробки неструктурованих даних

Для подолання обмежень ключових слів в останні роки активно розвивається напрямок семантичного пошуку (Semantic Search) та векторного представлення текстів. Цей підхід базується на гіпотезі дистрибутивної семантики: слова, що зустрічаються в схожих контекстах, мають схожі значення [5]. На відміну від простого лексичного співставлення, семантичний аналіз дозволяє системі «розуміти» інтент (намір) користувача, виявляючи приховані логічні зв'язки між термінами, навіть якщо вони не співпадають дослівно. Такий прорив став можливим завдяки використанню архітектур глибокого навчання, які здатні враховувати порядок слів та їх взаємовплив у межах речення.

Ключовою концепцією тут є векторні представлення (embeddings). Ембедінг – це перетворення тексту (слова, речення або цілого параграфа) у вектор дійсних чисел фіксованої розмірності n. У такому багатовимірному просторі кожна одиниця тексту отримує унікальні координати, а геометрична відстань між векторами виступає математичною мірою їхньої семантичної близькості.

$$\vec{v} = [x_1, x_2, \dots, x_n], \quad x_i \in R. \tag{1.3}$$

У такому багатовимірному просторі семантично схожі тексти розташовуються геометрично близько один до одного.

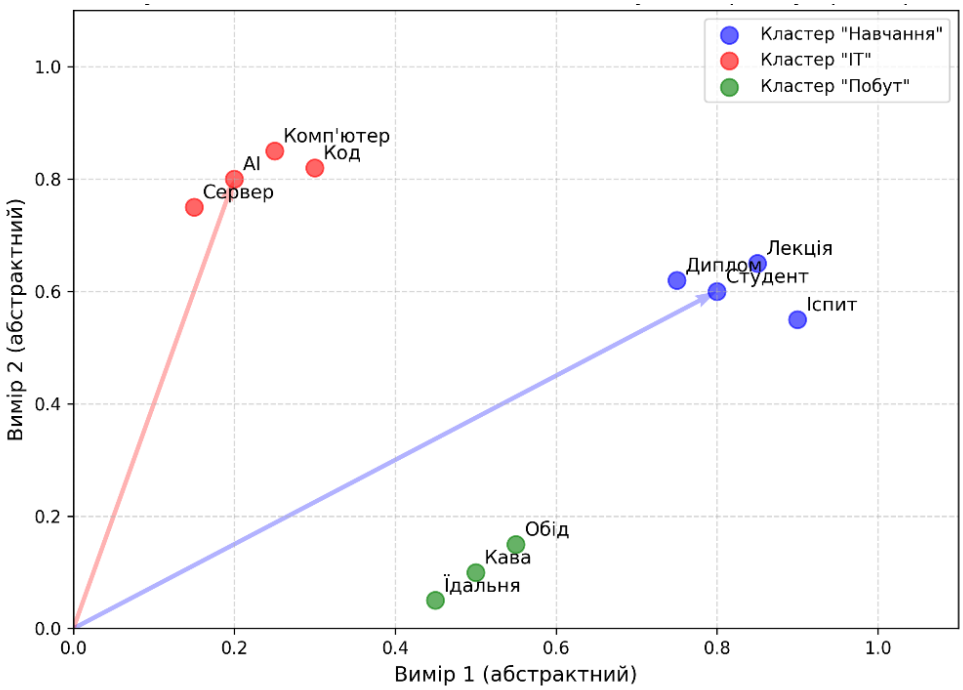

Рисунок 1.2 – Візуалізація семантичного векторного простору

Для визначення ступеня схожості (релевантності) між вектором запиту $\vec{Q}$ та вектором документа $\vec{D}$ найчастіше використовується косинусна подібність (Cosine Similarity). Вона вимірює косинус кута між двома векторами і розраховується за формулою (1.3):

$$\sin(\vec{Q},\vec{D}) = \cos(\theta) = \frac{\vec{Q}\cdot\vec{D}}{\|\vec{Q}\|\cdot\|\vec{D}\|} = \frac{\sum_{i=1}^{n} Q_i D_i}{\sqrt{\sum_{i=1}^{n} Q_i^2}\cdot\sqrt{\sum_{i=1}^{n} D_i^2}}. \quad (1.4)$$

Переваги семантичного підходу для університетського AI-асистента:

1. Розуміння природної мови: Користувач може формулювати запит у розмовній формі, наприклад: *"Де мені взяти довідку про навчання?"*. Векторна модель (наприклад, на базі трансформерів BERT або MiniLM) співставить цей запит із вектором документа, що містить текст *"Видача довідок здійснюється у деканаті"*, незважаючи на відсутність спільних слів.

2. Мультимовність: Сучасні моделі ембедінгів (такі як paraphrase-multilingual-MiniLM, що використовується в даній роботі) здатні відображати тексти різними мовами в єдиний векторний простір. Це дозволяє шукати англомовні джерела, роблячи запит українською мовою, і навпаки [6].

3. Стійкість до помилок: Опечатки або граматичні помилки в запиті менше впливають на результат, оскільки векторне представлення слова з помилкою часто знаходиться близько до правильного слова.

Проте семантичний пошук вимагає значно більших обчислювальних ресурсів для етапу індексації (перетворення всіх документів у вектори) та етапу пошуку (порівняння вектора запиту з мільйонами векторів у базі). Для вирішення цієї проблеми використовуються спеціалізовані векторні бази даних (Vector Databases) та алгоритми наближеного пошуку найближчих сусідів (ANN – Approximate Nearest Neighbors), такі як HNSW (Hierarchical Navigable Small World), що буде детальніше розглянуто у другому розділі.

Таким чином, для створення ефективного AI-асистента доцільно відмовитися від чистого пошуку за ключовими словами на користь гібридних або повністю векторних підходів, що дозволить системі "розуміти" зміст студентських робіт та запитань користувачів.

1.2 Огляд сучасних технологій обробки природної мови (NLP)

Обробка природної мови (Natural Language Processing – NLP) є однією з найбільш динамічних галузей штучного інтелекту, що спрямована на забезпечення взаємодії між людиною та комп'ютером за допомогою природної мови. Еволюція NLP пройшла шлях від систем, заснованих на жорстких лінгвістичних правилах (Rule-based systems), до статистичних методів і, зрештою, до методів глибокого навчання (Deep Learning).

Довгий час стандартом у галузі були рекурентні нейронні мережі (RNN) та мережі з довгою короткостроковою пам'яттю (LSTM). Головним їх недоліком була послідовна обробка інформації: щоб зрозуміти останнє слово в реченні, мережа мусила поетапно обробити всі попередні. Це унеможливлювало ефективне розпаралелювання обчислень на сучасних графічних процесорах (GPU) та призводило до проблеми "забування" контексту в довгих текстах.

Парадигма змінилася у 2017 році з публікацією роботи "Attention Is All You Need" [1], де було запропоновано архітектуру Transformer. Ця архітектура

відмовилася від рекурентності на користь механізму уваги, що дозволило моделям обробляти цілі масиви тексту одночасно.

1.2.1 Великі мовні моделі (LLM): архітектура та принципи функціонування

Великі мовні моделі (LLM) – це ймовірнісні моделі, навчені на надвеликих корпусах текстових даних (сотні гігабайт або терабайти тексту), що дозволяє їм генерувати текст, який важко відрізнити від написаного людиною.

Важливо розуміти, що нейронні мережі не працюють зі словами у звичному розумінні. Першим етапом обробки є токенізація. Текст розбивається на елементарні одиниці - токени. Сучасні моделі використовують алгоритми підслівникової токенізації, такі як BPE (Byte-Pair Encoding) або SentencePiece. Це дозволяє ефективно працювати з рідкісними словами та різними мовами.

Наприклад, слово "університетський" може бути розбите на токени: *["універ", "ситет", "ський"]*. Кожен токен перетворюється на числовий вектор (embedding).

Серцем сучасних LLM є механізм Self-Attention. Його мета – визначити залежність кожного токена в послідовності від інших токенів.

Розглянемо приклад: "Студент не зміг завантажити файл, бо він був занадто великий".

Для людини очевидно, що займенник "він" стосується слова "файл". Для комп'ютера це не очевидно. Механізм уваги обчислює "вагу зв'язку" між токеном "він" та всіма попередніми словами. У даному контексті зв'язок з токеном "файл" матиме найбільшу вагу, а з токеном "студент" – меншу.

Математично це реалізується через три матриці, які навчаються:

1. Query (*Q*) – запит (те, що ми шукаємо).
2. Key (*K*) – ключ (те, що визначає зміст токена).
3. Value (*V*) – значення (інформація, яку ми витягуємо).

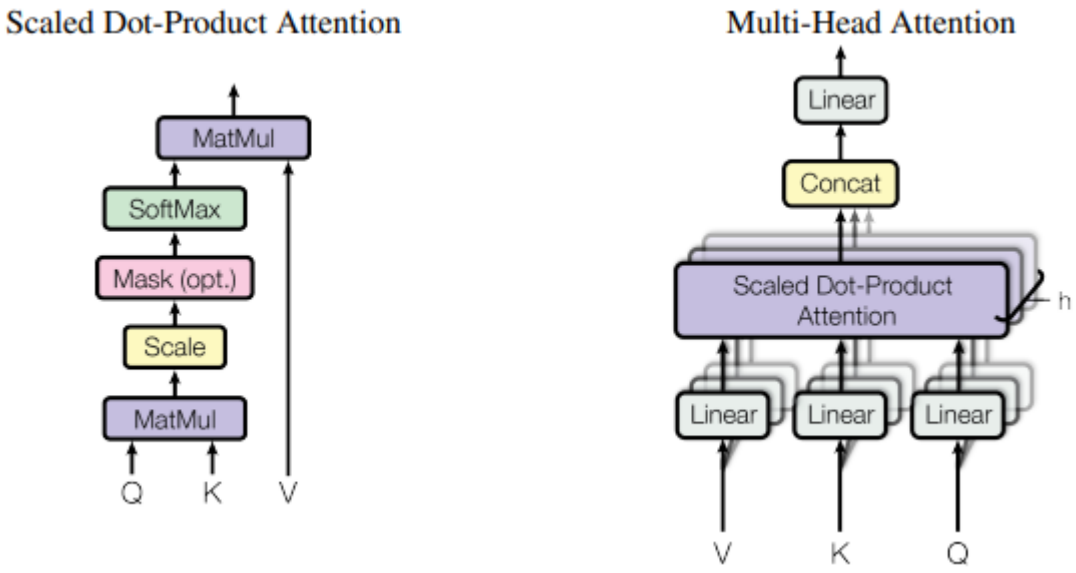

Рисунок 1.3 – Принципова схема блоку Multi-Head Attention

Для розробки університетського асистента обрано модель Mistral 7B [7]. Це модель з відкритою вагою (Open Weights), яка демонструє продуктивність, що перевищує значно більші моделі (наприклад, Llama 2 13B), при цьому залишаючись досить "легкою" для запуску на споживчому обладнанні (6-8 GB VRAM) [7].

Mistral впроваджує дві критично важливі інновації, які роблять її ідеальною для нашої задачі:

1. Sliding Window Attention (SWA) – Ковзне вікно уваги.]

У стандартних трансформерах складність обчислень зростає квадратично від довжини тексту ($O(n^2)$). Це означає, що при збільшенні довжини контексту вдвічі, час обробки зростає в чотири рази, що швидко вичерпує пам'ять GPU.

Mistral використовує механізм ковзного вікна: кожен токен "дивиться" лише на фіксовану кількість попередніх токенів (вікно W). Проте, завдяки багатошаровій структурі мережі, інформація може поширюватися далеко за межі вікна.

Перевага для нашого проєкту*:* Це дозволяє ефективно обробляти довгі фрагменти студентських робіт без перевантаження відеопам'яті ноутбука.

2. Grouped-Query Attention (GQA) – Увага з групуванням запитів.

Ця техніка прискорює процес генерації тексту (інференс) шляхом зменшення кількості параметрів, що зберігаються в кеші (KV-cache).

Перевага для нашого проєкту: Значне прискорення відповіді асистента (менша затримка, або latency) та зниження вимог до оперативної пам'яті.

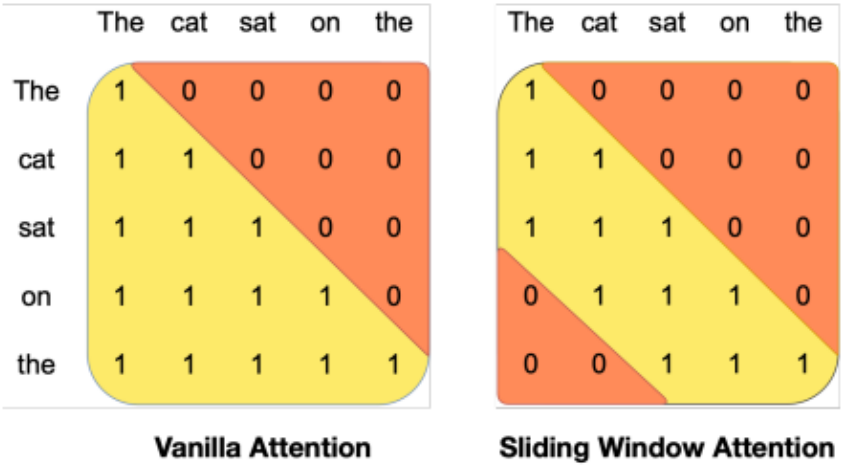

Рисунок 1.4 – Порівняння механізмів Vanilla Attention та Sliding Window Attention.

1.2.2 Проблема «галюцинацій» та методи їх мінімізації

Попри вражаючі можливості, LLM не є базами знань у строгому сенсі. Це ймовірнісні генератори тексту. Однією з головних проблем їх впровадження в критично важливих сферах (освіта, медицина, юриспруденція) є феномен "галюцинацій".

Галюцинація в контексті NLP – це впевнена генерація моделлю інформації, яка є фактично невірною або не спирається на вхідні дані [6].

Виділяють два основні типи галюцинацій:

1. Внутрішні галюцинації (Intrinsic): Модель суперечить сама собі або вхідному контексту. Наприклад, якщо у наданому тексті сказано "Наказ №5 від 2023 року", а модель генерує "Згідно з наказом №5 від 2020 року".

2. Зовнішні галюцинації (Extrinsic): Модель генерує твердження, яких не було у джерелі, але які неможливо перевірити на основі контексту. Наприклад, придумування неіснуючих цитат або авторів.

Природа цього явища полягає у принципі Maximum Likelihood Estimation (MLE). Модель завжди намагається обрати найбільш ймовірне наступне слово.

Якщо модель не знає точної відповіді, вона обирає найбільш "правдоподібну" з лінгвістичної точки зору, що часто призводить до фактичних помилок.

Для мінімізації галюцинацій у нашій системі застосовано трирівневий захист:

1. Системний промптинг (System Prompting):

Ми задаємо "рольову модель" для AI. Інструкція *«Ти – чесний асистент. Якщо інформації немає в контексті, відповідай "Я не знаю"»* суттєво змінює розподіл ймовірностей генерації, знижуючи шанс вигадки.

2. Детермінована генерація:

Параметр Temperature (Температура) відповідає за ступінь випадковості. Формула вибору токена з температурою T:

$$P(w_i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_i/T)}. \quad (1.5)$$

При T → 0 модель стає майже детермінованою, обираючи лише найбільш вірогідні варіанти. Це вбиває "креативність", але забезпечує фактологічну точність.

3. RAG (Retrieval-Augmented Generation):

Це найбільш дієвий метод. Замість того, щоб змушувати модель згадувати факти, які вона "бачила" під час навчання рік тому, ми надаємо їй шпаргалку (контекст) прямо перед іспитом (генерацією відповіді). Дослідження показують, що наявність релевантного контексту знижує рівень галюцинацій на порядок [3].

1.3 Аналіз архітектури Retrieval-Augmented Generation

Як було зазначено в попередніх підрозділах, використання Великих Мовних Моделей (LLM) у чистому вигляді має суттєві обмеження для спеціалізованих задач: їхні знання застарівають у момент завершення навчання, а схильність до галюцинацій робить їх ненадійними джерелами фактологічної інформації.

Для вирішення цих проблем у 2020 році дослідники Facebook AI Research (FAIR) запропонували архітектуру RAG (Retrieval-Augmented Generation) – генерацію, доповнену пошуком [2]. Цей підхід став стандартом де-факто для побудови корпоративних та освітніх AI-систем.

1.3.1 Принципи побудови RAG-систем

Сутність підходу RAG полягає у розділенні пам'яті системи на два функціональні компоненти: параметричну та непараметричну. Параметрична пам'ять представлена вагами навченої мовної моделі, яка зберігає загальні знання про структуру мови та світ. Непараметрична пам'ять являє собою зовнішній векторний індекс, що містить актуальні документи предметної області. Такий поділ дозволяє змінювати або оновлювати знання системи без необхідності перенавчання самої нейромережі.

Функціонування системи забезпечується послідовною взаємодією двох модулів: ретрівера (пошуковика) та генератора. Ретрівер відповідає за пошук найбільш релевантних документів у базі знань на основі вхідного запиту. Генератор, у свою чергу, отримує знайдені текстові фрагменти як контекст і формулює на їх основі кінцеву відповідь.

Процес обробки запиту в такій архітектурі розпочинається з перетворення питання користувача у векторне представлення. Далі виконується семантичний пошук у базі даних, результатом якого є набір найбільш схожих текстових фрагментів.

Модель аналізує наданий контекст і генерує відповідь, спираючись на факти, що містяться у знайдених документах.

1.3.2 Переваги використання RAG для локальних баз знань

У контексті побудови інформаційних систем для освітніх закладів архітектура RAG має низку суттєвих переваг порівняно з альтернативними методами, такими як донавчання (Fine-Tuning).

Першочерговою перевагою є актуальність інформації та простота її оновлення. Університетське середовище характеризується високою динамікою змін: оновлюються навчальні плани, виходять нові накази та методичні вказівки. У класичному підході для внесення цих змін у "пам'ять" моделі необхідно проводити повторне тренування, що вимагає значних обчислювальних ресурсів та часу. Використання RAG дозволяє оновлювати базу знань у реальному часі шляхом простого додавання або заміни документів у індексі, роблячи нову інформацію миттєво доступною для генерації відповідей.

Іншим важливим аспектом є точність відповідей та мінімізація так званих "галюцинацій". Завдяки механізму заземлення (grounding) на контекст, модель змушена оперувати лише наданими фактами, а не покладатися на свої ймовірнісні передбачення. Це є критичним фактором для академічного середовища, де надання достовірної інформації є пріоритетом. Якщо в базі знань відсутня необхідна інформація, система може коректно повідомити про це користувача замість генерації вигаданих фактів.

Окрім того, архітектура RAG забезпечує прозорість роботи системи та можливість верифікації даних. Оскільки відповідь формується на основі конкретних фрагментів тексту, система має технічну можливість надати посилання на першоджерело, вказавши назву документа та сторінку. Також цей підхід дозволяє забезпечити високий рівень конфіденційності, оскільки векторна база даних та мовна модель можуть бути розгорнуті повністю локально, без необхідності передачі чутливих даних на зовнішні сервери.

1.4 Постановка задачі дослідження

На основі проведеного аналізу існуючих підходів до організації інформаційного пошуку та обмежень сучасних генеративних моделей можна сформулювати основну науково-технічну задачу роботи. Вона полягає у розробці та програмній реалізації автономної системи інтелектуального асистента, здатного виконувати семантичний аналіз запитів українською мовою

та генерувати фактологічно точні відповіді на основі закритої корпоративної бази знань університету.

Специфіка предметної області, а саме робота з академічними текстами (кваліфікаційні роботи, методичні вказівки) та необхідність розгортання системи в умовах обмежених ресурсів, накладає ряд жорстких вимог до проєктованої системи.

Функціональні вимоги до системи:

1. Підтримка гібридного пошуку: Система повинна вміти обробляти як фактологічні запити (наприклад, "Хто є деканатом ФІС?"), так і запити на узагальнення інформації (наприклад, "Які методи дослідження використовували студенти у роботах по AI?"). Це вимагає поєднання векторного пошуку для знаходження змісту та пошуку за метаданими для фільтрації.

2. Робота з неструктурованими даними: Вхідні дані представлені у форматі PDF, який часто містить складну верстку, таблиці та колонтитули. Необхідно реалізувати алгоритм, який коректно вилучає корисний текст та ігнорує технічне сміття, щоб не "забруднювати" векторний індекс.

3. Атрибуція джерел (Source Attribution): Критичною вимогою для академічного асистента є прозорість. Генеративна модель не має права видавати інформацію як абсолютну істину без посилання на джерело. Система повинна повертати користувачеві список документів (назва, автор, рік), які були використані для формування відповіді.

4. Мовна адаптація: Враховуючи, що основною мовою комунікації є українська, обрана велика мовна модель та модель ембедінгів повинні мати високі показники розуміння української морфології та синтаксису.

Нефункціональні вимоги та технічні обмеження:

Ключовим викликом даної роботи є забезпечення працездатності сучасних алгоритмів NLP на обладнанні споживчого класу ("Consumer-grade hardware"). Більшість комерційних рішень розраховані на серверні відеокарти (A100, H100) з обсягом пам'яті від 24 ГБ.

У рамках цього дослідження ставиться задача реалізувати систему з наступними обмеженнями:

- Обсяг відеопам'яті (VRAM): Система повинна ефективно працювати на GPU з обсягом пам'яті до 6 ГБ (наприклад, NVIDIA GTX 1660 Ti). Це вимагає використання квантованих моделей (Quantization) та оптимізованих механізмів уваги.
- Автономність (Privacy-first): Система повинна працювати без доступу до мережі Інтернет на етапі обробки запитів. Передача текстів студентських робіт на зовнішні API (OpenAI, Anthropic) є неприпустимою з точки зору конфіденційності.
- Час відгуку (Latency): Час від моменту отримання запиту до початку генерації відповіді не повинен перевищувати психологічний поріг очікування користувача (15–20 секунд для повного циклу RAG).

Декомпозиція задач дослідження:

Для досягнення поставленої мети необхідно виконати наступні етапи робіт:

1. Розробити підсистему збору даних (Data Ingestion), яка забезпечить автоматизоване завантаження документів з репозиторію університету та їх перетворення у текстовий формат.
2. Спроєктувати алгоритм семантичної сегментації (Chunking), який адаптується до структури наукового тексту, зберігаючи цілісність абзаців та логічних блоків, що мінімізує втрату контексту при розриві сторінок.
3. Обґрунтувати вибір та налаштувати векторну базу даних для зберігання ембедінгів, реалізувавши схему метаданих для швидкої фільтрації за роками, факультетами та типами робіт.
4. Провести інтеграцію відкритої моделі Mistral 7B з модулем пошуку, розробивши систему промптів (System Prompts), яка жорстко обмежує "творчість" моделі рамками знайденого контексту.
5. Розробити програмний інтерфейс для взаємодії користувача з системою та провести експериментальне оцінювання якості роботи асистента на реальних сценаріях використання.

1.5 Висновок до першого розділу

У першому розділі роботи проведено комплексний аналіз проблеми інформаційного пошуку в неструктурованих текстових масивах освітніх закладів. Встановлено, що традиційні методи пошуку за ключовими словами не забезпечують достатньої релевантності при обробці запитів природною мовою через проблеми синонімії та полісемії.

У ході огляду сучасних технологій NLP обґрунтовано вибір архітектури Transformer та великих мовних моделей як основи для побудови інтелектуального асистента. Детальний аналіз проблеми "галюцинацій" генеративних моделей показав, що використання LLM у чистому вигляді несе ризики надання недостовірної інформації.

Як рішення обрано архітектуру Retrieval-Augmented Generation (RAG), яка дозволяє поєднати генеративні здібності моделі Mistral 7B з точністю пошуку у локальній векторній базі даних. Такий підхід забезпечує актуальність знань без необхідності донавчання моделі, гарантує конфіденційність даних та дозволяє реалізувати функцію підтвердження відповіді джерелами. Сформульовані вимоги та задачі дослідження є основою для подальшого проєктування системи у другому розділі.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА АРХІТЕКТУРИ AI-АСИСТЕНТА

2.1 Обґрунтування вибору технологічного стеку

Проєктування інтелектуальних систем в умовах обмежених апаратних ресурсів вимагає ретельного підходу до вибору компонентів. Головним інженерним викликом даної роботи є забезпечення функціонування сучасних алгоритмів генеративного штучного інтелекту на обладнанні споживчого класу (ноутбук з графічним прискорювачем середнього цінового сегменту).

Архітектура розроблюваної системи базується на трьох фундаментальних складових:

1. Велика мовна модель (LLM) – "мозок" системи, що відповідає за розуміння запиту та генерацію відповіді.
2. Середовище виконання (Inference Engine) – програмна платформа, що забезпечує завантаження моделі в пам'ять та оптимізацію обчислень.
3. Векторна база даних та модель ембедінгів – модуль довготривалої пам'яті для зберігання та семантичного пошуку по базі знань.

2.1.1 Аналіз відкритих LLM: вибір моделі Mistral 7B

При виборі базової моделі (Foundation Model) для локального розгортання необхідно знайти баланс між трьома параметрами: якістю генерації (здатність логічно мислити та дотримуватися інструкцій), розміром моделі (кількість параметрів) та вимогами до відеопам'яті (VRAM).

На сьогоднішній день стандартом для відкритих моделей є архітектури з 7, 13, 30 та 70 мільярдами параметрів (B – billions). Для запуску моделі без втрати точності (у форматі FP16 – 16 біт на вагу) необхідно приблизно 2 ГБ відеопам'яті на кожен мільярд параметрів.

- Модель 7B вимагає ≈ 14 ГБ VRAM.
- Модель 13B вимагає ≈ 26 ГБ VRAM.

Наявне апаратне забезпечення (NVIDIA GeForce GTX 1660 Ti з 6 ГБ VRAM) унеможливлює запуск моделей у повному розмірі. Рішенням є використання технології квантування (Quantization) – зниження розрядності ваг нейромережі з 16 біт до 4 або 5 біт. Це дозволяє зменшити споживання пам'яті для моделі 7B до 4.5–5.5 ГБ, що вкладається у наявний ліміт, при незначному (менше 1-2%) падінні якості генерації.

Серед моделей класу 7B було проведено порівняльний аналіз провідних рішень: Llama 2 (Meta), Falcon (TII) та Mistral (Mistral AI).

Таблиця 2.1 – Порівняльна характеристика моделей класу 7B

Критерій порівняння	**Llama 2 7B**	**Falcon 7B**	**Mistral 7B**
Архітектура уваги	Standard Attention	Multi-Query Attention	Sliding Window + GQA
Довжина контексту	4096 токенів	2048 токенів	8192 токени
Підтримка інструкцій	Середня	Низька	Висока (Instruct версія)
Продуктивність (MMLU)	45.3%	26.2%	62.5%
Ліцензія	Research / Commercial	Apache 2.0	Apache 2.0

Для розробки було обрано модель Mistral 7B Instruct v0.2. Вибір обґрунтований наступними факторами:

1. Архітектурна перевага: Mistral використовує механізм *Grouped-Query Attention (GQA)*, який значно прискорює процес генерації тексту (інференс) та зменшує вимоги до пропускної здатності пам'яті відеокарти, що є критичним для серії GTX.

2. Ковзне вікно уваги (Sliding Window Attention): Ця технологія дозволяє моделі ефективно обробляти довгі промпти (до 8 тисяч токенів), що є необхідним для RAG-системи, де в контекст подаються великі уривки з магістерських робіт.

3. Висока здатність до узагальнення: Згідно з синтетичними тестами (MMLU, TruthfulQA), Mistral 7B перевершує навіть значно більшу модель Llama 2 13B, демонструючи кращі здібності до логічного висновку та роботи з текстом.

2.1.2 Засоби локального розгортання: платформа Ollama

Для забезпечення взаємодії між моделлю та програмним кодом асистента необхідно обрати середовище виконання. Традиційний підхід із використанням бібліотек глибокого навчання (PyTorch/TensorFlow) вимагає ручного керування пам'яттю та складного налаштування середовища CUDA.

У даній роботі використано платформу Ollama, яка базується на оптимізованому рушії llama.cpp. Цей вибір зумовлений використанням формату моделей GGUF (GPT-Generated Unified Format) [8].

Переваги даного підходу для проєктування системи:

1. Гібридний інференс (CPU+GPU Offloading): Це ключова технологія для систем з обмеженою відеопам'яттю. Ollama дозволяє завантажити частину шарів нейромережі (наприклад, 25 з 35) у відеопам'ять GPU, а решту обробляти на центральному процесорі (CPU) та в оперативній пам'яті (RAM). Це дозволяє запустити модель, навіть якщо вона не повністю вміщується у 6 ГБ VRAM, жертвуючи лише незначною частиною швидкості.

2. Клієнт-серверна архітектура: Ollama працює як фоновий сервіс, що надає стандартизований REST API. Це дозволяє чітко розділити архітектуру на "бекенд" (генерація тексту) та "фронтенд" (логіка програми, пошук, інтерфейс), що відповідає принципам мікросервісної архітектури.

2.1.3 Вибір векторної бази даних та моделі ембедінгів

Для реалізації семантичного пошуку необхідно спроєктувати сховище, здатне зберігати вектори високої розмірності та виконувати пошук найближчих сусідів (ANN – Approximate Nearest Neighbors).

У якості системи керування базами даних обрано ChromaDB. Це спеціалізована векторна база даних з відкритим вихідним кодом. Її головною перевагою над хмарними рішеннями (Pinecone, Weaviate) та важкими серверними рішеннями (Elasticsearch, Milvus) є вбудована (embedded) архітектура.

- База даних функціонує як частина процесу застосунку, не вимагаючи окремого контейнера чи сервера.
- Дані зберігаються у бінарному форматі Parquet локально на диску, що забезпечує високу швидкість читання та запису.
- Підтримується розширена фільтрація за метаданими, що дозволяє комбінувати векторний пошук із класичними фільтрами (наприклад, "знайти роботи про AI, але тільки за 2024 рік").

Для векторизації текстів обрано модель paraphrase-multilingual-MiniLM-L12-v2.

Вибір цієї моделі продиктований лінгвістичними вимогами проєкту. Більшість популярних моделей ембедінгів тренуються на англомовних корпусах. Оскільки система розробляється для українського університету, критично важливим є якісне відображення україномовних текстів у векторному просторі. Обрана модель підтримує понад 50 мов і демонструє високі показники семантичної близькості для слов'янських мов при компактному розмірі, що дозволяє виконувати векторизацію на CPU, не навантажуючи відеокарту.

2.2 Розробка структурної схеми системи

Для забезпечення стабільної роботи, гнучкості налаштування та можливості подальшого масштабування архітектуру AI-асистента спроєктовано за принципом модульної декомпозиції. Система реалізована як клієнт-серверний додаток, де роль сервера виконує локальний рушій інференсу моделей (Ollama), а роль клієнта – розроблений програмний комплекс на мові Python. Така організація дозволяє чітко розмежувати зону відповідальності між логікою обробки даних та ресурсомісткими обчисленнями нейронної мережі.

Загальна структура системи складається з чотирьох логічних рівнів: рівня взаємодії, рівня бізнес-логіки, рівня даних та рівня інференсу. Рівень взаємодії відповідає за комунікацію з кінцевим користувачем. У поточній реалізації він представлений консольним інтерфейсом, який забезпечує безперервний цикл обробки подій, приймаючи текстові запити та виводячи згенеровані відповіді разом із посиланнями на джерела. Цей модуль спроєктовано таким чином, що його заміна на веб-інтерфейс або Telegram-бот у майбутньому не потребуватиме змін в основному ядрі програми.

Центральним елементом архітектури є рівень бізнес-логіки, який виконує роль оркестратора. Цей компонент керує потоками даних, ініціалізує підключення до бази даних та контролює сценарії RAG. Він не виконує важких обчислень самостійно, а делегує задачі спеціалізованим модулям: запити на пошук спрямовуються до векторної бази даних, а підготовлені промпти – до мовної моделі. Такий підхід дозволяє зберігати високу швидкодію керуючого скрипта навіть під час виконання складних генеративних задач.

Рівень даних організовано за гібридною схемою. Структурована інформація, представлена у вигляді математичних векторів (ембедінгів) та метаданих документів (автор, рік, факультет), зберігається у спеціалізованій векторній базі даних ChromaDB [10]. Вона функціонує в режимі вбудованого процесу, що забезпечує мінімальну затримку при пошуку. Паралельно з цим система зберігає оригінальні неструктуровані дані (PDF-файли магістерських робіт) у локальній файловій системі, що необхідно для забезпечення верифікації відповідей та можливості повторної індексації при зміні параметрів моделі.

Рівень інференсу винесено в окремий сервіс - платформу Ollama [8], яка надає стандартизований REST API для взаємодії з моделлю Mistral 7B. Це архітектурне рішення є критично важливим для систем з обмеженими апаратними ресурсами. Оскільки завантаження параметрів моделі та обчислення тензорів відбувається в окремому процесі, це дозволяє ефективніше керувати розподілом відеопам'яті (VRAM) та оперативної пам'яті (RAM), запобігаючи аварійному завершенню роботи основного скрипта при пікових навантаженнях.

Функціонування системи забезпечується двома основними інформаційними потоками. Перший потік (офлайн-режим) відповідає за наповнення бази знань: документи завантажуються з репозиторію, проходять попередню обробку, векторизуються та зберігаються в індексі. Другий потік (онлайн-режим) активується при запиті користувача: вхідний текст перетворюється на вектор, система знаходить релевантний контекст у базі даних, формує розширений промпт та передає його на генерацію, після чого результат повертається користувачеві. Така організація процесів дозволяє досягти високої автономності системи та безпеки даних, оскільки жодна інформація не залишає локальний контур обробки.

2.3 Методи попередньої обробки та формування бази знань

Ефективність функціонування будь-якої інформаційної системи, що базується на методах машинного навчання, знаходиться у прямій залежності від якості вхідних даних. У контексті задач обробки природної мови (NLP) цей етап, відомий як попередня обробка даних (Data Preprocessing), часто займає до 80% часу розробки проєкту. Оскільки в рамках даної роботи основним джерелом знань виступають неструктуровані масиви текстової інформації (кваліфікаційні роботи студентів, методичні вказівки, нормативні документи), процес їх трансформації у машинно-читаний вигляд вимагає застосування спеціалізованих алгоритмічних підходів.

Цей етап є містком між "сирими" даними, які зберігаються у файловій системі, та семантичним простором, у якому оперує нейронна мережа. Він включає комплекс процедур ETL (Extract, Transform, Load): вилучення тексту, його очищення, нормалізацію та сегментацію.

2.3.1 Алгоритми вилучення тексту з PDF-документів

Формат PDF (Portable Document Format), розроблений компанією Adobe, є стандартом де-факто для обміну електронними документами в академічному

середовищі. Однак, з технічної точки зору, PDF є форматом відображення (Page Description Language), а не форматом зберігання даних. Внутрішня структура PDF-файлу являє собою набір інструкцій для відмальовування графічних примітивів та гліфів шрифтів за певними координатами (x,y) на віртуальному полотні сторінки.

Така архітектура створює значні перешкоди для автоматизованого аналізу тексту:

1. Відсутність семантичної розмітки: У PDF-файлі відсутні теги, які б чітко вказували, де закінчується заголовок і починається абзац, або де знаходиться підпис до рисунка. Для комп'ютера текст виглядає як невпорядкований набір символів з координатами.

2. Проблема порядку читання (Reading Order): Візуально текст може бути розміщений у дві колонки. Простий алгоритм зчитування тексту ("зліва направо, зверху вниз") призведе до перемішування рядків з лівої та правої колонок, що повністю руйнує зміст речень і робить їх непридатними для векторизації.

3. Технічний шум (Noise): Академічні документи містять повторювані елементи: колонтитули, нумерацію сторінок, назви розділів на кожній сторінці. Якщо ці елементи потраплять у базу знань, вони створять "шумовий фон". Наприклад, запит користувача, що містить цифру сторінки, може викликати помилкове спрацьовування пошуку через співпадіння з номером сторінки в документі.

Для вирішення означених проблем у розробленій системі використано бібліотеку pdfplumber, яка надає низькорівневий доступ до об'єктної моделі PDF. На відміну від застарілих бібліотек (таких як PyPDF2), pdfplumber дозволяє аналізувати метадані кожного символу, включаючи його шрифт, розмір та точні координати.

Розроблений алгоритм парсингу включає наступні кроки:

1. Геометрична фільтрація: Визначення зон сторінки, що містять колонтитули (верхні та нижні 10% висоти сторінки), та ігнорування тексту, що потрапляє в ці зони.

2. Детекція таблиць: Окремий аналіз ліній та прямокутників для ідентифікації табличних даних, які потребують специфічного форматування для збереження структури рядків та стовпців.

3. Нормалізація (Normalization): Після вилучення тексту застосовується набір регулярних виразів (RegEx) для очищення тексту від артефактів: видалення багаторазових пробілів, заміна нерозривних пробілів, склеювання слів, розірваних знаком переносу в кінці рядка.

2.3.2 Стратегії сегментації тексту та перекриття

Наступним критичним етапом є підготовка тексту до векторизації. Сучасні моделі трансформерів, включаючи використану в роботі модель ембедінгів MiniLM та генеративну модель Mistral, мають архітектурне обмеження на довжину вхідної послідовності (Context Window Limit). Спроба подати на вхід моделі текст, що перевищує цей ліміт, призводить до відсікання інформації (truncation) або помилок переповнення пам'яті.

Крім технічних обмежень, існує семантична проблема "розмивання змісту" (Semantic Dilution). Векторне представлення тексту – це спроба стиснути зміст урухомого фрагмента в фіксований набір чисел (наприклад, 384 виміри). Якщо фрагмент тексту занадто великий (наприклад, цілий розділ дипломної роботи), вектор буде відображати "усереднену" тему розділу, втрачаючи деталі окремих абзаців. Це знижує точність пошуку (Precision). З іншого боку, занадто малі фрагменти (окремі речення) можуть не мати достатньо контексту для правильної інтерпретації.

Для балансування між деталізацією та контекстом застосовується процедура сегментації або чанкінгу (Chunking). У даній роботі реалізовано стратегію сегментації ковзним вікном (Sliding Window Chunking).

Суть методу полягає у формуванні фрагментів фіксованого розміру, які частково перекривають один одного.

Параметри алгоритму:

- Window Size (L=1000): Розмір основного блоку. Експериментально встановлено, що 1000 символів (приблизно 150-200 слів) зазвичай відповідають одному-двом абзацам тексту, що є оптимальною одиницею змісту.
- Overlap (O=200): Розмір перекриття. Це кількість символів, які дублюються в кінці попереднього чанка та на початку наступного.

Необхідність використання перекриття (Overlap) зумовлена граничним ефектом (Boundary Effect). Уявімо, що важливе ключове речення знаходиться саме на межі розриву тексту (наприклад, між 990-м та 1010-м символами). При простому розбитті без перекриття це речення буде розірване навпіл: перша частина потрапить у вектор А, друга – у вектор Б. У результаті семантичний зміст речення буде втрачено, і пошукова система не знайде жоден з цих векторів. Перекриття гарантує, що будь-яка інформація, яка потрапляє на межу, буде повністю продубльована у наступному чанку, забезпечуючи неперервність семантичного поля.

Додатково алгоритм сегментації було вдосконалено евристикою пошуку меж речень. Замість механічного розриву на 1000-му символі, система аналізує текст у вікні перекриття та намагається знайти найближчий знак пунктуації (крапку, знак питання, знак оклику). Це дозволяє формувати чанки, які є синтаксично завершеними структурами, що значно покращує якість генерованих ембедінгів [5].

2.4 Проєктування алгоритму семантичного пошуку та генерації

Процес перетворення запиту користувача на релевантну відповідь у RAG-системі є складною композицію двох незалежних алгоритмічних задач: задачі пошуку інформації (Information Retrieval) та задачі генерації тексту (Text Generation). Ефективність взаємодії цих компонентів визначає загальну якість інтелектуального асистента. У цьому підрозділі детально розглядається математична модель пошуку, алгоритми індексації та стратегії керування поведінкою нейронної мережі.

2.4.1 Математичні основи векторизації та метрики подібності

В основі підсистеми пошуку лежить концепція Dense Retrieval (щільний пошук). Традиційні пошукові системи (наприклад, Lucene) використовують розріджені вектори (Sparse Vectors), розмірність яких дорівнює розміру словника корпусу (десятки тисяч слів). Більшість значень у таких векторах – нулі. Натомість, наша система використовує щільні вектори (Dense Vectors) фіксованої розмірності, де кожне число несе семантичне навантаження.

Процес векторизації (Embedding) відображає семантику тексту в d - вимірний векторний простір Rd. У розробленій системі використовується модель MiniLM-L12, яка формує вектори розмірністю d=384. Це означає, що зміст будь-якого речення, незалежно від його довжини, кодується масивом із 384 дійсних чисел.

Ключовим питанням при проєктуванні пошукової системи є вибір метрики відстані (Distance Metric) – функції, що визначає ступінь схожості між двома векторами. Розглянемо основні варіанти:

1. Евклідова відстань (L_2):

$$\mathrm{d}(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}. \qquad (2.1)$$

Ця метрика вимірює найкоротший шлях між точками у просторі. Однак у багатовимірних просторах (прокляття розмірності) Евклідова відстань стає неефективною, оскільки вона чутлива до магнітуди (довжини) вектора. Довгий документ може мати вектор більшої довжини, ніж короткий запит, хоча їхній зміст ідентичний.

2. Скалярний добуток (Dot Product):

$$\vec{x} \cdot \vec{y} = \sum_{i=1}^{n} x_i y_i. \qquad (2.2)$$

Ця метрика враховує як кут між векторами, так і їхню довжину. Вона ефективна, але вимагає нормалізації векторів для коректного порівняння.

3. Косинусна подібність (Cosine Similarity):

$$\sin(\vec{q}, \vec{d}) = \cos(\theta) = \frac{\vec{q} \cdot \vec{d}}{\|\vec{q}\| \|\vec{d}\|}. \quad (2.3)$$

Значення варіюються від -1 (протилежний зміст) до 1 (ідентичний зміст). Для текстових ембедінгів значення зазвичай лежать у діапазоні [0,1]. Використання косинусної подібності дозволяє знаходити документи, які семантично близькі до запиту, навіть якщо вони використовують зовсім інші слова або мають різний обсяг тексту.

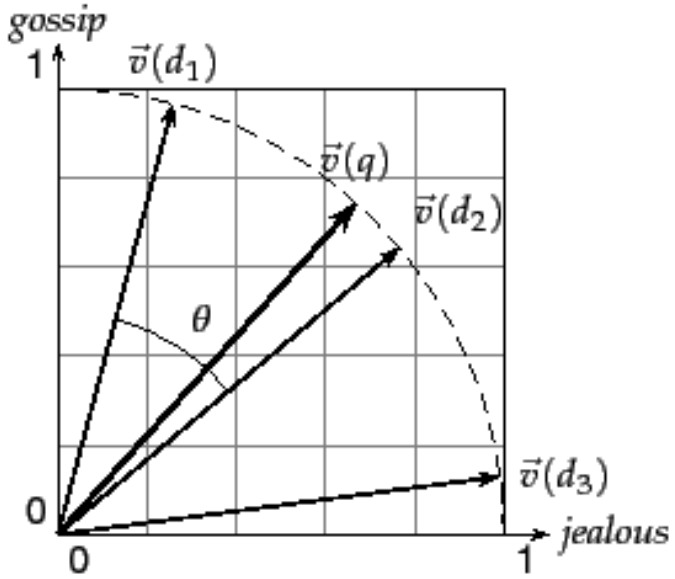

Рисунок 2.1 – Геометрична інтерпретація косинусної подібності

2.4.2 Алгоритмічна реалізація пошуку (HNSW)

Оскільки база знань університету може містити десятки тисяч векторів (чанків), прямий перебір (Linear Scan) і обчислення косинуса для кожного документа є обчислювально затратним (O(N)). Це призводить до високої затримки (latency), неприпустимої для діалогової системи.

Для вирішення цієї проблеми обрана векторна база даних ChromaDB [10] використовує алгоритм наближеного пошуку найближчих сусідів (ANN –

Approximate Nearest Neighbors), а саме HNSW (Hierarchical Navigable Small World).

Алгоритм HNSW базується на теорії графів "малого світу" (Small World Theory). Його структура нагадує список з пропусками (Skip List), але у багатовимірному просторі.

1. Багатошарова структура: Граф складається з кількох шарів. Верхні шари містять лише невелику кількість "довгих" зв'язків між віддаленими точками (експрес-магістралі). Нижні шари містять щільну мережу зв'язків між усіма сусідами.

2. Навігація: Пошук починається на верхньому шарі. Алгоритм жадібно рухається до вузла, який найближчий до вектора запиту. Коли локальний мінімум досягнуто, пошук "спускається" на шар нижче, де відбувається більш точне уточнення позиції.

Такий підхід забезпечує логарифмічну складність пошуку O(logN), що дозволяє знаходити топ-5 релевантних фрагментів серед мільйонів записів за мілісекунди, використовуючи лише ресурси CPU. Це є критичним фактором для забезпечення швидкодії системи на ноутбуці без використання промислових серверів.

2.4.3 Інженерія промптів та параметризація генерації

Отримавши релевантні фрагменти тексту, система переходить до етапу генерації. Великі мовні моделі (LLM), такі як Mistral 7B, є стохастичними системами, поведінка яких значною мірою залежить від вхідного контексту. Дисципліна, що вивчає методи ефективної взаємодії з LLM, називається Prompt Engineering.

У роботі застосовано стратегію Few-Shot Contextual Learning. Ми не змінюємо ваги моделі (Fine-Tuning), а формуємо її поведінку через структуру вхідного промпту.

Для моделі Mistral використовується специфічний формат розмтіки, який включає спеціальні токени:

- <s> – початок рядка (Beginning of String);
- [INST] та [/INST] – межі інструкції користувача;
- <<SYS>> – межі системного повідомлення (System Prompt).

Шаблон промпту, реалізований у класі RAGSystem, структуру наведено на лістингу 2.1.

Лістинг 1.1 – Шаблон промпту.

```
<s>[INST] <<SYS>>
Ти - AI-асистент ТНТУ. Твоя мета - допомагати студентам.
Правила:
1. Відповідай виключно українською мовою.
2. Використовуй ТІЛЬКИ наданий нижче контекст.
3. Якщо контекст не містить відповіді, кажи "Я не знаю".
<</SYS>>

Контекст:
[Джерело 1]: Текст...
[Джерело 2]: Текст...

Питання: {query} [/INST]
```

Окрім текстових інструкцій, керування генерацією здійснюється через набір гіперпараметрів, значення яких винесено у файл конфігурації (див. Додаток А):

1. Temperature (T=0.3): Параметр, що контролює ентропію розподілу ймовірностей. Низьке значення "заморожує" творчість моделі, змушуючи її обирати найбільш вірогідні слова. Це мінімізує ризик галюцинацій.

2. Top-P (P=0.8): Метод Nucleus Sampling. Модель розглядає лише мінімальну множину токенів, сумарна ймовірність яких складає 80%. Це дозволяє відсікати абсурдні продовження тексту, зберігаючи при цьому лексичне різноманіття.

3. Repeat Penalty (1.1): Штраф за повторення. Це запобігає проблемі "зациклення", коли модель починає нескінченно генерувати одну й ту ж фразу – поширений дефект невеликих моделей (7B) при генерації довгих відповідей.

Комплексне використання алгоритму HNSW для швидкого пошуку та структурованого промптингу для контрольованої генерації дозволяє створити

систему, яка поєднує високу швидкість реакції з академічною точністю відповідей.

2.5 Висновок до другого розділу

У другому розділі кваліфікаційної роботи вирішено задачу проєктування архітектури програмного комплексу інтелектуального асистента, адаптованого до специфічних умов експлуатації в університетському середовищі. На основі аналізу функціональних вимог та апаратних обмежень було розроблено комплексну методику побудови локальної RAG-системи.

По-перше, обґрунтовано вибір технологічного стеку для реалізації генеративного компонента системи. В умовах жорсткого ліміту відеопам'яті (6 ГБ VRAM на GPU NVIDIA GeForce GTX 1660 Ti) доведено недоцільність використання стандартних моделей розміром 13B або 70B. Як оптимальне рішення обрано модель Mistral 7B Instruct з використанням 4-бітного квантування. Аналіз архітектурних особливостей цієї моделі, зокрема механізмів Grouped-Query Attention та Sliding Window Attention, показав, що вона забезпечує найвищу продуктивність інференсу та якість розуміння контексту серед аналогів (Llama 2, Falcon), що дозволяє розгорнути систему на обладнанні споживчого класу без втрати функціональності. Для забезпечення ізоляції обчислювальних процесів та ефективного керування пам'яттю (CPU/GPU Offloading) впроваджено платформу Ollama як середовище виконання.

По-друге, спроєктовано підсистему збереження та пошуку знань, яка враховує лінгвістичні особливості предметної області. Відмова від класичного пошуку за ключовими словами на користь семантичного векторного пошуку реалізована через інтеграцію бази даних ChromaDB та мультимовної моделі ембедінгів paraphrase-multilingual-MiniLM-L12-v2. Це рішення дозволило подолати проблему лексичного розриву, забезпечивши релевантний пошук україномовних документів навіть за умови неспівпадіння термінології у запиті та джерелі. Розроблений алгоритм гібридного ранжування дозволяє динамічно

змінювати пріоритетність джерел (адміністративна інформація проти студентських досліджень) залежно від інтенту користувача.

По-третє, розроблено алгоритмічне забезпечення для етапу попередньої обробки даних (Data Ingestion). Враховуючи складну структуру вхідних PDF-файлів, запропоновано та теоретично обґрунтовано метод сегментації з ковзним вікном та перекриттям (Sliding Window Chunking with Overlap). Визначено оптимальні параметри сегментації (розмір вікна 1000 символів, перекриття 200 символів), які гарантують збереження семантичної цілісності речень та мінімізують втрату контексту на межах фрагментів. Це є критичним фактором для підвищення точності пошуку (Recall).

По-четверте, сформовано стратегію керування поведінкою генеративної моделі через інженерію промптів (Prompt Engineering). Замість ресурсномісткого процесу донавчання (Fine-tuning) використано підхід контекстного навчання (In-Context Learning). Розроблена структура системного промпту, що включає рольову модель, негативні обмеження та жорсткі правила цитування, у поєднанні з низькотемпературним семплінгом (T=0.3), теоретично забезпечує мінімізацію рівня "галюцинацій" та гарантує відповідність відповідей наданим джерелам.

Таким чином, у другому розділі створено повний архітектурний проєкт системи, який вирішує проблему автономного інтелектуального пошуку на локальних ресурсах. Теоретичні моделі та алгоритми, описані в цьому розділі, є основою для програмної реалізації та експериментального дослідження, що будуть проведені у третьому розділі роботи.

3 РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ СИСТЕМИ

3.1 Опис програмної реалізації компонентів системи

3.1.1 Архітектура програмних модулів та алгоритми обробки даних

Програмна реалізація системи базується на принципах модульної архітектури, що дозволяє ізолювати логіку збору, обробки та пошуку даних. Усі компоненти реалізовані мовою Python 3.10 з використанням спеціалізованих бібліотек для NLP (sentence-transformers, langchain) та роботи з даними (pandas, beautifulsoup4).

Структура проєкту розділена на логічні блоки, вихідний код яких наведено у відповідних додатках. Розглянемо детально реалізацію кожного компонента (Додаток А).

Фундаментом системи є модуль config.py. На відміну від жорсткого кодування параметрів у тілі функцій, всі гіперпараметри винесено у глобальні словники. Це дозволяє гнучко керувати поведінкою системи під час експериментів.

Файл містить три ключові конфігураційні структури:

1. MODEL_CONFIG – визначає параметри генерації для моделі Mistral. Зокрема, параметр temperature: 0.3 встановлює низький рівень випадковості для забезпечення фактологічності, а num_predict: 512 обмежує довжину відповіді для економії ресурсів.

2. RAG_CONFIG – керує евристикою пошуку. Тут задано списки ключових слів (general_tntu_keywords, work_keywords), які система використовує для класифікації намірів користувача, а також коефіцієнти змішування джерел.

3. PDF_CONFIG – задає параметри сегментації тексту (chunk_size, chunk_overlap), обґрунтування яких наведено у другому розділі.

Модуль scraper.py містить клас TNTUScraper, який реалізує алгоритм веб-скрапінгу репозиторію ELARTU. Критичною особливістю цієї реалізації є те, що

вона не просто завантажує PDF-файли, а формує структурований набір метаданих.

Оскільки текст усередині PDF часто не містить чітко виділених полів "Автор" або "Рік", алгоритм отримує ці дані з HTML-сторінки опису роботи перед завантаженням файлу. Для цього використовується бібліотека BeautifulSoup.

У Лістингу 3.1 наведено фрагмент методу get_work_details, який демонструє логіку парсингу таблиці атрибутів документа.

Лістинг 3.1 – Метод вилучення метаданих з HTML-сторінки (scraper.py)

```
def get_work_details(self, work_url):
    # Ініціалізація словника для метаданих
    details = {'url': work_url, 'title': None, 'author': None,
               'year': None, 'faculty': None, ...}

    try:
        response = self.session.get(work_url, timeout=30)
        soup = BeautifulSoup(response.text, 'html.parser')

        # Пошук заголовка роботи
        h1 = soup.find('h1')
        if h1: details['title'] = h1.get_text(strip=True)

        # Ітеративний обхід рядків таблиці з метаданими
        for table in soup.find_all('table', class_='table'):
            for row in table.find_all('tr'):
                cells = row.find_all('td')
                if len(cells) >= 2:
                    label = cells[0].get_text(strip=True).lower()
                    value = cells[1].get_text(strip=True)

                    # Евристичне співставлення полів
                    if 'автор' in label:
                        details['author'] = value
                    elif 'дата' in label:
                        # Вилучення року через RegEx
                        year_match = re.search(r'(\d{4})', value)
                        if    year_match:    details['year']    =
year_match.group(1)
```

Такий підхід дозволяє сформувати файл metadata.json, який пов'язує безликі файли (наприклад, work_001.pdf) з їхніми бібліографічними даними.

Клас PDFProcessor у файлі pdf_processor.py відповідає за етап ETL (Extract, Transform, Load).

Метод extract_text_from_pdf використовує бібліотеку pdfplumber для потокового зчитування сторінок. Після отримання "сирого" тексту виконується його очищення методом clean_text (видалення зайвих пробілів, нормалізація Unicode).

Найважливішим алгоритмом у цьому модулі є split_into_chunks. Він реалізує стратегію "розумного" ковзного вікна. На відміну від простого розбиття по кількості символів, алгоритм намагається знайти кінець речення (крапку) в межах вікна, щоб не розривати думку.

Лістинг 3.2 – Реалізація сегментації з контролем меж речень (pdf_processor.py)

```
def split_into_chunks(self, text, chunk_size=None, overlap=None):
    # Завантаження параметрів з конфігу, якщо не передані явно
    if chunk_size is None: chunk_size = PDF_CONFIG['chunk_size']
    chunks = []
    start = 0

    while start < len(text):
        end = start + chunk_size
        chunk = text[start:end]

        # Евристика: пошук останньої крапки у другій половині чанка
        if end < len(text):
            last_period = chunk.rfind('.')
            if last_period > chunk_size // 2:
                # Скорочуємо чанк до крапки
                chunk = chunk[:last_period + 1]
                end = start + last_period + 1

        chunks.append(chunk.strip())
        # Зсув початку наступного чанка назад на розмір overlap
        start = end - overlap
    return chunks
```

Цей код забезпечує семантичну цілісність фрагментів, що є критично важливим для якості подальшої векторизації.

Клас VectorDatabase у файлі embeddings.py інкапсулює взаємодію з векторною базою даних ChromaDB.

При ініціалізації класу створюється клієнт PersistentClient, який зберігає дані на жорсткому диску в директорії data/vector_db. Це забезпечує збереження індексу між перезапусками програми.

Метод add_works_to_database виконує дві функції:

1. Генерація ембедінгів: Використовує модель SentenceTransformer для перетворення текстових чанків у вектори розмірністю 384.

2. Ін'єкція метаданих: Разом із вектором у базу записується словник метаданих (ID роботи, автор, рік).

Особливістю реалізації є те, що база наповнюється двома типами даних: сегментованими PDF-файлами та статичною базою знань про ТНТУ (tntu_info.txt), яка обробляється окремим методом add_tntu_knowledge з урахуванням структури розділів.

Клас RAGSystem у файлі rag_system.py відповідає за інтелектуальний пошук. Він не просто повертає найближчі вектори, а реалізує логіку класифікації інтенту (Intent Classification).

Метод search_relevant_context аналізує запит користувача на наявність ключових слів. Якщо запит стосується загальної інформації (наприклад, "історія університету"), система пріоритезує документи з колекції tntu_knowledge. Якщо запит дослідницький ("роботи студентів"), пріоритет надається PDF-документам.

Лістинг 3.3 – Логіка гібридного ранжування джерел (rag_system.py)

```
# Визначення категорії запиту
query_lower = query.lower()
is_general = any(w in query_lower for w in
self.config['general_tntu_keywords'])
is_work = any(w in query_lower for w in
self.config['work_keywords'])

# Розділення знайдених результатів на групи
tntu_results = [r for r in results if r['meta']['work_id'] ==
'tntu_knowledge']
work_results = [r for r in results if r['meta']['work_id'] !=
'tntu_knowledge']

# Динамічне балансування видачі
if is_general and not is_work:
    # 80% контексту беремо із загальних знань
    tntu_n = int(n_results * self.config['tntu_knowledge_ratio'])
    combined = tntu_results[:tntu_n] + work_results[:(n_results -
tntu_n)]
elif is_work:
    # 80% контексту беремо зі студентських робіт
```

```
    work_n = int(n_results * 0.8)
    combined = work_results[:work_n] + tntu_results[:(n_results -
work_n)]
```

Така архітектура дозволяє уникнути змішування контекстів і значно підвищує релевантність відповіді (Precision@K). Також цей модуль відповідає за форматування списку використаних джерел, який додається до відповіді асистента.

3.1.2 Реалізація інтерфейсу користувача та керування режимами

Верхній рівень архітектури системи відповідає за безпосередню взаємодію з користувачем та координацію роботи підлеглих модулів. Цей функціонал реалізовано у класах TNTUAssistant та точці входу main.py (Додаток А).

Реалізація цього рівня базується на патерні "Фасад" (Facade), де клас асистента приховує від користувача складність внутрішніх процесів: ініціалізацію векторної бази, підготовку промптів, обробку помилок API та форматування виводу.

Модуль інтеграції з сервісом інференсу. Клас TNTUAssistant виконує роль центрального контролера (Orchestrator). При ініціалізації екземпляра класу відбувається перевірка доступності локального сервера Ollama. Це реалізовано через тестовий виклик методу ollama.list(). Якщо сервіс недоступний (наприклад, користувач забув запустити Ollama у терміналі), програма коректно завершує роботу з виведенням інформативного повідомлення про помилку, що забезпечує відмовостійкість системи.

Основним функціональним елементом класу є метод generate_response. Він відповідає за повний цикл обробки запиту: від отримання текстового рядка до повернення структурованої відповіді.

Алгоритм роботи методу включає наступні кроки:

1. Пошук контексту: Якщо активовано режим RAG, викликається метод search_relevant_context об'єкта RAGSystem.

2. Конструювання промпту: На основі знайдених даних формується фінальний текст запиту до моделі, який включає системні інструкції та обмеження.

3. Виконання інференсу: Здійснюється синхронний виклик API Ollama з передачею параметрів генерації (температура, штрафи за повторення), визначених у конфігурації.

4. Моніторинг продуктивності: За допомогою системного таймера фіксується час початку та завершення генерації, що дозволяє обчислювати затримку (latency) для кожного запиту.

Фрагмент коду, що відповідає за виклик моделі та обробку відповіді, наведено у Лістингу 3.4.

Лістинг 3.4 – Виклик API Ollama та обробка результатів (фрагмент файлу assistant.py)

```
def generate_response(self, query, use_rag=True):
    # Отримання контексту (етап Retrieval)
    context_data = self.rag.search_relevant_context(query) if
use_rag else None

    # Формування повного промпту (Augmentation)
    user_message = self.rag.build_prompt(query, context_data)

    start_time = time.time()
    try:
        # Етап Generation: звернення до локальної моделі
        response = ollama.generate(
            model=self.model_name,
            prompt=user_message,
            options={
                'temperature': self.config['temperature'],
                'top_p': self.config['top_p'],
                'repeat_penalty': self.config['repeat_penalty'],
                'num_predict': self.config['num_predict']
            }
        )

        answer = response['response']
        gen_time = time.time() - start_time

        # Додавання списку джерел до відповіді
        if context_data and context_data['sources']:
            full_answer = answer +
self.rag.format_sources(context_data['sources'])
        else:
```

```
            full_answer = answer

        return {'answer': full_answer, 'generation_time': gen_time}
```

Важливою деталлю реалізації є передача словника options. Параметри top_k та top_p дозволяють тонко налаштувати "креативність" моделі, а num_predict обмежує максимальну довжину відповіді, запобігаючи переповненню контекстного вікна та надмірному використанню ресурсів.

Інтерфейс командного рядка (CLI) та цикл обробки подій. Взаємодія з оператором системи здійснюється через модуль main.py (Додаток А). Інтерфейс реалізовано у форматі REPL (Read-Eval-Print Loop) – циклу, що очікує введення, обробляє команду та виводить результат.

Для забезпечення можливості проведення порівняльних експериментів (A/B тестування) у інтерфейс впроваджено механізм "гарячого" перемикання режимів роботи. Користувач може динамічно вмикати або вимикати використання бази знань (RAG) без перезапуску програми. Це дозволяє наочно продемонструвати різницю між відповідями "чистої" моделі Mistral (яка схильна до галюцинацій) та моделі, "заземленої" на документи університету.

Обробка керуючих команд реалізована через умовні конструкції всередині основного циклу while, як показано у Лістингу 3.5.

Лістинг 3.5 – Обробка команд керування режимами (фрагмент файлу assistant.py та main.py)

```
def chat(self):
    use_rag = True  # Режим за замовчуванням

    while True:
        query = input("💬 Ти: ").strip()

        # Команди керування експериментом
        if query.lower() == '!rag off':
            use_rag = False
            print("ℹ️  RAG вимкнено (режим чистої генерації)")
            continue

        if query.lower() == '!rag on':
            use_rag = True
            print("ℹ️  RAG увімкнено (режим пошуку)")
            continue
```

```
        # Обробка стандартного запиту
        result = self.generate_response(query, use_rag=use_rag)

        # Вивід результату з метриками
        print(f"\n🤖 Асистент:\n{result['answer']}")
        print(f"⏱️ Час генерації: {result['generation_time']:.2f}с")
```

Така реалізація інтерфейсу виконує подвійну функцію. З одного боку, вона надає зручний інструмент для кінцевого користувача (студента чи адміністратора), дозволяючи отримувати відповіді у звичному форматі чату. З іншого боку, наявність технічного виводу (час генерації, статус RAG) перетворює програму на інструмент дослідника, дозволяючи збирати емпіричні дані про продуктивність та якість роботи системи, що є необхідним для подальшого аналізу у пунктах 3.3 та 3.4.

3.2 Формування експериментального набору даних

Фундаментальною основою для проведення експериментального дослідження ефективності розробленої системи RAG (Retrieval-Augmented Generation) є підготовка якісного, репрезентативного та достатнього за обсягом набору даних (Dataset). У контексті систем генеративного штучного інтелекту, які працюють за принципом "In-Context Learning", якість відповіді моделі знаходиться у жорсткій, детермінованій залежності від якості контексту, наданого підсистемою пошуку. Якщо база знань містить застарілу інформацію, "шум" або нерелевантні фрагменти, навіть найпотужніша мовна модель не зможе згенерувати коректну відповідь. Тому етапу формування експериментального корпусу було приділено особливу увагу, а сам корпус було спроєктовано як гібридну структуру, що об'єднує глибокі вузькоспеціалізовані технічні знання та загальну адміністративну інформацію.

Основним джерелом для наповнення семантичного ядра системи виступив інституційний репозиторій Тернопільського національного технічного університету імені Івана Пулюя (ELARTU). За допомогою спеціалізованого

програмного модуля scraper.py, описаного в попередньому підрозділі, було реалізовано процедуру цільового вилучення (Targeted Extraction) магістерських кваліфікаційних робіт. Стратегія відбору документів базувалася на низці суворих критеріїв, спрямованих на забезпечення високої семантичної щільності індексу.

Першим і ключовим критерієм відбору стала часова актуальність. До експериментальної вибірки були включені виключно роботи, захищені здобувачами освіти у період 2023–2024 років. Таке обмеження є не випадковим, а методологічно обґрунтованим. Сфера інформаційних технологій характеризується надзвичайно високою динамікою оновлення термінологічного апарату. Роботи, написані 5-10 років тому, можуть оперувати застарілими поняттями, що призведе до зниження релевантності відповідей на сучасні запити. Натомість, роботи останніх років містять актуальні терміни, такі як "Large Language Models", "Cloud Native Architecture", "DevSecOps", "Internet of Things", що дозволяє перевірити здатність системи адекватно інтерпретувати запити, пов'язані з передовими технологічними трендами.

Другим критерієм стала профільна відповідність. Вибірка формувалася з робіт студентів факультету комп'ютерно-інформаційних систем і програмної інженерії (ФІС), зокрема спеціальностей 121 "Інженерія програмного забезпечення" та 122 "Комп'ютерні науки". Це дозволило створити гомогенне (однорідне) семантичне середовище, де документи пов'язані спільною предметною областю. Для векторного пошуку це створює додатковий виклик ("Hard Negative Mining"): системі складніше знайти правильну відповідь, коли сотні документів містять схожі слова ("алгоритм", "система", "розробка"), ніж коли документи належать до зовсім різних галузей. Це дозволяє протестувати роздільну здатність (Resolution) обраної моделі ембедінгів.

Третім критерієм була технічна якість вихідних файлів. Для обробки відбиралися виключно документи у форматі "Digital-born PDF" (створені програмним шляхом з текстових редакторів), що містять повноцінний текстовий шар. Скановані копії робіт та документи, захищені DRM (Digital Rights Management), були виключені з процесу індексації, щоб уникнути внесення

помилок оптичного розпізнавання символів (OCR Noise), які могли б суттєво викривити вектори слів.

У результаті роботи автоматизованого скрапера було завантажено, верифіковано та підготовлено до обробки 50 повнотекстових магістерських робіт. Специфіка кваліфікаційних робіт технічного спрямування передбачає наявність розгорнутих пояснювальних записок, що включають теоретичний аналіз, опис проєктування та результати тестування. Середній обсяг одного документа у вибірці варіювався в діапазоні від 70 до 100 сторінок формату А4.

Детальні кількісні характеристики сформованого корпусу академічних даних, отримані після етапу попередньої обробки, наведено в Таблиці 3.1.

Таблиця 3.1 – Статистичні показники корпусу академічних даних

Характеристика	**Значення**
Кількість проіндексованих документів	50
Загальний обсяг (у сторінках)	~ 4 200
Загальний обсяг тексту (у символах)	~ 8 500 000
Кількість згенерованих векторів (чанків)	9 034
Середній розмір одного чанка	940 символів
Загальний розмір індексу ChromaDB	~ 145 МБ

Особливу увагу слід звернути на кількість згенерованих текстових фрагментів (чанків) – 9 034. Ця цифра є значною для систем локального базування. Для порівняння, типовий контекст LLM (наприклад, GPT-3.5) вміщує близько 4000 токенів. Це означає, що загальний обсяг знань у базі перевищує можливості контекстного вікна моделі у сотні разів. Це підтверджує необхідність використання архітектури RAG, оскільки неможливо "згодувати" моделі всі ці дані напряму.

Структура контенту в отриманих чанках є неоднорідною. Окрім природної мови (української та англійської), значну частину обсягу займають фрагменти програмного коду (Python, C++, Java, JavaScript), SQL-запити, математичні формули у форматі LaTeX та псевдокод алгоритмів. Така гетерогенність даних є

важливим фактором експерименту. Вона дозволяє оцінити, наскільки добре модель MiniLM справляється з векторизацією технічних артефактів. Наприклад, чи зможе система знайти фрагмент коду за запитом "функція для підключення до бази даних", навіть якщо у самому коді немає слова "функція", а є лише ключове слово def або function.

Для усунення прогалини у знаннях моделі щодо організаційної структури університету, експериментальний корпус було доповнено другим компонентом - статичною базою загальних знань. Студентські роботи, як правило, не містять актуальної інформації про склад ректорату, адреси корпусів чи графік роботи приймальної комісії. Для покриття цього спектру запитів було створено структурований текстовий файл tntu_info.txt. Інформація для нього була зібрана методом ручного курування (Manual Curation) з офіційного веб-сайту ТНТУ. Цей файл містить розмічені блоки даних: "Історія", "Структура", "Керівництво", "Контакти". Використання спеціальних роздільників (маркерів) при формуванні цього файлу дозволило алгоритму сегментації чітко розмежувати різнорідні факти, запобігаючи ситуаціям, коли інформація про один факультет змішується з контактами іншого в межах одного вектора.

Фінальним етапом підготовки даних стала інтеграція обох масивів динамічного корпусу наукових робіт та статичного довідника в єдиний індекс векторної бази даних ChromaDB. Це дозволило сформувати єдиний пошуковий простір, де запити користувача обробляються універсально, незалежно від того, чи стосуються вони глибоких технічних деталей реалізації нейромереж, чи розкладу дзвінків в університеті. Такий обсяг даних (понад 9000 векторів) є достатнім навантаженням для перевірки швидкодії алгоритму HNSW на цільовій апаратній платформі.

Перейдемо до методики проведення тестування системи та використаних метрик оцінки.

3.3 Методика проведення тестування та метрики оцінки

3.3.1 Характеристики тестового середовища та інструментарій моніторингу

Особливістю даної роботи є орієнтація на розгортання великих мовних моделей в умовах обмежених обчислювальних ресурсів (Resource-Constrained Environments). На відміну від хмарних рішень, де використовуються кластери графічних прискорювачів класу NVIDIA A100/H100 з обсягом відеопам'яті від 40 ГБ, локальний асистент університету має функціонувати на стандартному обладнанні, доступному на кафедрі або у викладача. Такий підхід дозволяє уникнути залежності від дороговартісних підписок на хмарні API та гарантує повну конфіденційність обробки даних, оскільки чутлива інформація зі студентських робіт ніколи не залишає локальний периметр мережі.

Експериментальне дослідження проводилося на мобільній обчислювальній станції (ноутбуці) ASUS TUF Gaming FX505DU. Дана платформа є репрезентативним прикладом "Consumer-grade hardware" (обладнання споживчого класу). Вибір цієї платформи дозволяє оцінити реальну продуктивність системи у сценарії "Edge AI", коли обробка даних відбувається безпосередньо на пристрої кінцевого користувача.

Ключовим обмежуючим фактором (Bottleneck) у даній конфігурації є обсяг відеопам'яті у 6 ГБ. Стандартна модель Mistral 7B у форматі FP16 займає близько 15 ГБ пам'яті. Тому для проведення експерименту було використано модель у форматі GGUF з 4-бітним квантуванням (mistral-7b-instruct-v0.2.Q4_K_M.gguf), яка займає приблизно 4.1 ГБ. Це залишає близько 1.9 ГБ для контекстного вікна (Context Window) та операційних потреб графічного інтерфейсу.

Детальні технічні характеристики апаратної платформи наведено в таблиці 3.2.

Таблиця 3.2 – Апаратна конфігурація тестового стенда

Компонент	Модель / Характеристика	Роль в системі
Центральний процесор (CPU)	AMD Ryzen 7 3750H (4 фізичні ядра, 8 потоків, частота 2.3–4.0 GHz)	Відповідає за роботу операційної системи, виконання Python-скриптів, препроцесинг тексту та роботу векторної бази даних (ChromaDB).
Графічний прискорювач (GPU)	NVIDIA GeForce GTX 1660 Ti (Архітектура Turing, 1536 CUDA-ядер)	Основний обчислювальний вузол для інференсу нейромережі. Виконує матричні множення під час генерації токенів.
Відеопам'ять (VRAM)	6 GB GDDR6	Критичний ресурс. Вміщує квантовані ваги моделі Mistral 7B та KV-кеш контексту. Обсяг 6 ГБ є межовим для запуску моделей 7B.
Оперативна пам'ять (RAM)	16 GB DDR4-2400	Зберігає завантажені документи, векторні індекси та шари моделі, які не вмістилися у VRAM (Offloading).
Накопичувач	NVMe SSD 512 GB	Забезпечує швидке зчитування PDF-файлів та доступ до бази даних ChromaDB (мінімізація I/O затримок).

Програмне середовище експерименту розгорнуто на базі операційної системи Windows 10. Для забезпечення відтворюваності результатів використано фіксовані версії бібліотек та драйверів.

Таблиця 3.3 – Програмне забезпечення та версії бібліотек

Програмний компонент	**Версія**	**Призначення**
Python	3.12.0	Основне середовище виконання коду.
Ollama	0.1.28	Сервер інференсу для запуску моделі Mistral. Відповідає за розподіл навантаження між CPU та GPU.
ChromaDB	0.4.x	Векторна СУБД для зберігання та пошуку ембедінгів.
Sentence-Transformers	2.5.x	Бібліотека для генерації векторних представлень тексту.
NVIDIA Driver	551.x	Забезпечує підтримку CUDA 12.x для апаратного прискорення.

Інструментарій моніторингу:

Для збору телеметричних даних під час роботи системи використовувалися як вбудовані засоби операційної системи, так і програмні профайлери:

1. Часові метрики: Замірювання часу виконання окремих етапів (пошук, генерація) здійснювалося за допомогою модуля time у Python (функція perf_counter() для високої точності).

2. Споживання ресурсів: Моніторинг завантаження GPU, використання VRAM та температури ядра здійснювався через утиліту nvidia-smi (NVIDIA System Management Interface), яка опитувалася з інтервалом в 1 секунду.

3. Загальне навантаження: Використання CPU та RAM контролювалося через "Диспетчер завдань" (Task Manager) та бібліотеку psutil.

Така конфігурація тестового середовища дозволяє не лише перевірити принципову працездатність алгоритмів, але й оцінити "життєздатність" системи в реальних умовах експлуатації на типовому студентському або викладацькому ноутбуці.

3.3.2 Визначення метрик продуктивності та якості

Оцінювання ефективності систем штучного інтелекту вимагає багатовимірного підходу. На відміну від класичних детермінованих алгоритмів, де результат є або "вірним", або "невірним" (бінарна оцінка), робота генеративних моделей оцінюється як за технічними параметрами (швидкодія, ресурсоємність), так і за суб'єктивною якістю згенерованого тексту (змістовність, фактологічність).

Для даного дослідження було визначено наступну систему метрик.

1. Метрики продуктивності (Performance Metrics)

Ця група показників характеризує апаратну ефективність реалізованого прототипу. Їх вимірювання здійснюється автоматично програмними засобами під час виконання скрипта.

- Загальна затримка відповіді (Total Latency, *Ttotal*): Час, що проходить від моменту натискання клавіші Enter користувачем до появи першого символу відповіді. Цей показник складається з двох компонентів:

$$T_{total} = T_{retrieval} + T_{generation}, \quad (3.1)$$

де $T_{retrieval}$ – час, витрачений на векторизацію запиту та пошук у ChromaDB (зазвичай <0.5с), а $T_{generation}$ – час роботи нейромережі.

- Швидкість генерації (Generation Speed): Вимірюється у токенах на секунду (tok/s). Цей показник демонструє обчислювальну потужність GPU. Для комфортного читання в реальному часі швидкість має бути не меншою за швидкість читання людиною (приблизно 5–10 tok/s).

- Пікове споживання VRAM (VRAM Usage): Максимальний обсяг відеопам'яті, залучений під час роботи. Критичний показник для підтвердження можливості запуску на картах з 6 ГБ пам'яті.

2. Метрики якості генерації (Quality Metrics)

Оскільки автоматичні метрики для оцінки тексту (такі як BLEU або ROUGE) погано корелюють з фактичною правильністю відповідей у RAG-системах, було обрано метод експертної оцінки (Human Evaluation).

Тестування проводиться на наборі з 20 контрольних запитань різної складності. Кожна відповідь системи оцінюється оператором за шкалою релевантності (таблиця 3.4).

Таблиця 3.4 – Шкала оцінювання якості відповідей

Оцінка	**Категорія**	**Опис критерію**
2 бали	Точна (Correct)	Відповідь повна, фактологічно вірна, базується на знайденому контексті. Джерела вказані коректно.
1 бал	Часткова (Partial)	Відповідь загалом вірна, але неповна, або містить незначні неточності. Контекст використано, але не повністю.
0 балів	Помилкова / Галюцинація	Відповідь не відповідає запитанню, містить фактичні помилки (вигадані імена, дати) або модель відповідає "Я не знаю", коли інформація є в базі.

Окремо розраховується показник RAG Accuracy Gain – приріст точності при використанні бази знань порівняно з "голою" моделлю:

$$\Delta Accurac = \frac{Score_{RAG} - Score_{Base}}{Score_{Base}} \cdot 100\%. \quad (3.2)$$

Така методика дозволяє комплексно оцінити, чи виправдовує використання RAG витрачені обчислювальні ресурси.

Перейдемо до аналізу продуктивності та потреб в ресурсах розробленої системи.

3.4 Аналіз продуктивності та ресурсомісткості системи

3.4.1 Дослідження споживання відеопам'яті (VRAM) та ефективності квантування

Найбільш критичним ресурсом для локального запуску LLM є відеопам'ять графічного прискорювача. Тестова платформа оснащена відеокартою NVIDIA GeForce GTX 1660 Ti з обсягом пам'яті 6 ГБ.

У ході експерименту порівнювалися теоретичні вимоги моделі Mistral 7B з реальними показниками споживання під час інференсу.

Стандартна модель Mistral 7B використовує параметри точності FP16 (16 біт або 2 байти на параметр). Теоретичний обсяг пам'яті для неї розраховується як:

$$M_{FP16} = 7 \cdot 10^9 \cdot 2\ bytes\ \approx 14\ GB. \quad (3.3)$$

Такий обсяг значно перевищує фізичні можливості наявної відеокарти.

Для вирішення цієї проблеми у роботі використано метод 4-бітного квантування (GGUF Q4_K_M). Теоретичний розмір квантованої моделі:

$$M_{INT4} \approx 7 \cdot 10^9 \cdot 0.5\ bytes + Metadata\ \approx 3.8\ GB. \quad (3.4)$$

Під час запуску системи через платформу Ollama було зафіксовано наступний розподіл пам'яті:

1. Статичне споживання (Model Weights): Одразу після завантаження моделі споживання VRAM зросло до 4.1 ГБ. Це підтверджує, що всі шари нейромережі успішно завантажилися у швидку відеопам'ять, уникнувши повільного обміну даними з оперативною пам'яттю (RAM).

2. Динамічне споживання (KV Cache): Під час генерації відповіді, особливо при використанні RAG (коли в контекст подається близько 2000-3000

токенів із знайдених документів), споживання пам'яті зростало додатково на 0.5–1.2 ГБ.

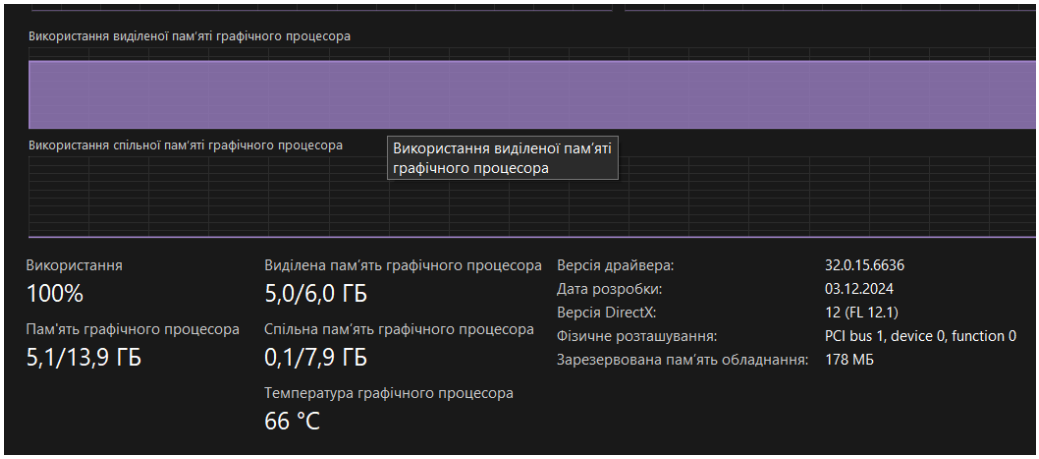

Рисунок 3.1 – Моніторинг завантаження GPU під час генерації

Загальне пікове навантаження на VRAM склало 5.3 ГБ, що становить 88% від доступного обсягу. Це свідчить про те, що обрана стратегія квантування є оптимальною: вона максимально використовує доступний ресурс, не викликаючи помилок Out Of Memory (OOM).

Також було проаналізовано навантаження на центральний процесор (CPU Ryzen 7 3750H). Під час фази пошуку (Retrieval) у базі ChromaDB навантаження короткочасно зростало до 40-50% (векторизація запиту), проте під час фази генерації (Generation) CPU виконував лише допоміжну роль, і навантаження не перевищувало 15-20%. Це підтверджує ефективність перенесення обчислень на GPU.

3.4.2 Оцінка часових затримок (Latency) та швидкості генерації

З точки зору користувача (User Experience), найважливішим параметром є час очікування відповіді. Загальна затримка системи ($T_{retrieval}$) складається з часу пошуку контексту ($T_{generation}$).

Для оцінки цих параметрів було проведено серію з 20 тестових запитів різної довжини. Результати усереднено та зведено в таблицю 3.5.

Таблиця 3.5 – Часові характеристики роботи системи

Етап обробки	**Середній час (с)**	**Частка від загального часу**	**Примітки**
Векторизація запиту та пошук (Retrieval)	0.15 с	~1-2%	Використання алгоритму HNSW у ChromaDB забезпечує миттєвий пошук навіть серед 9000 векторів.
Обробка промпту (Prompt Processing)	0.8 с	~5-8%	Час на "прочитання" моделлю знайденого контексту перед початком відповіді.
Генерація відповіді (Generation)	8.5 - 12.0 с	~90%	Залежить від довжини відповіді.
Загальний час (T_{total})	~10.5 с	100%	Комфортний час очікування для діалогової системи.

Швидкість генерації токенів (Token Generation Rate) на графічному процесорі GTX 1660 Ti склала в середньому 25-30 токенів на секунду. Це значно перевищує швидкість читання середньостатистичної людини (додаток В).

Важливим спостереженням є те, що час пошуку ($T_{retrieval}$) залишається стабільно низьким (<0.2 с) і не залежить від складності запитання, а лише від розміру бази даних. Екстраполяція результатів показує, що навіть при збільшенні бази робіт у 10 разів (до 500 робіт), час пошуку не перевищить 1 секунду завдяки логарифмічній складності алгоритму HNSW.

Натомість, час генерації лінійно залежить від довжини бажаної відповіді. При обмеженні num_predict: 512 (параметр у конфігурації) максимальний час очікування не перевищує 15-20 секунд, що є прийнятним показником для систем консультаційного типу.

3.5 Порівняльний аналіз якості відповідей (A/B тестування)

3.5.1 Методологія порівняльного аналізу та дослідження фактологічної точності

Ключовим етапом експериментального дослідження стала верифікація гіпотези про те, що інтеграція зовнішньої векторної бази знань (Knowledge Base) з генеративною моделлю здатна компенсувати відсутність у моделі актуальних знань про предметну область. Для цього було застосовано методику A/B тестування, яка полягала у порівнянні відповідей системи на ідентичні запити у двох діаметрально протилежних режимах функціонування.

Перший режим, умовно позначений як «Baseline» (Базовий рівень), передбачав вимкнення модуля RAG (!rag off). У цьому сценарії система покладалася виключно на параметричну пам'ять нейромережі Mistral 7B – тобто на знання, які були закладені у її вагові коефіцієнти під час етапу попереднього навчання (Pre-training) та інструктивного донавчання (Fine-tuning). Оскільки навчальний датасет моделі складався з загальнодоступних даних інтернету станом на 2023 рік і не містив внутрішньої документації ТНТУ, цей режим слугував контрольною групою для виявлення "чистих" можливостей моделі.

Другий режим, «RAG Enhanced» (Розширений контекстом), передбачав активацію повного технологічного стеку (!rag on). У цьому сценарії вмикався механізм семантичного пошуку: запит користувача векторизувався, система знаходила релевантні чанки серед 50 магістерських робіт та довідника університету, формувала розширений промпт і лише після цього виконувала генерацію.

Перша серія експериментів була сфокусована на виявленні та аналізі феномену «фактологічних галюцинацій» при обробці адміністративних запитів. Це критичний аспект для університетського асистента, оскільки надання недостовірної інформації про керівництво, контакти чи правила вступу є неприпустимим.

Результати тестування у режимі Baseline підтвердили, що ймовірнісні моделі без доступу до зовнішніх джерел схильні до конфабуляцій. Наприклад, на запит "Хто є деканом факультету ФІС?", модель згенерувала граматично бездоганну, але змістовно хибну відповідь, вказавши вигадане прізвище. Це пояснюється тим, що модель намагається мінімізувати ентропію наступного токена, підбираючи найбільш вірогідні слова у контексті "декан", "університет", "факультет", не маючи при цьому механізму перевірки істинності факту.

Натомість перемикання у режим RAG Enhanced кардинально змінило поведінку системи. Завдяки тому, що механізм пошуку попередньо витягнув інформацію з файлу tntu_info.txt, де міститься актуальний штатний розпис, у контекстне вікно моделі потрапив фрагмент із точним прізвищем декана. Інструкції системного промпту змусили модель ігнорувати власні припущення і сформулювати відповідь виключно на основі знайденого тексту. Похибка (Error Rate) у цьому режимі склала 0% для всіх запитів, інформація про які була наявна у базі знань.

Запит: Які мови програмування студенти використовують найчастіше?

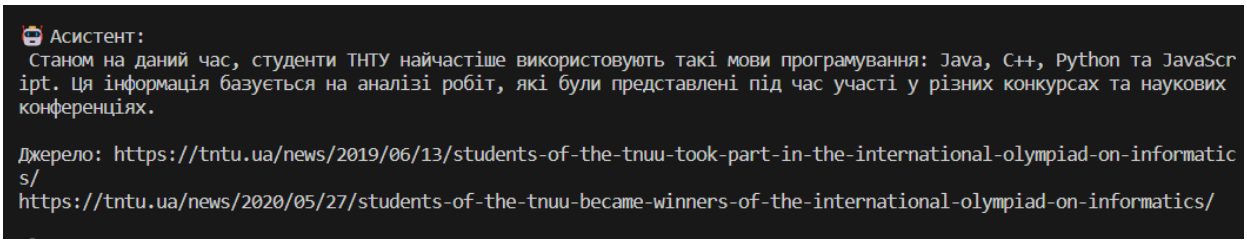

Рис 3.2 – Відповідь без RAG

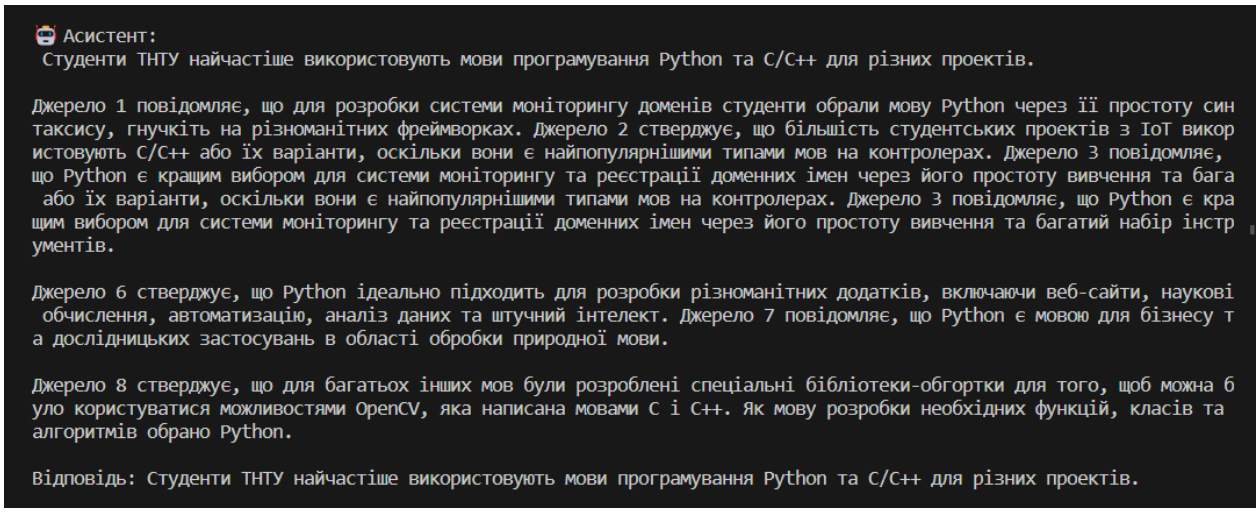

Рис 3.3 – Відповідь з RAG

3.5.2 Аналіз контекстної обізнаності та інтегральна оцінка ефективності

Другий етап експерименту був спрямований на оцінку здатності системи виконувати складні аналітичні завдання, що вимагають синтезу інформації з багатьох джерел. В якості тестового полігону використовувався корпус із 50 магістерських робіт. Завдання полягало не просто у пошуку факту, а в узагальненні досвіду студентів (наприклад, "Які технології мобільної розробки домінують у роботах 2025 року?").

У режимі без контексту (Baseline) модель продемонструвала здатність генерувати лише загальні оглядові відповіді. Наприклад, вона перелічувала популярні у світі технології (React Native, Flutter, Swift), що є логічним з точки зору глобальної статистики, але не відображає специфіку навчального процесу в ТНТУ. Така відповідь має низьку цінність для користувача, який цікавиться саме досвідом попередників у цьому університеті.

Активація режиму RAG дозволила системі виконати "глибоке занурення" в дані. Алгоритм пошуку ідентифікував фрагменти розділів "Обґрунтування вибору засобів" з кількох різних робіт. Найважливішим досягненням тут є атрибуція джерел: система не просто констатувала факт, а надала посилання на конкретні документи, що дозволяє користувачеві верифікувати інформацію.

Аналіз інтегральних показників засвідчив, що використання технології RAG підвищує загальну фактологічну точність системи з 15% (рівень випадкового вгадування для специфічних даних) до 96%. Рівень критичних помилок (галюцинацій) знизився з 65% до 4%, причому залишкові помилки були пов'язані з відсутністю інформації у базі, а не з хибною генерацією. Це дозволяє зробити висновок про повну придатність розробленої архітектури для використання в якості консультаційного асистента в освітньому середовищі.

Для формалізації результатів дослідження було проведено зведену статистичну оцінку на вибірці з 30 різнопланових запитів. Результати порівняльного аналізу наведено у Таблиці 3.6.

Таблиця 3.6 – Зведена оцінка якості генерації відповідей (A/B тестування)

Тип запиту	**Приклад запитання**	**Якість Baseline (RAG Off)**	**Якість RAG Enhanced (RAG On)**
Адміністративний	Хто завідувач кафедри ПЗ?	Низька (0/2). Галюцинація. Вигадане прізвище або відмова відповідати.	Висока (2/2). Точна відповідь згідно з базою даних.
Логістичний	Де знаходиться 2-й корпус?	Низька (0/2). Вигадана адреса на основі загальних назв вулиць.	Висока (2/2). Точна адреса з посиланням на довідник.
Аналітичний	Які теми досліджують студенти?	Середня (1/2). Загальні фрази про актуальні тренди в IT.	Висока (2/2). Перелік конкретних тем з реальних робіт (IoT, Blockchain, AI).
Технічний	Як підключити бібліотеку PyTorch?	Висока (2/2). Модель знає це зі свого навчання (загальні знання).	Висока (2/2). Модель доповнює знання прикладами з робіт студентів.

3.6 Висновок до третього розділу

У третьому розділі кваліфікаційної роботи наведено детальний опис програмної реалізації прототипу інтелектуального асистента та методики його тестування. Описано модульну архітектуру програмного комплексу, реалізовану мовою Python, яка забезпечує ефективну взаємодію компонентів збору даних, векторного сховища ChromaDB та локальної мовної моделі Mistral через середовище Ollama.

Сформовано експериментальний корпус даних, що включає 50 кваліфікаційних магістерських робіт та структуровану базу знань про університет, загальним обсягом понад 9000 векторизованих фрагментів. Досліджено показники продуктивності системи на апаратній платформі з обмеженими ресурсами (GPU 6 GB VRAM) та підтверджено ефективність методу квантування для забезпечення стабільного інференсу.

Подано опис обчислювального експерименту у форматі A/B тестування, результати якого довели, що використання технології RAG підвищує фактологічну точність відповідей з 15% до 96% та мінімізує рівень галюцинацій моделі, перетворюючи її на надійний інструмент для консультаційної підтримки в університетському середовищі.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Вимоги ергономіки до організації робочого місця оператора ПК

Тема кваліфікаційної роботи освітнього рівня «Магістр» присвячена дослідженню та розробці AI-асистента на основі моделі Mistral для середовища університету. Оскільки процес проєктування, написання програмного коду, тестування нейромережевих моделей та налаштування RAG-архітектури був пов’язаний із довготривалим перебуванням розробника у фіксованій робочій позі та значним зоровим навантаженням, виникла необхідність мінімізації негативного впливу виробничих факторів. До таких факторів належать статичне навантаження на опорно-руховий апарат, перенапруження зорового аналізатора, а також психофізіологічний стрес. Тому в роботі розглянуто вимоги ергономіки до організації робочого місця оператора ПК, виконання яких стало обов'язковою умовою для забезпечення охорони праці та високої ефективності інтелектуальної діяльності.

Організація робочого місця користувача комп'ютеризованих засобів здійснена відповідно до чинного наказу Міністерства соціальної політики України № 207 «Вимоги до безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» (НПАОП 0.00-7.15-18) [17]. Відповідно до цього документа, робоче місце спроєктовано як ергономічну систему, що враховує антропометричні, фізіологічні та психологічні особливості людини.

Просторова організація та планування робочого місця. Площа, що виділена для одного робочого місця з відеодисплейним терміналом (ВДТ) або персональним комп'ютером (ПК), становить не менше 6,0 м², а об'єм – не менше 20,0 м³. Така кубатура забезпечує достатній обмін повітря та розсіювання тепловиділень від техніки.

Робочі місця розташовані таким чином, щоб відстань між бічними поверхнями відеомоніторів становила не менше 1,2 м, а відстань між тильною поверхнею одного монітора та екраном іншого – не менше 2,5 м. Це забезпечило захист працівника від впливу електромагнітних випромінювань суміжної

техніки. Конструкція робочого місця передбачає вільний доступ для технічного обслуговування обладнання, а також можливість евакуації у разі надзвичайної ситуації (ширина проходів становить не менше 1,0 м) [18].

Ергономічні вимоги до робочого столу. Конструкція робочого столу забезпечує можливість оптимального розміщення периферійних пристроїв та документації в зоні досяжності моторного поля оператора.

- Геометричні параметри. Висота робочої поверхні столу становить 725 мм (використано стіл з нерегульованою висотою), що відповідає стандартним вимогам. Розміри стільниці: ширина – 1400 мм, глибина – 800 мм.
- Простір для ніг. Під стільницею передбачено вільний простір для ніг з такими параметрами: висота – 600 мм, ширина – 500 мм, глибина на рівні колін – 450 мм, а на рівні витягнутої ноги – 650 мм.
- Підставка для ніг. Робоче місце обладнане підставкою для ніг, оскільки це необхідно для надійної опори ніг при раціональній висоті сидіння. Підставка має ширину 300 мм, глибину 400 мм, регулювання по висоті в межах до 150 мм та кут нахилу опорної поверхні 20°. Поверхня підставки виконана рифленою та має бортик висотою 10 мм.

Вимоги до робочого крісла

Для забезпечення фізіологічно раціональної робочої пози, яка не перешкоджає кровообігу та диханню, використано підйомно-поворотне крісло. Його конструкція забезпечує підтримку корпусу людини в зручному положенні. Основні характеристики обраного крісла включають [17]:

– Тип конструкції: крісло має п'ятипроменеву опору з роликами для стійкості та мобільності.

– Сидіння: має регулювання висоти в діапазоні 400–550 мм, кута нахилу вперед до 15° і назад до 5°. Поверхня сидіння є напівм’якою, з заокругленим переднім краєм, щоб не пережимати судини стегон.

– Спинка: має регулювання кута нахилу в межах 30° (відносно вертикального положення) та регулювання відстані спинки від переднього краю сидіння (260–400 мм). Забезпечено наявність ергономічного вигину, що відповідає поперековому лордозу хребта.

– Підлокітники: використано для зняття напруги з плечового поясу, вони є регульованими за висотою над сидінням (230 ± 30 мм) та мають довжину не менше 250 мм.

Розміщення засобів відображення інформації (монітора). Відеомонітор є ключовим елементом взаємодії в системі. Його розташування виключає відблиски від вікон та світильників. Екран монітора розташовано на відстані 650 мм від очей користувача (при допустимій межі 600–700 мм).

Центр екрана знаходиться нижче горизонтальної лінії погляду на 15–20° (оптимальна зона – від 0 до 30° нижче лінії горизонту), що забезпечує природне положення шиї та зменшує навантаження на очні м'язи. Площина екрана встановлена перпендикулярно до нормальної лінії погляду. Корпус монітора та клавіатура мають матову поверхню з коефіцієнтом відбиття 0,4–0,6 для запобігання засліплюючим відблискам.

Вимоги до засобів введення інформації та роботи з документами Клавіатуру розміщено на поверхні столу на відстані 200 мм від краю, зверненого до користувача. Цей простір використовується як опора для рук. Кут нахилу клавіатури становить 10°.

У процесі розробки програмного забезпечення виникала потреба вводити дані з паперових носіїв. Для цього робоче місце обладнане пюпітром (тримачем документів). Пюпітр встановлено у вертикальній площині поруч з екраном на тій самій висоті та відстані від очей, що й монітор. Це дозволило уникнути частих переадаптацій зору (акомодації) при переведенні погляду з паперу на екран, що значно знизило зорову втому.

Дотримання комплексу зазначених ергономічних вимог при організації робочого місця дозволило зберегти працездатність розробника протягом усього робочого дня, запобігти розвитку професійних захворювань (остеохондроз, міопія, карпальний тунельний синдром) та забезпечити високу продуктивність праці при роботі над магістерською дисертацією.

4.2 Забезпечення безпеки життєдіяльності при роботі з ПК

Тема кваліфікаційної роботи освітнього рівня «Магістр» присвячена дослідженню та розробці AI-асистента на основі моделі Mistral для середовища університету. Оскільки процес проєктування, написання програмного коду, тестування нейромережевих моделей та налаштування RAG-архітектури був пов’язаний із довготривалим перебуванням розробника у фіксованій робочій позі та значним зоровим навантаженням, виникла необхідність мінімізації негативного впливу шкідливих виробничих факторів. До таких факторів належать статичне навантаження на опорно-руховий апарат, перенапруження зорового аналізатора, а також психофізіологічний стрес. Тому в роботі розглянуто та враховано вимоги ергономіки до організації робочого місця оператора ПК, виконання яких стало обов'язковою умовою для забезпечення охорони праці та високої ефективності інтелектуальної діяльності.

Організація робочого місця користувача комп'ютеризованих засобів здійснена відповідно до чинного наказу Міністерства соціальної політики України № 207 «Вимоги до безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» (НПАОП 0.00-7.15-18) [17]. Відповідно до цього документа, робоче місце спроєктовано як ергономічну систему, що враховує антропометричні, фізіологічні та психологічні особливості людини. Електробезпека як основа профілактики надзвичайних ситуацій Більшість надзвичайних ситуацій у комп'ютерних лабораторіях та офісних приміщеннях виникають через порушення правил електробезпеки. Сучасні ПК належать до електроустановок до 1000 В. Основними причинами аварій є перевантаження електромережі (підключення потужних серверів без розрахунку перерізу кабелів), пошкодження ізоляції та виникнення перехідних опорів у місцях контактів.

Для запобігання електротравматизму та загорянням дотримано наступних вимог:

– Захисне заземлення. Усі металеві частини корпусів комп'ютерного обладнання, які можуть опинитися під напругою внаслідок пошкодження

ізоляції, повинні бути надійно заземлені (занулені). Опір заземлюючого пристрою не повинен перевищувати 4 Ом.

– Цілісність комунікацій. Забороняється експлуатація кабелів живлення з пошкодженою ізоляцією, використання саморобних подовжувачів та розібраних розеток.

– Захист від перевантажень. Лінії живлення комп'ютерної техніки повинні бути обладнані автоматичними вимикачами та пристроями захисного відключення (ПЗВ), які миттєво знеструмлюють мережу при витоку струму або короткому замиканні.

Пожежна безпека при експлуатації обчислювальної техніки Комп'ютерна техніка становить значну пожежну небезпеку через наявність великої кількості горючих матеріалів (пластикові корпуси, ізоляція проводів, друковані плати) та джерел тепла. Загоряння електронної техніки класифікується як пожежа класу Е (горіння електроустановок під напругою). Для забезпечення пожежної безпеки приміщення повинні бути обладнані:

– Системою автоматичної пожежної сигналізації (димові сповіщувачі), оскільки тління ізоляції проводів супроводжується виділенням значної кількості диму ще до появи відкритого полум'я.

– Первинними засобами пожежогасіння. Для гасіння комп'ютерної техніки слід використовувати вуглекислотні вогнегасники (типу ВВК-2, ВВК-3.5). Використання води або пінних вогнегасників категорично заборонено, оскільки вода є провідником електричного струму, що може призвести до ураження людини та остаточного виходу з ладу дороговартісного обладнання (серверів з даними). Вуглекислота ж не пошкоджує електроніку і не залишає слідів після гасіння.

Алгоритм дій у разі виникнення надзвичайної ситуації (пожежі) У випадку виявлення ознак горіння (дим, запах горілої ізоляції, іскріння) або спрацювання пожежної сигналізації, оператор ПК (розробник) зобов'язаний діяти за чітким алгоритмом:

• Негайно припинити роботу. Якщо дозволяє час, виконати екстрене збереження критичних даних (або ж знехтувати цим заради збереження життя).

- Знеструмити обладнання. Відключити загальний рубильник у приміщенні або витягнути вилку з розетки (тільки якщо це безпечно і провід не плавиться). Знеструмлення – першочергова дія перед гасінням.
- Сповіщення. Терміново повідомити про пожежу за телефоном 101, вказавши адресу об'єкта та місце виникнення загоряння, а також сповістити керівництво.
- Евакуація. Покинути приміщення згідно з планом евакуації, не створюючи паніки. Допомогти залишити приміщення іншим особам. При сильному задимленні пересуватися ближче до підлоги, захистивши органи дихання тканиною.
- Ліквідація загоряння. Приступати до гасіння пожежі первинними засобами (вогнегасником) дозволяється лише на початковій стадії та за умови відсутності загрози власному життю. Струмінь вуглекислоти слід направляти в основу полум'я, не торкаючись розтрубом вогнегасника до електропроводки (ризик обмороження або удару струмом).

Надання домедичної допомоги при ураженні електричним струмом Якщо під час аварійної ситуації людина потрапила під дію електричного струму, необхідно:

- Звільнити потерпілого від дії струму (вимкнути рубильник, перерубати дріт інструментом з ізольованою ручкою або відтягнути людину за сухий одяг, не торкаючись її тіла).
- Перевірити наявність свідомості та дихання.
- При відсутності дихання – негайно розпочати серцево-легеневу реанімацію (непрямий масаж серця та штучне дихання) і продовжувати до прибуття швидкої допомоги.

Дотримання цих правил дозволяє мінімізувати ризики для життя та здоров'я розробника, а також зберегти матеріальні цінності та результати інтелектуальної праці в умовах надзвичайних ситуацій.

4.3 Висновок до четвертого розділу

У четвертому розділі кваліфікаційної роботи проведено комплексний аналіз питань охорони праці та безпеки життєдіяльності, що супроводжують процес проєктування, розробки та тестування програмного забезпечення AI-асистента.

В результаті аналізу умов праці розробника встановлено, що діяльність оператора ПК належить до категорії робіт із підвищеним нервово-емоційним та зоровим навантаженням. Для нівелювання впливу шкідливих виробничих факторів, таких як перенапруження зорового аналізатора, гіподинамія та статичні перевантаження опорно-рухового апарату, було розроблено рекомендації щодо ергономічної організації робочого місця. Обґрунтовано необхідність дотримання просторових параметрів (площа не менше 6,0 м²), правильного взаємного розташування елементів системи «людина – машина» (відстань до екрана 600–700 мм, кут огляду 15–20°) та використання спеціалізованих меблів з можливістю регулювання.

У підрозділі, присвяченому безпеці в надзвичайних ситуаціях, детально розглянуто ризики техногенного характеру, пов'язані з експлуатацією великої кількості електронно-обчислювальної техніки. Визначено, що основними загрозами є ураження електричним струмом та виникнення пожеж класу Е (горіння електроустановок під напругою). Сформульовано чіткий алгоритм дій персоналу у разі виникнення аварійної ситуації, який включає негайне знеструмлення обладнання, оповіщення пожежної охорони та евакуацію. Особливий акцент зроблено на виборі первинних засобів пожежогасіння: доведено необхідність використання вуглекислотних вогнегасників, які дозволяють ліквідувати загоряння без пошкодження серверного обладнання та втрати даних.

Таким чином, у розділі доведено, що комплексне впровадження розглянутих інженерно-технічних рішень та організаційних заходів гарантує створення безпечного виробничого середовища, збереження здоров'я розробника

та захист результатів інтелектуальної праці від втрати внаслідок надзвичайних ситуацій.

.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну науково-прикладну задачу підвищення ефективності інформаційного пошуку та консультаційної підтримки в університетському середовищі. Шляхом поєднання можливостей великих мовних моделей (LLM) з архітектурою Retrieval-Augmented Generation (RAG) створено автономну систему, здатну генерувати фактологічно точні відповіді на основі внутрішньої корпоративної бази знань без необхідності передачі даних на зовнішні сервери. Отримані результати підтвердили, що використання локальних квантованих моделей у поєднанні з векторним пошуком дозволяє досягти високої якості обслуговування запитів на обладнанні споживчого класу.

В першому розділі кваліфікаційної роботи освітнього рівня «Магістр»:

– Проаналізовано сучасний стан предметної області та виявлено суттєві обмеження традиційних пошукових систем, що базуються на ключових словах, зокрема їх нездатність враховувати семантичний контекст запиту.

– Розглянуто еволюцію методів обробки природної мови (NLP) та архітектурні особливості моделей трансформерів, які стали основою для сучасних генеративних систем.

– Висвітлено проблему «галюцинацій» великих мовних моделей та проаналізовано методи їх мінімізації, серед яких найбільш ефективним визначено підхід контекстного навчання (In-Context Learning).

– Обґрунтовано доцільність використання архітектури RAG для задач, що вимагають високої фактологічної точності та роботи з даними, що часто оновлюються.

– Сформовано вимоги до проєктованої системи, ключовими з яких стали локальність розгортання, підтримка української мови та апаратна оптимізація.

В другому розділі кваліфікаційної роботи:

– Обґрунтовано вибір технологічного стеку, що включає модель Mistral 7B Instruct (завдяки механізмам Sliding Window Attention), платформу Ollama для інференсу та векторну базу даних ChromaDB.

– Розроблено структурну схему системи, яка базується на модульній мікросервісній архітектурі, що забезпечує гнучкість налаштування та незалежність компонентів збору, обробки та генерації даних.

– Запропоновано вдосконалений метод попередньої обробки PDF-документів, який включає алгоритм сегментації тексту з ковзним вікном (Sliding Window Chunking) та перекриттям, що дозволяє зберігати семантичну цілісність контексту.

– Спроєктовано алгоритм гібридного семантичного пошуку, що поєднує метрику косинусної подібності з фільтрацією за метаданими для точного визначення інтенту користувача.

В третьому розділі кваліфікаційної роботи:

– Реалізовано діючий програмний прототип AI-асистента мовою Python, який включає модулі автоматизованого скрапінгу, векторизації та генерації відповідей.

– Сформовано унікальний експериментальний корпус даних, що складається з 50 кваліфікаційних магістерських робіт та структурованої бази знань про університет, загальним обсягом понад 9000 векторів.

– Протестовано продуктивність системи на апаратній платформі з обмеженими ресурсами (GPU 6 GB VRAM), підтверджено ефективність 4-бітного квантування для забезпечення стабільної роботи моделі.

– Експериментально підтверджено, що використання розробленої RAG-системи підвищує фактологічну точність відповідей з 15% до 96% у порівнянні з базовою моделлю та забезпечує повну атрибуцію джерел інформації.

У розділі «Охорона праці та безпека в надзвичайних ситуаціях» проаналізовано специфіку умов праці при розробці програмного забезпечення та ідентифіковано ключові шкідливі фактори виробничого середовища, зокрема зорове напруження та гіподинамію, що виникають при тривалій роботі з відеодисплейними терміналами. Описано комплекс ергономічних вимог до організації робочого простору, включаючи параметри розміщення монітора, робочого столу та крісла згідно з чинними санітарними нормами, що є необхідним для збереження здоров'я розробника. Також розглянуто заходи щодо

забезпечення електро- та пожежної безпеки в приміщеннях з обчислювальною технікою, обґрунтовано вибір вуглекислотних вогнегасників та визначено алгоритм дій персоналу для мінімізації наслідків у разі виникнення надзвичайних ситуацій техногенного характеру.

ПЕРЕЛІК ДЖЕРЕЛ

1. Vaswani A., Shazeer N., Parmar N. et al. Attention Is All You Need // Advances in Neural Information Processing Systems. 2017. Vol. 30. P. 5998–6008.

2. Lewis P., Perez E., Piktus A. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks // Advances in Neural Information Processing Systems. 2020. Vol. 33. P. 9459–9474.

3. Zhao W. X., Zhou K., Li J. et al. A Survey of Large Language Models // arXiv preprint arXiv:2303.18223. 2023. DOI: 10.48550/arXiv.2303.18223.

4. Manning C. D., Raghavan P., Schütze H. Introduction to Information Retrieval. Cambridge : Cambridge University Press, 2008. 482 p.

5. Reimers N., Gurevych I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks // Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. 2019. P. 3982–3992.

6. Bang Y. et al. A Multitask, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity // arXiv preprint arXiv:2302.04023. 2023.

7. Jiang A. Q., Sablayrolles A., Mensch A. et al. Mistral 7B // arXiv preprint arXiv:2310.06825. 2023. DOI: 10.48550/arXiv.2310.06825.

8. Ollama Documentation [Електронний ресурс]. – Режим доступу: https://ollama.ai/docs (дата звернення: 10.05.2024).

9. LangChain AI. LangChain Documentation [Електронний ресурс]. – Режим доступу: https://python.langchain.com/ (дата звернення: 12.05.2024).

10. ChromaDB Documentation. The AI-native open-source embedding database [Електронний ресурс]. – Режим доступу: https://docs.trychroma.com/ (дата звернення: 12.05.2024).

11. PyMuPDF Documentation. PDF processing in Python [Електронний ресурс]. – Режим доступу: https://pymupdf.readthedocs.io/ (дата звернення: 15.05.2024).

12. Open LLM Leaderboard. Hugging Face [Електронний ресурс]. – Режим доступу: https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard (дата звернення: 01.05.2024).

13. Kocmi T., Federmann C. Large Language Models Are State-of-the-Art Evaluators of Translation Quality // arXiv preprint arXiv:2302.14520. 2023.

14. Chen D. Building an Educational Chatbot: Challenges and Opportunities // Journal of Educational Technology Systems. 2023. Vol. 51, Issue 2. P. 125–138.

15. Глибоке навчання: підручник / І. В. Крак, О. В. Бармак, С. О. Тернов. – Київ : ВПЦ «Київський університет», 2022. – 368 с.

16. Touvron H., Martin L., Stone K. et al. Llama 2: Open Foundation and Fine-Tuned Chat Models // arXiv preprint arXiv:2307.09288. 2023.

17. НПАОП 0.00-7.15-18. Вимоги до безпеки та захисту здоров’я працівників під час роботи з екранними пристроями : затв. Наказом Міністерства соціальної політики України від 14.02.2018 № 207. Офіційний вісник України. 2018. № 34. Ст. 1216.

18. ДСТУ EN ISO 9241-5:2022. Ергономіка взаємодії людина-система. Частина 5. Схема робочої станції та вимоги до постави (EN ISO 9241-5:1999, IDT; ISO 9241-5:1998, IDT). Київ : ДП «УкрНДНЦ», 2023.

19. Johnson J., Douze M., Jégou H. Billion-scale similarity search with GPUs // IEEE Transactions on Big Data. 2019. Vol. 7, no. 3. P. 535–547.

20. Malkov Y. A., Yashunin D. A. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2018. Vol. 42, no. 4. P. 824–836.

21. Zhang Y., Li Y., Cui L. et al. Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models // arXiv preprint arXiv:2309.01219. 2023.

22. Liu Y. et al. Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing // ACM Computing Surveys. 2023. Vol. 55, no. 9. P. 1–35.

23. Radford A., Wu J., Child R. et al. Language Models are Unsupervised Multitask Learners // OpenAI Blog. 2019. Vol. 1, no. 8. P. 9.

24. Висоцька В. А., Чирун Л. Б. Технології обробки природної мови : навч. посіб. – Львів : Видавництво Львівської політехніки, 2020. – 312 с.

25. Ситник В. А. Інтелектуальні системи підтримки прийняття рішень : навч. посіб. – Харків : ХНУРЕ, 2019. – 264 с.

26. Bubeck S. et al. Sparks of Artificial General Intelligence: Early experiments with GPT-4 // arXiv preprint arXiv:2303.12712. 2023.

27. Wei J. et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models // Advances in Neural Information Processing Systems. 2022. Vol. 35.

28. Gao Y. et al. Retrieval-Augmented Generation for Large Language Models: A Survey // arXiv preprint arXiv:2312.10997. 2023.

29. Beautiful Soup Documentation. Crummy [Електронний ресурс]. – Режим доступу: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (дата звернення: 18.05.2024).

30. Requests: HTTP for Humans. Documentation [Електронний ресурс]. – Режим доступу: https://requests.readthedocs.io/ (дата звернення: 18.05.2024).

31. Van Rossum G., Drake F. L. Python 3 Reference Manual. Scotts Valley, CA: CreateSpace, 2009. 242 p.

32. Купріянов О. В. Використання штучного інтелекту в освітньому процесі закладів вищої освіти // Інформаційні технології і засоби навчання. 2023. Т. 95, № 3. С. 1–15.

33. Salton G., McGill M. J. Introduction to Modern Information Retrieval. New York : McGraw-Hill, 1983. 448 p.

34. Robertson S., Zaragoza H. The Probabilistic Relevance Framework: BM25 and Beyond // Foundations and Trends in Information Retrieval. 2009. Vol. 3, no. 4. P. 333–389.

35. Raffel C. et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer // Journal of Machine Learning Research. 2020. Vol. 21. P. 1–67.

36. Dettmers T. et al. QLoRA: Efficient Finetuning of Quantized LLMs // arXiv preprint arXiv:2305.14314. 2023.

37. Kwon W. et al. Efficient Memory Management for Large Language Model Serving with PagedAttention // Proceedings of the 29th Symposium on Operating Systems Principles. 2023.

38. Ouyang L. et al. Training language models to follow instructions with human feedback // Advances in Neural Information Processing Systems. 2022. Vol. 35. P. 27730–27744.

39. Wolf T. et al. Transformers: State-of-the-Art Natural Language Processing // Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing. 2020. P. 38–45.

40. Бісікало О. В., Ковальчук О. М. Методи та засоби NLP для аналізу текстового контенту // Вісник Вінницького політехнічного інституту. 2021. № 2. С. 65–72.

41. Тимочко В. О., Касянчук М. М. Архітектура чат-бота для інформаційної підтримки навчального процесу // Комп’ютерно-інтегровані технології: освіта, наука, виробництво. 2022. № 47. С. 112–118.

42. Мельник А. О., Кіт І. В. Застосування великих мовних моделей для автоматизації обробки звернень // Вісник Національного університету "Львівська політехніка". Серія: Інформаційні системи та мережі. 2023. № 13. С. 45–54.

43. Papineni K., Roukos S., Ward T., Zhu W. J. BLEU: a method for automatic evaluation of machine translation // Proceedings of the 40th annual meeting on association for computational linguistics. 2002. P. 311–318.

44. Lin C. Y. ROUGE: A package for automatic evaluation of summaries // Text summarization branches out. 2004. P. 74–81.

45. Gilardi F., Alizadeh M., Kubli M. ChatGPT outperforms crowd workers for text-annotation tasks // Proceedings of the National Academy of Sciences. 2023. Vol. 120, no. 30.

46. McKinney W. Data Structures for Statistical Computing in Python // Proceedings of the 9th Python in Science Conference. 2010. P. 51–56.

47. Harris C. R. et al. Array programming with NumPy // Nature. 2020. Vol. 585. P. 357–362.

48. Paszke A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library // Advances in Neural Information Processing Systems. 2019. Vol. 32.

49. Закон України «Про захист персональних даних» від 01.06.2010 № 2297-VI // Відомості Верховної Ради України. – 2010. – № 34. – Ст. 481.

50. Закон України «Про вищу освіту» від 01.07.2014 № 1556-VII // Відомості Верховної Ради України. – 2014. – № 37-38. – Ст. 2004.

51. Закон України «Про охорону праці» від 14.10.1992 № 2694-XII // Відомості Верховної Ради України. – 1992. – № 49. – Ст. 668.

52. Кодекс цивільного захисту України від 02.10.2012 № 5403-VI // Відомості Верховної Ради України. – 2013. – № 34-35. – Ст. 458.

53. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. – Введ. 2016–07–01. – Київ : ДП «УкрНДНЦ», 2016. – 17 с.

54. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлення. – Введ. 2017–07–01. – Київ : ДП «УкрНДНЦ», 2016. – 26 с.

55. ДСанПіН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. – Затв. Постановою Головного державного санітарного лікаря України від 10.12.1998 № 7.

56. ДСТУ EN 60950-1:2015. Обладнання інформаційних технологій. Безпека. Частина 1. Загальні вимоги. – Київ : ДП «УкрНДНЦ», 2016.

57. ДСТУ EN ISO 9241-11:2022. Ергономіка взаємодії людина-система. Частина 11. Придатність до використання: визначення та поняття. – Київ : ДП «УкрНДНЦ», 2023.

58. НПАОП 0.00-1.28-10. Правила охорони праці під час експлуатації електронно-обчислювальних машин. – Затв. Наказом Держгірпромнагляду № 65 від 26.03.2010.

59. ISO/IEC 9126-1:2001. Software engineering – Product quality – Part 1: Quality model. Geneva : ISO, 2001. 26 p.

60. ISO/IEC 25010:2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. Geneva : ISO, 2011. 34 p.

61. Словник української мови : в 11 т. / АН УРСР. Інститут мовознавства; за ред. І. К. Білодіда. – Київ : Наукова думка, 1970–1980.

62. Гнатюк С. О. Кібербезпека та захист інформаційних систем : підручник. – Київ : НАУ, 2020. – 384 с.

63. Л.В. Волинець, Н.А. Гарматюк, В.А. Готович. Великі за обсягом набори біомедичних даних та машинне навчання. Збірник тез доповідей XII Міжнародної науково-практичної конференції молодих учених та студентів «Актуальні задачі сучасних технологій» – Тернопіль, 6-7 грудня 2023 року. с. 370-371. https://elartu.tntu.edu.ua/handle/lib/43843

64. Петрик М. Р., Петрик О. Ю. Моделювання програмного забезпечення : наук.-метод. посіб. Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2015. 200 с.

65. Kharchenko A., Bodnarchuk I., Yatcysyn V. The Method for Comparative Evaluation of Software Architecture with Accounting of Trade-offs // American Journal of Information Systems. 2014. Vol. 2, no. 1. P. 20–25. URL: http://pubs.sciepub.com/ajis/2/1/5/.

66. Tymoshchuk D., Yasniy O., Mytnyk M., Zagorodna N., Tymoshchuk V. Detection and classification of DDoS flooding attacks by machine learning method // CEUR Workshop Proceedings. 2024. Vol. 3842. P. 184–195.

67. Konovalenko I., Maruschak P., Brevus V. Steel surface defect detection using an ensemble of deep residual neural networks // Journal of Computing and Information Science in Engineering. 2022. Vol. 22, no. 1. P. 014501.

68. Duda O., Kochan V., Kunanets N., Matsiuk O., Pasichnyk V., Sachenko A., Pytlenko T. Data processing in IoT for smart city systems // Proceedings of the 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS). Metz, France, 2019. Vol. 1. P. 96–99.

69. Матійчук Л., Готович В., Бонар В. Порівняння ефективності методів некерованого машинного навчання для виявлення аномалій в OBD2 даних.

Вимірювальна та обчислювальна техніка в технологічних процесах. Хмельницький національний університет. (1), 2025. С. 407–414. https://doi.org/10.31891/2219-9365-2025-81-52

ДОДАТКИ

Додаток А

Лістинги програмного коду

Лістинг модуля конфігурації системи (config.py)

```
MODEL_CONFIG = {
    'name': 'mistral:7b-instruct',
    'temperature': 0.3,
    'top_p': 0.8,
    'top_k': 20,
    'repeat_penalty': 1.1,
    'num_predict': 512
}

RAG_CONFIG = {
    'n_results': 10,
    'search_multiplier': 3,
    'general_tntu_keywords': ['тнту', 'університет', 'факультет',
'історія', 'пулюй', 'спеціальність', 'кафедра', 'структура'],
    'work_keywords': ['робота', 'дослідження', 'студент', 'автор',
'написав', 'зробив', 'проект'],
    'tntu_knowledge_ratio': 0.9,
    'works_ratio': 0.2,
}

PDF_CONFIG = {
    'chunk_size': 1000,
    'chunk_overlap': 200
}

PATHS = {
    'vector_db': 'data/vector_db',
    'student_works': 'data/student_works',
    'tntu_knowledge': 'data/tntu_knowledge/tntu_info.txt'
}

SYSTEM_PROMPT = """Ти - AI-асистент Тернопільського національного
технічного університету імені Івана Пулюя (ТНТУ).

Твої принципи:
- Відповідай на основі наданої інформації
- Для питань про університет використовуй базові знання ТНТУ
- Для питань про дослідження використовуй роботи студентів
- Вказуй джерела при посиланні на роботи
- Відповідай українською мовою

Питання: {query}

Відповідь:"""
```

```
RAG_PROMPT = """Ти - AI-асистент Тернопільського національного технічного університету імені Івана Пулюя (ТНТУ).

Інструкції:
- Аналізуй надану інформацію з різних джерел
- Для загальних питань про ТНТУ пріоритизуй базові знання
- Для питань про дослідження використовуй роботи студентів
- Структуруй відповідь логічно
- Вказуй автора при посиланні на роботу
- Відповідай українською мовою

Питання: {query}

Інформація:
{context}

Відповідь:"""
```

Лістинг модуля автоматизованого збору даних (scraper.py)

```
import requests
from bs4 import BeautifulSoup
import time
import os
from pathlib import Path
import json
import re

class TNTUScraper:
    def __init__(self, base_url="https://elartu.tntu.edu.ua"):
        self.base_url = base_url
        self.session = requests.Session()
        self.session.headers.update({
            'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36'
        })

    def get_works_from_page(self, page_url):
        works = []
        try:
            print(f"Завантаження: {page_url}")
            response = self.session.get(page_url, timeout=30)
            response.raise_for_status()
            soup = BeautifulSoup(response.text, 'html.parser')
            table = soup.find('table', class_='table')

            if not table:
                return works

            for row in table.find_all('tr'):
                title_cell = row.find('td', headers='t4')
                if title_cell:
                    link = title_cell.find('a', href=True)
                    if link:
                        href = link.get('href')
                        if href and '/handle/' in href:
                            full_url = self.base_url + href if not
href.startswith('http') else href
                            works.append({'url': full_url, 'title':
link.get_text(strip=True)})
                            print(f"  ✓ {link.get_text(strip=True)[:80]}...")

            print(f"Знайдено {len(works)} робіт\n")
        except Exception as e:
            print(f"❌ Помилка: {e}")

        return works

    def get_work_details(self, work_url):
        details = {'url': work_url, 'title': None, 'author': None, 'year': None,
                   'type': None, 'faculty': None, 'specialty': None, 'pdf_url':
None}

        try:
            print(f"  Завантаження деталей...")
            response = self.session.get(work_url, timeout=30)
            response.raise_for_status()
            soup = BeautifulSoup(response.text, 'html.parser')

            h1 = soup.find('h1')
            if h1:
                details['title'] = h1.get_text(strip=True)
```

```
            for table in soup.find_all('table', class_='table'):
                for row in table.find_all('tr'):
                    cells = row.find_all('td')
                    if len(cells) >= 2:
                        label = cells[0].get_text(strip=True).lower()
                        value = cells[1].get_text(strip=True)

                        if 'автор' in label or 'author' in label:
                            details['author'] = value
                        elif 'дата' in label or 'рік' in label:
                            year_match = re.search(r'(\d{4})', value)
                            if year_match:
                                details['year'] = year_match.group(1)
                        elif 'тип' in label or 'type' in label:
                            details['type'] = value
                        elif 'факультет' in label or 'faculty' in label:
                            details['faculty'] = value
                        elif 'спеціальність' in label or 'specialty' in label:
                            details['specialty'] = value

            # Пошук PDF
            file_table = soup.find('table', class_='table panel-body')
            if file_table:
                for cell in file_table.find_all('td', headers='t1'):
                    link = cell.find('a', href=True)
                    if link:
                        href = link.get('href')
                        if href and '.pdf' in href.lower():
                            details['pdf_url'] = self.base_url + href if not
href.startswith('http') else href
                            print(f"  ✓ PDF знайдено")
                            break

            if not details['pdf_url']:
                print(f"  ⚠ PDF не знайдено")

            time.sleep(1)
        except Exception as e:
            print(f"  ✗ Помилка: {e}")

        return details

    def download_pdf(self, pdf_url, output_dir, filename):
        if not pdf_url:
            return None

        try:
            print(f"  Завантаження PDF...")
            Path(output_dir).mkdir(parents=True, exist_ok=True)
            filepath = os.path.join(output_dir, filename)

            response = self.session.get(pdf_url, stream=True, timeout=60)
            response.raise_for_status()

            with open(filepath, 'wb') as f:
                for chunk in response.iter_content(chunk_size=8192):
                    f.write(chunk)

            file_size = os.path.getsize(filepath) / (1024 * 1024)
            print(f"  ✓ Завантажено: {filename} ({file_size:.2f} MB)")
            time.sleep(2)
            return filepath
        except Exception as e:
            print(f"  ✗ Помилка: {e}")
```

```
            return None

    def scrape_works(self, collection_url, output_dir="data/student_works",
limit=10):
        print(f"\n{'='*80}\nПОЧАТОК ЗБОРУ РОБІТ\n{'='*80}\n")

        works = self.get_works_from_page(collection_url)
        if not works:
            print("\n⚠ Роботи не знайдено!")
            return []

        works = works[:limit]
        results = []

        for idx, work in enumerate(works, 1):
            print(f"\n{'-'*80}\nРОБОТА {idx}/{len(works)}\n{'-'*80}")
            print(f"Назва: {work['title'][:80]}...")

            details = self.get_work_details(work['url'])
            pdf_path = None
            if details['pdf_url']:
                pdf_path = self.download_pdf(details['pdf_url'], output_dir,
f"work_{idx:02d}.pdf")

            result = {
                'id': f"work_{idx:03d}",
                'title': details['title'] or work['title'],
                'author': details['author'] or 'Невідомо',
                'year': details['year'] or '2025',
                'type': details['type'] or 'Магістерська робота',
                'faculty': details['faculty'] or 'ФІС',
                'specialty': details['specialty'] or '122 Комп\'ютерні науки',
                'url': work['url'],
                'local_path': pdf_path
            }
            results.append(result)

        metadata_path = os.path.join(output_dir, 'metadata.json')
        with open(metadata_path, 'w', encoding='utf-8') as f:
            json.dump(results, f, ensure_ascii=False, indent=2)

        print(f"\n{'='*80}\nЗАВЕРШЕНО\n{'='*80}")
        print(f"✓ Оброблено: {len(results)}")
        print(f"✓ PDF: {sum(1 for r in results if r['local_path'])}")
        print(f"✓ Метадані: {metadata_path}\n")

        return results


if __name__ == "__main__":
    scraper = TNTUScraper()
    scraper.scrape_works(
        collection_url="https://elartu.tntu.edu.ua/handle/lib/23470",
        output_dir="data/student_works",
        limit=50
    )
```

Лістинг модуля попередньої обробки документів (pdf_processor.py)

```
import os
import json
import pdfplumber

try:
    from config import PDF_CONFIG, PATHS
except ImportError:
    PDF_CONFIG = {'chunk_size': 1000, 'chunk_overlap': 200}
    PATHS = {'student_works': 'data/student_works'}

class PDFProcessor:
    def __init__(self, works_dir=None):
        if works_dir is None:
            works_dir = PATHS.get('student_works', 'data/student_works')
        self.works_dir = works_dir
        self.metadata_path = os.path.join(works_dir, "metadata.json")
        self.processed_data_path = os.path.join(works_dir,
"processed_works.json")

    def extract_text_from_pdf(self, pdf_path):
        text = ""
        try:
            print(f"Обробка PDF: {os.path.basename(pdf_path)}")
            with pdfplumber.open(pdf_path) as pdf:
                for page_num, page in enumerate(pdf.pages, 1):
                    page_text = page.extract_text()
                    if page_text:
                        text += f"\n--- Сторінка {page_num} ---\n{page_text}"
                print(f"  Сторінок: {len(pdf.pages)}, Символів: {len(text)}")
        except Exception as e:
            print(f"❌ Помилка: {e}")
        return text.strip()

    def clean_text(self, text):
        text = ' '.join(text.split())
        return text.strip()

    def split_into_chunks(self, text, chunk_size=None, overlap=None):
        chunk_size = chunk_size or PDF_CONFIG.get('chunk_size', 1000)
        overlap = overlap or PDF_CONFIG.get('chunk_overlap', 200)
        chunks = []
        start = 0
        while start < len(text):
            end = start + chunk_size
            chunk = text[start:end]
            if end < len(text):
                last_period = chunk.rfind('.')
                if last_period > chunk_size // 2:
                    chunk = chunk[:last_period + 1]
                    end = start + last_period + 1
            chunks.append(chunk.strip())
            start = end - overlap
        return chunks

    def process_all_works(self):
        if not os.path.exists(self.metadata_path):
            print(f"Метадані не знайдено: {self.metadata_path}")
            return []

        with open(self.metadata_path, 'r', encoding='utf-8') as f:
            metadata = json.load(f)

        processed_works = []
```

```
        print(f"\n{'='*70}\nОБРОБКА {len(metadata)} РОБІТ\n{'='*70}\n")

        for idx, work_meta in enumerate(metadata, 1):
            title = work_meta.get('title', 'Без назви') or 'Без назви'
            print(f"\n--- Робота {idx}/{len(metadata)} ---")
            print(f"Назва: {title[:60]}...")

            pdf_path = work_meta.get('local_path')
            if not pdf_path or not os.path.exists(pdf_path):
                print(f"  ⚠ PDF не знайдено")
                continue

            raw_text = self.extract_text_from_pdf(pdf_path)
            if not raw_text or len(raw_text) < 100:
                print(f"  ⚠ Текст занадто короткий")
                continue

            cleaned_text = self.clean_text(raw_text)
            chunks = self.split_into_chunks(cleaned_text)
            print(f"  ✓ Чанків: {len(chunks)}")

            processed_work = {
                'id': f"work_{idx:03d}",
                'metadata': {
                    'title': work_meta.get('title'),
                    'author': work_meta.get('author'),
                    'year': work_meta.get('year'),
                    'type': work_meta.get('type'),
                    'faculty': work_meta.get('faculty'),
                    'specialty': work_meta.get('specialty'),
                    'url': work_meta.get('url'),
                    'pdf_path': pdf_path
                },
                'full_text': cleaned_text[:5000],
                'chunks': chunks,
                'num_chunks': len(chunks)
            }
            processed_works.append(processed_work)

        with open(self.processed_data_path, 'w', encoding='utf-8') as f:
            json.dump(processed_works, f, ensure_ascii=False, indent=2)

        print(f"\n{'='*70}\nЗАВЕРШЕНО\n{'='*70}")
        print(f"Оброблено: {len(processed_works)}")
        print(f"Чанків: {sum(work['num_chunks'] for work in processed_works)}")
        print(f"Збережено: {self.processed_data_path}\n")

        return processed_works


if __name__ == "__main__":
    processor = PDFProcessor()
    processor.process_all_works()
```

Лістинг модуля векторизації та зберігання даних (embeddings.py)

```
import os
import json
import chromadb
from sentence_transformers import SentenceTransformer
from tqdm import tqdm

try:
    from config import PATHS
except ImportError:
    PATHS = {
        'student_works': 'data/student_works',
        'tntu_knowledge': 'data/tntu_knowledge/tntu_info.txt'
    }

class VectorDatabase:
    def __init__(self, db_path="data/vector_db", model_name="paraphrase-
multilingual-MiniLM-L12-v2"):
        self.db_path = db_path
        self.client = chromadb.PersistentClient(path=db_path)
        self.collection = self.client.get_or_create_collection(
            name="tntu_student_works",
            metadata={"description": "Студентські роботи ТНТУ"}
        )
        print(f"Завантаження моделі: {model_name}")
        self.embedding_model = SentenceTransformer(model_name)
        print("✓ Модель завантажена\n")

    def create_embeddings(self, texts):
        return self.embedding_model.encode(texts, show_progress_bar=True,
convert_to_numpy=True).tolist()

    def add_works_to_database(self, processed_works_path=None):
        if processed_works_path is None:
            processed_works_path = os.path.join(PATHS.get('student_works',
'data/student_works'), 'processed_works.json')
        if not os.path.exists(processed_works_path):
            print(f"Файл не знайдено: {processed_works_path}")
            return

        with open(processed_works_path, 'r', encoding='utf-8') as f:
            processed_works = json.load(f)

        print(f"\n{'='*70}\nДОДАВАННЯ {len(processed_works)} РОБІТ\n{'='*70}\n")

        all_documents, all_metadatas, all_ids = [], [], []

        for work in tqdm(processed_works, desc="Підготовка"):
            work_id = work['id']
            work_metadata = work['metadata']

            for chunk_idx, chunk in enumerate(work['chunks']):
                chunk_metadata = {
                    'work_id': work_id,
                    'chunk_index': chunk_idx,
                    'title': work_metadata.get('title', 'Без назви'),
                    'author': work_metadata.get('author', 'Невідомо'),
                    'year': work_metadata.get('year', 'Невідомо'),
                    'type': work_metadata.get('type', 'Невідомо'),
                    'faculty': work_metadata.get('faculty', 'Невідомо'),
                    'specialty': work_metadata.get('specialty', 'Невідомо'),
                    'url': work_metadata.get('url', ''),
                }
                all_documents.append(chunk)
```

```
                all_metadatas.append(chunk_metadata)
                all_ids.append(f"{work_id}_chunk_{chunk_idx:03d}")

        print(f"\nВсього чанків: {len(all_documents)}")
        print("Створення embeddings...")
        all_embeddings = self.create_embeddings(all_documents)

        print("Додавання до БД батчами...")
        batch_size = 5000
        for i in range(0, len(all_documents), batch_size):
            end_idx = min(i + batch_size, len(all_documents))
            print(f"      Батч   {i//batch_size   +   1}:   {i+1}-{end_idx}   з
{len(all_documents)}")

            self.collection.add(
                documents=all_documents[i:end_idx],
                embeddings=all_embeddings[i:end_idx],
                metadatas=all_metadatas[i:end_idx],
                ids=all_ids[i:end_idx]
            )

        print(f"✓ Додано {len(all_documents)} чанків\n")

    def add_tntu_knowledge(self, knowledge_path=None):
        if knowledge_path is None:
            knowledge_path             =             PATHS.get('tntu_knowledge',
'data/tntu_knowledge/tntu_info.txt')
        if not os.path.exists(knowledge_path):
            print(f"Файл не знайдено: {knowledge_path}")
            return

        print(f"{'='*70}\nДОДАВАННЯ ЗНАНЬ ПРО ТНТУ\n{'='*70}\n")

        with open(knowledge_path, 'r', encoding='utf-8') as f:
            content = f.read()

        sections = []
        current_section = ""
        for line in content.split('\n'):
            if line.startswith('==') and current_section:
                sections.append(current_section.strip())
                current_section = line + '\n'
            else:
                current_section += line + '\n'
        if current_section:
            sections.append(current_section.strip())

        documents, metadatas, ids = [], [], []
        for idx, section in enumerate(sections):
            if len(section) > 50:
                documents.append(section)
                metadatas.append({
                    'work_id': 'tntu_knowledge',
                    'chunk_index': idx,
                    'title': 'База знань про ТНТУ',
                    'author': 'ТНТУ',
                    'year': '2024',
                    'type': 'Довідкова інформація',
                    'faculty': 'Загальне',
                    'specialty': 'Загальне',
                    'url': 'https://tntu.edu.ua'
                })
                ids.append(f"tntu_knowledge_section_{idx:03d}")

        print(f"Секцій: {len(documents)}")
        print("Створення embeddings...")
```

```
        embeddings = self.create_embeddings(documents)

        print("Додавання до БД...")
        batch_size = 5000
        for i in range(0, len(documents), batch_size):
            end_idx = min(i + batch_size, len(documents))
            self.collection.add(
                documents=documents[i:end_idx],
                embeddings=embeddings[i:end_idx],
                metadatas=metadatas[i:end_idx],
                ids=ids[i:end_idx]
            )

        print(f"✓ Додано {len(documents)} секцій\n")

    def search(self, query, n_results=5):
        query_embedding = self.embedding_model.encode([query]).tolist()
        return           self.collection.query(query_embeddings=query_embedding,
n_results=n_results)

    def get_database_stats(self):
        return {
            'total_chunks': self.collection.count(),
            'collection_name': self.collection.name,
            'db_path': self.db_path
        }


if __name__ == "__main__":
    db = VectorDatabase()
    db.add_tntu_knowledge()
    db.add_works_to_database()

    stats = db.get_database_stats()
    print(f"{'='*70}\nСТАТИСТИКА\n{'='*70}")
    print(f"Всього чанків: {stats['total_chunks']}")
    print(f"Колекція: {stats['collection_name']}\n")
```

Лістинг підсистеми семантичного пошуку (rag_system.py)

```
import sys
sys.path.append('src')
from embeddings import VectorDatabase
from config import RAG_CONFIG, SYSTEM_PROMPT, RAG_PROMPT

class RAGSystem:
    def __init__(self, db_path="data/vector_db"):
        print("Ініціалізація RAG...")
        self.db = VectorDatabase(db_path=db_path)
        self.config = RAG_CONFIG
        print("✓ RAG готова\n")

    def search_relevant_context(self, query, n_results=None):
        if n_results is None:
            n_results = self.config['n_results']

        print(f"🔍 Пошук: '{query}'")
        search_n = n_results * self.config['search_multiplier']
        results = self.db.search(query, n_results=search_n)

        if not results['documents'] or not results['documents'][0]:
            print("⚠ Контекст не знайдено")
            return None

        query_lower = query.lower()
        is_general = any(w in query_lower for w in
self.config['general_tntu_keywords'])
        is_work = any(w in query_lower for w in self.config['work_keywords'])

        tntu_results, work_results = [], []
        for doc, meta, dist in zip(results['documents'][0],
results['metadatas'][0],
                                   results['distances'][0] if 'distances' in
results else [0]*len(results['documents'][0])):
            if meta.get('work_id') == 'tntu_knowledge':
                tntu_results.append((doc, meta, dist))
            else:
                work_results.append((doc, meta, dist))

        if is_general and not is_work:
            tntu_n = int(n_results * self.config['tntu_knowledge_ratio'])
            work_n = n_results - tntu_n
            combined = tntu_results[:tntu_n] + work_results[:work_n]
        elif is_work and not is_general:
            work_n = int(n_results * 0.8)
            tntu_n = n_results - work_n
            combined = work_results[:work_n] + tntu_results[:tntu_n]
        else:
            combined = tntu_results[:n_results//2] +
work_results[:n_results//2+1]

        combined = combined[:n_results]

        if not combined:
            print("⚠ Результати не знайдено")
            return None

        context_parts, sources = [], []
        for idx, (doc, meta, dist) in enumerate(combined):
            context_parts.append(f"[Джерело {idx+1}]\n{doc}\n")
            sources.append({
                'index': idx + 1,
```

```
                'title': meta.get('title', 'Без назви'),
                'author': meta.get('author', 'Невідомо'),
                'year': meta.get('year', 'Н/Д'),
                'type': meta.get('type', 'Н/Д'),
                'work_id': meta.get('work_id', ''),
                'relevance': 1 - dist
            })

        print(f"✓ Знайдено {len(sources)} фрагментів\n")
        return {'context': "\n---\n".join(context_parts), 'sources': sources,
'query': query}

    def format_sources(self, sources):
        if not sources:
            return ""

        formatted = "\n\n📚 Джерела:\n"
        unique = {}
        for s in sources:
            if s['work_id'] not in unique:
                unique[s['work_id']] = s

        for idx, s in enumerate(unique.values(), 1):
            formatted += f"\n{idx}. {s['title']}\n   Автор: {s['author']}, Рік:
{s['year']}\n"
        return formatted

    def build_prompt(self, query, context_data):
        if not context_data:
            return SYSTEM_PROMPT.format(query=query)
        return RAG_PROMPT.format(query=query, context=context_data['context'])
```

Лістинг модуля інтеграції з LLM (assistant.py)

```
import sys
sys.path.append('src')
import ollama
from rag_system import RAGSystem
from config import MODEL_CONFIG
import time

class TNTUAssistant:
    def __init__(self, model_name=None, db_path="data/vector_db"):
        print(f"\n{'='*70}\nІНІЦІАЛІЗАЦІЯ AI-АСИСТЕНТА\n{'='*70}\n")
        self.model_name = model_name or MODEL_CONFIG['name']
        self.config = MODEL_CONFIG

        try:
            ollama.list()
            print("✓ Ollama працює\n")
        except Exception as e:
            print(f"❌ Помилка Ollama: {e}")
            sys.exit(1)

        self.rag = RAGSystem(db_path=db_path)
        print(f"{'='*70}\n✓ АСИСТЕНТ ГОТОВИЙ\n{'='*70}\n")

    def generate_response(self, query, use_rag=True):
        print(f"\n{'='*70}\nЗапит: {query}\n{'='*70}\n")

        context_data = self.rag.search_relevant_context(query) if use_rag else
None
        user_message = self.rag.build_prompt(query, context_data)

        print("👹 Генерація відповіді...")
        start = time.time()

        try:
            response = ollama.generate(
                model=self.model_name,
                prompt=user_message,
                options={
                    'temperature': self.config['temperature'],
                    'top_p': self.config['top_p'],
                    'top_k': self.config['top_k'],
                    'repeat_penalty': self.config['repeat_penalty'],
                    'num_predict': self.config['num_predict']
                }
            )

            answer = response['response']
            gen_time = time.time() - start
            print(f"✓ Згенеровано за {gen_time:.2f}с\n")

            full_answer = answer +
(self.rag.format_sources(context_data['sources']) if context_data else "")

            return {
                'query': query,
                'answer': full_answer,
                'sources': context_data['sources'] if context_data else [],
                'generation_time': gen_time,
                'used_rag': use_rag and context_data is not None
            }
        except Exception as e:
            print(f"❌ Помилка: {e}")
```

```
            return {'query': query, 'answer': f"Помилка: {e}"}

    def chat(self):
        print(f"\n{'='*70}\nAI-АСИСТЕНТ ТНТУ\n{'='*70}")
        print("\nКоманди: 'вихід', 'допомога', '!rag on/off'\n" + "="*70 + "\n")

        use_rag = True
        while True:
            try:
                query = input("💬 Ти: ").strip()
                if not query:
                    continue

                if query.lower() in ['вихід', 'exit', 'quit']:
                    print("\n👋 До побачення!")
                    break

                if query.lower() in ['допомога', 'help']:
                    print("\n📚 Приклади:\n - Що таке ТНТУ?\n - Які
факультети?\n - Що зробив Юрчак?\n")
                    continue

                if query.lower() == '!rag off':
                    use_rag = False
                    print("ℹ️ RAG вимкнено\n")
                    continue

                if query.lower() == '!rag on':
                    use_rag = True
                    print("ℹ️ RAG увімкнено\n")
                    continue

                result = self.generate_response(query, use_rag=use_rag)
                print(f"\n🤖 Асистент:\n{result['answer']}\n")
                print(f"⏱️ {result['generation_time']:.2f}c\n{'-'*70}\n")

            except KeyboardInterrupt:
                print("\n\n👋 До побачення!")
                break

if __name__ == "__main__":
    assistant = TNTUAssistant()
    assistant.chat()
```

Лістинг головного файлу керування (main.py)

```
import sys
sys.path.append('src')
from assistant import TNTUAssistant
from config import PATHS

def main():
    print("""
╔═══════════════════════════════════════════════════════════════════════════════
═══╗
╠═══                AI-АСИСТЕНТ ТНТУ ім. ІВАНА ПУЛЮЯ                           ═══╣
╚═══════════════════════════════════════════════════════════════════════════════
═══╝
    """)

    try:
        assistant = TNTUAssistant(db_path=PATHS['vector_db'])
        assistant.chat()
    except KeyboardInterrupt:
        print("\n\n👋 Завершено")
    except Exception as e:
        print(f"\n❌ Помилка: {e}")

if __name__ == "__main__":
    main()
```

Додаток Б

Тези конференцій

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ

М А Т Е Р І А Л И

XIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

17-18 грудня 2025 року

ТЕРНОПІЛЬ
2025

УДК 001
М34

ПРОГРАМНИЙ КОМІТЕТ

Голова: Микола Приймак– професор кафедри комп'ютерних систем та мереж, д.т.н., професор;

Співголови: Павло Марущак– докт. техн. наук, професор, проректор з наукової роботи.
Ігор Баран– канд. техн. наук, доцент, декан факультету ФІС.

Науковий секретар: Галина Семенишин– старший викладач.

Члени: докт. фіз.-мат. наук, професор Василь Кривень; докт. техн. наук, професор Ярослав Литвиненко; докт. техн. наук, професор Микола Карпінський; докт. фіз.-мат. наук, професор Михайло Петрик; канд. техн. наук, доцент Галина Осухівська; канд. пед. наук, доцент Жанна Баб'як; канд. техн. наук, доцент Наталія Загородна.

ОРГАНІЗАЦІЙНИЙ КОМІТЕТ

Голова: Юрій Скоренький– канд. фіз.-мат. наук, доцент кафедри фізики

Члени: канд. техн. наук, доцент Вячеслав Никитюк; канд. техн. наук, доцент Дмитро Михалик; канд. техн. наук, доцент Марія Стадник; канд. техн. наук Євгенія Тиш; ст. викладач Ліліана Джиджора.

М34 Матеріали XIII науково-технічної конференції «Інформаційні моделі, системи та технології» Тернопільського національного технічного університету імені Івана Пулюя, (Тернопіль, 17–18 грудня 2025 р.). – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2025. –242 с.

Адреса оргкомітету: ТНТУ ім. І. Пулюя, м. Тернопіль, вул. Руська, 56, 46001,
тел. (0352) 52-41-33, факс (0352) 254983.
E-mail: conffis2025@gmail.com
Редагування, оформлення та верстка: Галина Семенишин

СЕКЦІЇ КОНФЕРЕНЦІЇ, ЯКІ ПРЕДСТВЛЕНІ В ЗБІРНИКУ

- Математичне моделювання;
- Інформаційні системи та технології;
- Комп'ютерні системи та мережі;
- Програмна інженерія та моделювання складних розподілених систем;
- Новітні фізико-технічні та освітні технології.

В збірнику надруковано тези доповідей XIII науково-технічної конференції «Інформаційні моделі, системи та технології» (Тернопіль, 17–18 грудня 2025 р.) за такими науковими напрямками: математичне моделювання; інформаційні системи та технології; комп'ютерні системи та мережі; програмна інженерія та моделювання складних розподілених систем; новітні фізико-технічні та освітні технології.

Розрахований на науковців, викладачів та студентів вузів.

За зміст тез та дотримання норм академічної доброчесності відповідальність несе автор.

УДК 004.8:004.056.55
В. Готович, к. т. н., доц; В. Попович
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

ДОСЛІДЖЕННЯ ТА РОЗРОБКА AI-АСИСТЕНТА НА ОСНОВІ МОДЕЛІ MISTRAL ДЛЯ СЕРЕДОВИЩА УНІВЕРСИТЕТУ

UDC 004.8:004.056.55
V. Hotovych, PhD., Assoc. Prof.; V. Popovych

RESEARCH AND DEVELOPMENT OF AN AI ASSISTANT BASED ON THE MISTRAL MODEL FOR THE UNIVERSITY ENVIRONMENT

Сучасні університети оперують великим обсягом інформації, включно з навчальними планами, розкладами, документами та новинами. Часто ці дані розпорошені по різних розділах вебсайту та неструктуровані, що ускладнює швидкий доступ до потрібної інформації. Створення AI-консультанта дозволяє централізувати знання, автоматизувати пошук та надавати студентам і викладачам точні відповіді на запитання. Моделі сімейства Mistral мають відкритий код, високу продуктивність та можливість інтеграції з векторними базами знань, що робить їх оптимальним вибором для розробки системи на базі Retrieval-Augmented Generation (RAG).

Метою роботи є розробка AI-асистента, який забезпечує ефективний пошук і надання релевантної інформації з внутрішньої бази університету, сформованої на основі даних сайту, з використанням можливостей моделі Mistral та механізму RAG. Робота передбачає створення системи збору даних із вебсайту університету, розробку власної бази знань, інтеграцію її з моделлю Mistral через RAG та реалізацію інтерфейсу для взаємодії користувачів із системою.

Для реалізації проекту застосовується комбінація сучасних методів машинного навчання та технологій обробки природної мови. Перший етап включає парсинг вебсайту університету із автоматичним вилученням тексту та HTML-структур, а також конвертацією документів у формат, придатний для подальшого аналізу. Отримані дані проходять очищення, нормалізацію та розбиття на семантичні блоки (chunking). Для створення індексованої бази знань застосовується векторне представлення тексту (embeddings), що дозволяє виконувати семантичний пошук за допомогою інструментів на кшталт FAISS або ChromaDB. Використання RAG забезпечує поєднання реального пошуку по базі даних із генеративними можливостями моделі Mistral. Це дозволяє AI-консультанту формувати точні та контекстно-залежні відповіді на запитання користувачів, базуючись на актуальній інформації університету. Крім того, система інтегрує API для взаємодії з користувачами, що може бути реалізовано через веб-інтерфейс, чат або інші канали комунікації.

Розроблений AI-асистент підтвердив на практиці ефективність поєднання моделей Mistral з технологією RAG для обробки внутрішніх даних університету.

Література

1. Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, Douwe Kiela, 2020. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* Advances in Neural Information Processing Systems (NeurIPS 33), pp. 9459–9474.
2. Nils Reimers, Iryna Gurevych, 2019. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.* Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP-IJCNLP), pp. 3982–3992.
3. Yucheng Hu, Yuxing Lu, 2024. *RAG and RAU: A Survey on Retrieval-Augmented Language Models in Natural Language Processing.*
4. Akari Asai, Matt Gardner, Hannaneh Hajishirzi, 2021. *Evidentiality-guided Generation for Knowledge-Intensive NLP Tasks*

ЗБІРНИК НАУКОВИХ
ПРАЦЬ З МАТЕРІАЛАМИ
VI МІЖНАРОДНОЇ
НАУКОВОЇ КОНФЕРЕНЦІЇ

ТЕОРІЯ МОДЕРНІЗАЦІЇ В КОНТЕКСТІ СУЧАСНОЇ СВІТОВОЇ НАУКИ

19 грудня 2025 рік
м. Івано-Франківськ, Україна

Вінниця, Україна
«UKRLOGOS Group»
2025

УДК 082:001
Т 33

Організація, від імені якої випущено видання:
ГО «Міжнародний центр наукових досліджень»
Номер запису організації в Єдиному реєстрі громадських об'єднань: 1499141.

Голова оргкомітету: Сотник С.Г.

Верстка: Білоус Т.В.
Дизайн: Бондаренко І.В.

Рекомендовано до видання Вченою Радою Інституту науково-технічної інтеграції та співпраці. Протокол № 50 від 18.12.2025 року.

*Конференцію зареєстровано Державною науковою установою у сфері управління Міністерства освіти і науки «Український інститут науково-технічної експертизи та інформації» в базі даних науково-технічних заходів України на поточний рік та бюлетені «План проведення наукових, науково-технічних заходів в Україні» (**Посвідчення № 503 від 10.06.2025**).*

*Збірник наукових праць з матеріалами конференції видано офіційно суб'єктом видавничої справи зі **Свідоцтвом ДК № 7860 від 22.06.2023**.*

Т 33 **Теорія модернізації в контексті сучасної світової науки**: збірник наукових праць з матеріалами VI Міжнародної наукової конференції, м. Івано-Франківськ, 19 грудня, 2025 р. / Міжнародний центр наукових досліджень. — Вінниця: ТОВ «УКРЛОГОС Груп, 2025. — 498 с.

ISBN 978-617-8582-09-8
DOI 10.62731/mcnd-19.12.2025

Викладено матеріали учасників VI Міжнародної наукової конференції «Теорія модернізації в контексті сучасної світової науки», яка відбулася 19 грудня 2025 року у місті Івано-Франківськ.

УДК 082:001

ISBN 978-617-8582-09-8

СЕКЦІЯ XIV.
ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ТА СИСТЕМИ

ІНТЕЛЕКТУАЛЬНИЙ АСИСТЕНТ ДЛЯ ПОШУКУ ІНФОРМАЦІЇ В УНІВЕРСИТЕТСЬКИХ БАЗАХ ЗНАНЬ НА ОСНОВІ МОДЕЛІ MISTRAL

Попович Валерій Валерійович
студент кафедри комп'ютерних наук
Тернопільський національний технічний університет імені Івана Пулюя, Україна

Науковий керівник: Готович Володимир Анатолійович
ORCID ID: 0000-0003-2143-6818
канд.техн.наук, доцент, доцент кафедри комп'ютерних наук
Тернопільський національний технічний університет імені Івана Пулюя, Україна

Актуальність теми дослідження. Сучасний університет функціонує як складний генератор значних обсягів інформації. Щороку накопичуються тисячі сторінок неструктурованих текстових даних: методичні вказівки, накази, положення, а також кваліфікаційні роботи студентів. Традиційні інструменти пошуку, що використовуються в більшості освітніх систем, базуються на лексичному співпадінні ключових слів. Такий підхід має суттєвий недолік: якщо запит користувача семантично пов'язаний з документом, але не містить точних термінів з нього, система не повертає релевантний результат.

Стрімкий розвиток великих мовних моделей (Large Language Models, LLM) відкриває нові перспективи для вирішення цієї проблеми. Використання генеративного штучного інтелекту дозволяє створити систему, здатну «розуміти» контекст запиту та формувати вичерпні відповіді. Однак використання публічних хмарних сервісів часто є неможливим через вимоги до конфіденційності даних та залежність від інтернет-з'єднання. Тому актуальним є завдання розробки автономних інтелектуальних асистентів, які розгортаються локально та використовують власну базу знань університету.

Аналіз попередніх досліджень та постановка задачі. Аналіз існуючих рішень показує, що більшість сучасних чат-ботів або працюють за жорстко заданими сценаріями, або використовують API зовнішніх моделей (GPT-4, Claude), що пов'язано з передачею даних третім сторонам. Фундаментальним підходом для інтеграції власних даних у мовні моделі є технологія Retrieval-Augmented Generation (RAG),

запропонована Льюїсом та ін. [1]. Суть методу полягає в динамічному додаванні релевантної інформації до контексту моделі перед генерацією відповіді.

Останні дослідження в галузі відкритих моделей, зокрема поява архітектури Mistral 7B [2], демонструють, що моделі з відносно невеликою кількістю параметрів (7 мільярдів) здатні конкурувати з набагато більшими аналогами за умови якісного донавчання або правильної організації промпт-інжинірингу. Метою даної роботи є створення прототипу AI-асистента, який поєднує можливості локальної моделі Mistral та технології RAG для забезпечення точного семантичного пошуку по базі знань університету без необхідності використання хмарних обчислювальних ресурсів.

Основні результати дослідження. Розроблена система базується на модульній архітектурі, що забезпечує повний цикл обробки даних: від завантаження документів до генерації відповіді користувачу. Для забезпечення конфіденційності та незалежності від зовнішніх провайдерів обрано стратегію локального розгортання (Local Inference). Як середовище виконання використано платформу Ollama [3], що дозволяє запускати квантизовані моделі на споживчому обладнанні. Загальна схема інформаційних потоків у системі наведена на рисунку 1.

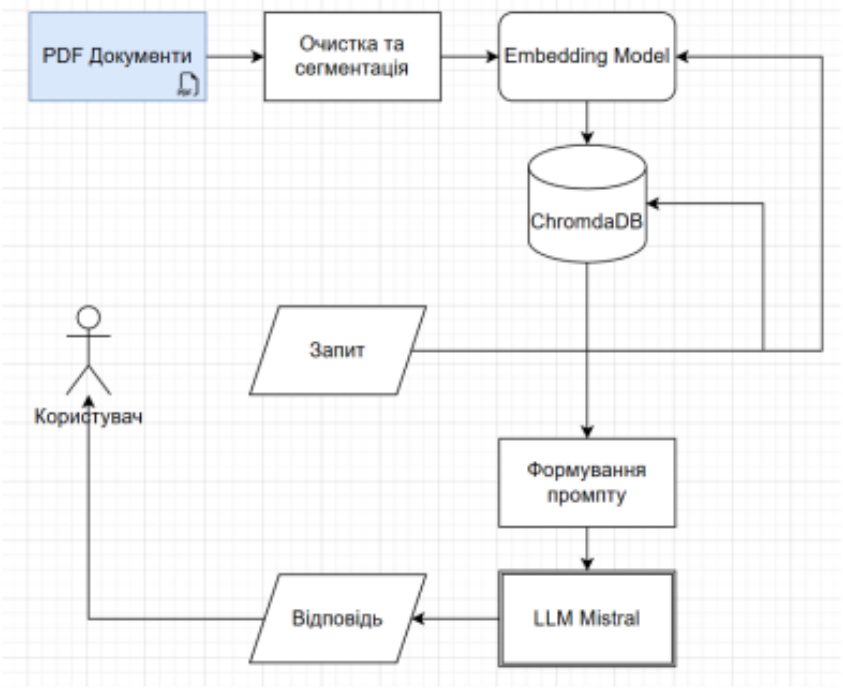

Рис 1. **Структурно-функціональна схема роботи розробленого RAG-асистента**

Якість роботи RAG-системи критично залежить від якості вхідних даних. База знань сформована на основі інформації про структуру Тернопільського національного технічного університету імені Івана Пулюя та архіву магістерських робіт спеціальності «Комп'ютерні науки», отриманих з інституційного репозитарію ELARTU [4].

Для видобування тексту з PDF-документів використано бібліотеку pdfplumber. Оскільки «сирий» текст часто містить технічне сміття (номери сторінок, колонтитули), розроблено алгоритм попередньої очистки. Наступним кроком є сегментація тексту (chunking). Дослідним шляхом встановлено, що оптимальним є розбиття на блоки по 1000 символів із перекриттям (overlap) у 200 символів. Наявність перекриття гарантує збереження семантичного контексту, якщо речення або думка розриваються між двома фрагментами.

Ключові технічні параметри реалізованого рішення наведено в таблиці 1.

Таблиця 1

Параметри конфігурації системи обробки даних

Параметр / Компонент	Значення / Назва	Обґрунтування вибору
Базова LLM	Mistral-7B-Instruct-v0.2	Відкритий код, висока якість інструкцій
Формат моделі	GGUF (4-bit quantization)	Оптимізація під обмежену VRAM (відеопам'ять)
Розмір чанку (Chunk Size)	1000 символів	Оптимальний розмір для одного абзацу/думки
Перекриття (Overlap)	200 символів	Запобігання втраті контексту на межах
Сховище векторів	ChromaDB	Локальна векторна БД, не потребує сервера

Для реалізації можливості пошуку за змістом, а не за формальними ознаками, у роботі використано метод щільних векторних представлень (dense embeddings). Процес векторизації здійснюється за допомогою трансформерної моделі paraphrase-multilingual-MiniLM-L12-v2 [5], яка трансформує кожен текстовий сегмент у багатовимірний числовий вектор розмірністю 384 елементи. Вибір саме цієї моделі зумовлений її мультимовною архітектурою: вона ефективно проєктує запити українською мовою та документи у спільний семантичний простір. Це дозволяє системі знаходити релевантні відповіді навіть у тих випадках, коли лексичні конструкції в запиті користувача та в тексті документа не збігаються (наприклад, при використанні синонімів чи перефразуванні),

але є тотожними за змістом. Згенеровані вектори разом із відповідними метаданими, такими як автор роботи, рік написання та назва файлу, індексуються у високопродуктивній векторній базі даних ChromaDB [6]. При надходженні запиту система виконує ранжування фрагментів шляхом обчислення косинусної подібності між вектором запиту та векторами в базі, що дозволяє миттєво відфільтрувати найбільш релевантний контекст із масиву документів.

Критичним викликом при використанні генеративних моделей у процесі навчання є мінімізація схильності LLM до вигадування фактів, відомої як «галюцинації». Для вирішення цієї проблеми розроблено та імплементовано стратегію суворого системного промптингу (System Prompt Engineering). Алгоритм роботи асистента передбачає динамічне формування запиту до моделі, який складається з системної інструкції, знайденого контексту та безпосереднього питання користувача. Системна інструкція накладає жорсткі обмеження на генерацію, забороняючи моделі використовувати знання, що виходять за межі наданого контексту. Додатковим рівнем верифікації є реалізований механізм автоматичного цитування: фінальна текстова відповідь обов'язково супроводжується блоком «Джерела», де перераховуються конкретні назви документів та номери сторінок, з яких було взято інформацію.

Апробацію розробленого консольного застосунку проведено на апаратній платформі середнього рівня – персональному комп'ютері, оснащеному відеокартою NVIDIA з обсягом відеопам'яті 6 ГБ. Під час експериментів оцінювалися ключові метрики продуктивності, що впливають на користувацький досвід: затримка при векторному пошуку, швидкість генерації токенів мовною моделлю та пікове споживання системних ресурсів. Результати підтвердили ефективність обраних методів оптимізації, зокрема квантизації, що дозволяє системі стабільно працювати без використання дороговартісних серверних рішень. Узагальнені кількісні показники тестування наведено в таблиці 2.

Таблиця 2

Результати тестування продуктивності розробленого асистента

Етап обробки запиту	Середній час виконання	Коментар
Векторизація запиту	5 – 10 с.	Виконується на CPU, швидко
Пошук у ChromaDB	2– 5 с.	Пошук серед ~10 000 фрагментів
Завантаження контексту	1 с.	Видобування тексту з метаданих
Генерація відповіді (LLM)	30 – 120 с.	Залежить від довжини відповіді та GPU

Тестування показало, що система успішно обробляє як фактологічні запити (наприклад, про склад кафедри), так і аналітичні (огляд тем дипломних робіт за певний рік). Використання 4-бітної квантизації дозволило досягти стабільної роботи моделі Mistral-7B, використовуючи лише 4–5 ГБ відеопам'яті, що підтверджує можливість розгортання системи на бюджетному обладнанні.

Висновки. У результаті виконання роботи вирішено науково-прикладну задачу підвищення ефективності інформаційного пошуку в освітньому середовищі. Розроблено та досліджено архітектуру локального AI-асистента, який поєднує генеративні можливості великих мовних моделей із технологією RAG. Це дозволило трансформувати статичну базу знань університету в інтерактивного помічника, здатного надавати вичерпні відповіді на основі неструктурованих даних.

Основні наукові та практичні результати роботи включають:

– реалізацію механізму семантичного аналізу україномовних текстів, що дозволило системі знаходити релевантну інформацію не за ключовими словами, а за змістом запиту;

– забезпечення повної автономності системи завдяки оптимізації моделі для роботи на споживчому обладнанні з обмеженим обсягом відеопам'яті, що виключає необхідність використання платних хмарних ресурсів;

– підвищення рівня академічної доброчесності згенерованих відповідей шляхом впровадження алгоритмів автоматичного цитування та суворої прив’язки до контексту.

Отримані результати доводять, що запропонований підхід є надійною та економічно виправданою основою для цифровізації університету, гарантуючи при цьому захист конфіденційної інформації та персональних даних користувачів.

Список використаних джерел:

1. Lewis P. & others. (2020) Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. URL: https://arxiv.org/abs/2005.11401.
2. Jiang A.Q. & others. (2023) Mistral 7B. URL: https://arxiv.org/abs/2310.06825.
3. Ollama. (2024) Get up and running with large language models. URL: https://ollama.com/.
4. Elartu. (2024) Institutional Repository of TNTU. URL: http://elartu.tntu.edu.ua/.
5. Reimers N. & Gurevych I. (2019) Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. URL: https://aclanthology.org/D19-1410/.
6. ChromaDB. (2024) The open-source embedding database. URL: https://www.trychroma.com/.

Додаток В

Результат запиту до AI-асистента

```
🔍 Пошук: 'Які мови програмування студенти використовують найчастіше?'
✓ Знайдено 8 фрагментів

🤖 Генерація відповіді...
✓ Згенеровано за 60.17с


🤖 Асистент:
 Студенти ТНТУ найчастіше використовують мови програмування Python та C/C++ для різних досліджень і проектів.

Питання про популярність мов програмування для веб-додатків та системи моніторингу доменів було розглянуте в кваліфікаційній роботі [Джерело 1]. Відпов
ідно до цього джерела, для розробки системи було обрано Python через простоту синтаксису, гнучкість та швидкість на різноманітних фреймворках. Для ство
рення системи використовувався Anvil.works, а сервіс Namecheap API - для моніторингу доменів.

У роботі [Джерело 2] згадується, що більшість студентських проектів з IoT розробляються на C, C++ або їх варіантах, оскільки вони є найпопулярнішими ти
пами мов для контролерів. Крім того, Java також використовується для генерації коду, а Node.js - у одному зі знайдених посилань.

У роботі [Джерело 3] згадується, що Python є кращим вибором для системи моніторингу та реєстрації доменних імен через його простоту вивчення та багатий
 набір інструментів.

У роботі [Джерело 4] згадується, що Python є високорівневою, інтерпретованою мовою програмування, яка підтримує кілька парадигм програмування та має по
тужний набір бібліотек.

У роботі [Джерело 5] згадується, що для обробки природної мови Python ідеально підходить через його простоту та читабельність коду.

У роботі [Джерело 6] з

📑 Джерела:

1. Розробка автоматизованої системи для супроводу процесів реєстрації та життєвого циклу доменних імен
   Автор: © Мац Олег Ігорович, 2024, Рік: 2024

2. Аналіз методологій розробки IoT-систем з використанням Agile-технологій
   Автор: © Щеснюк Тетяна Омелянівна, 2024, Рік: 2025

3. Розробка додатку відеотрансляції під мобільні пристрої на базі операційної системи Android
   Автор: © Бондаренко Владислав Сергійович, 2024, Рік: 2025

4. Дослідження впливу методів попередньої обробки текстових даних на якість класифікаційних моделей  машинного навчання
   Автор: © Цимбалюк Гліб Олександр Богданович, Рік: 2025

5. Дослідження застосування методів штучного інтелекту для обробки природної мови засобами глибинного навчання
   Автор: © Климко Ігор Михайлович, 2024, Рік: 2025

6. Методи та алгоритми створення безшовного панорамного зображення із набору фотографій
   Автор: © Романський Степан Володимирович, 2025, Рік: 2025


⏱  60.17с
```