ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Локальна велика мовна модель для автономних комп'ютерних пристроїв
Назва теми

Рівень вищої освіти перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»
Шифр і найменування

Спеціальність 121 «Інженерія програмного забезпечення»
Код і найменування

Освітня програма «Інженерія програмного забезпечення»
Найменування

Шифр КвРІПЗ.2201102.01.11.ПЗ

Виконав здобувач IV курсу група ІПЗ-22-1 _____ Микола ПАРФУТІК
Шифр Підпис Ім'я, ПРІЗВИЩЕ

Керівник к. пед. наук, доцент _____ Наталія ПРАВОРСЬКА
Науковий ступінь, учене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер старший викладач _____ Ганна БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

1 червня 2026
Дата

Хмельницький, 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Інженерії програмного забезпечення

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
_____ Л. П. Бедратюк
02 01 2026 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Парфутіку Миколі Олександровичу
Прізвище, ім'я, по батькові студента

1. Тема роботи Локальна велика мовна модель для автономних комп'ютерних пристроїв

Керівник проєкту Праворська Наталія Іванівна к. пед. наук, доцент
Прізвище, ім'я, по батькові науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. №7-КП

2. Строк подання студентом проєкту на кафедру 01.06.2026 р.

3. Вихідні дані до роботи: Матеріали переддипломної практики

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

Аналіз предметної області та постановка задачі

Проєктування програмного забезпечення

Програмна реалізація та тестування програмного забезпечення

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Три креслення у форматі А3:
1. Діаграма класів
2. DFD-діаграма
3. Діаграма процесів

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Бедратюк Г. І., старший викладач	5.05.26	4.05.26
Антиплагіат	Форкун Ю. В., доцент	15.05.2026	25.05.26

7. Дата видачі завдання « 02 » січня 2026 р.

Календарний графік

виконання кваліфікаційної роботи студентом гр. ІПЗ-22-1 за спеціальністю 121 «Інженерія програмного забезпечення»

№ з/п	Назва етапу виконання	Дата початку-завершення етапу	Примітка
1	Ознайомлення з тематикою дипломного проєктування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	01.12 - 31.12.2025	
2	Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 - 20.02.2026	
3	Проєктування програмного забезпечення	21.02 - 20.03.2026	
4	Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 - 30.04.2026	
5	Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 - 25.05.2026	
6	Попередній захист КвР	травень 2026	Згідно графіка
7	перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 - 30.06.2026	
8	Здача КвР на кафедру; підготовка КвР для розміщення у репозиторій ХНУ; підготовка до захисту та захист КвР	з 01.06.2026	

Студент _____ М. О. ПАРФУТІК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник проєкту (роботи) _____ Н. І. Праворська
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Автономна велика мовна модель для комп'ютерних пристроїв»

Автор проєкту: Парфутік Микола Олександрович.

Керівник проєкту: Праворська Н.І., к. пед. наук, доцент

Пояснювальна записка: 55 ст., 8 рис., 6 таблиці, 3 діаграм, 2 додатка, 30 джерел.

Графічна частина: 3 креслення ф. А3

Ключові слова: велика мовна модель, штучний інтелект, RAG, семантичний пошук, квантизація, GGUF, Python.

Метою проєкту є підвищення рівня конфіденційності та автономності процесів обробки природної мови шляхом розробки оптимізованого локального веб-застосунку на базі великої мовної моделі.

У кваліфікаційній роботі виконано аналіз методів оптимізації нейронних мереж для роботи в умовах обмежених апаратних ресурсів (Edge AI). Спроєктовано та реалізовано локальний веб-застосунок, що поєднує використання квантованої мовної моделі у форматі GGUF та підсистеми семантичного аналізу локальних документів (технологія RAG). Розроблено інтуїтивно зрозумілий графічний інтерфейс користувача з підтримкою збереження історії сесій у локальну базу даних. Програмний продукт забезпечує повну ізоляцію оброблюваних даних (Air-gapped середовище) без підключення до Інтернету.

27.05.2026 -

Парфутік

ВІДОМІСТЬ ДОКУМЕТІВ

№ рядка	Формат	Позначення документа	Найменювання документа	К-ть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.2201102.01.11.ПЗ	Пояснювальна записка			
2	А4		Завдання на кваліфіка-			
			ційну роботу			
3	А4		Анотація			
			Графічні документи			
4	А3	КвРІПЗ.2201102.01.11.Е8	Діаграма класів	1		
5	А3	КвРІПЗ.2201102.01.11.Е8	DFD-діаграма	1		
6	А3	КвРІПЗ.2201102.01.11.Е8	Діаграма процесів	1		

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.2201102.01.11.ПЗ		
Розроб.		Парфутік М.О.		29.05.20	Локальна велика мовна модель для автономних комп'ютерних пристроїв	Літ.	Арк. 5 / Акрушів 66
Перевір.		Праворська Н.І.		29.05.20			
Реценз.						ХНУ. ІПЗ-22-1	
Н. Контр.		Бедратюк Г. І.		29.05.20	Пояснювальна записка		
Затверд.		Бедратюк Л. П.		29.05.20			

ЗМІСТ

ВСТУП

Сучасний світ характеризується стрімким розвиток інформаційних технологій. Комп'ютерні системи стали невід'ємною частиною повсякденного життя, еволюціонувавши від інструментів для виконання специфічних обчислень до персональних помічників [1]. Якщо на ранніх етапах розвитку обчислювальної техніки користувачеві доводилося вивчати спеціальні команди та адаптуватися до машинної логіки, то сучасна парадигма передбачає адаптацію комп'ютерних систем до потреб людини.

Ключовим фактором, що забезпечує таку взаємодію, є прогрес у галузі обробки природної мови (NLP) та поява великих мовних моделей (LLM) [2]. Ці технології дозволяють перейти від командних інтерфейсів до інтуїтивно зрозумілого діалогу, де система розуміє контекст та наміри користувача. Однак більшість сучасних рішень (наприклад, ChatGPT, Gemini) базуються на хмарних обчисленнях. Це створює низку проблем: залежність від інтернет-з'єднання, затримки у передачі даних та, що найважливіше, ризики конфіденційності персональної інформації.

У зв'язку з цим виникає гостра потреба у впровадженні концепції Edge AI (штучний інтелект на периферійних пристроях). Світова тенденція до збільшення кількості застосувань On-Device AI стимулює попит на легкі та енергоефективні моделі, здатні працювати автономно. Інтеграція таких моделей у комп'ютерні пристрої вимагає вирішення науково-практичних завдань щодо їх оптимізації методами квантизації та прунінгу для роботи в умовах обмежених ресурсів. Таким чином, розробка автономної великої мовної моделі є актуальною задачею, що відповідає сучасним вимогам до безпеки, швидкодії та зручності користувача.

Метою роботи є створення інтуїтивно зрозумілого, людино-орієнтованого програмного засобу на базі автономної мовної моделі, що забезпечує високу

продуктивність та конфіденційність діалогу на локальному комп'ютерному пристрої.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- здійснити аналіз предметної області та сучасних тенденцій у сфері NLP, виявити недоліки хмарних рішень та обґрунтувати переваги локального використання великих мовних моделей (Edge AI).
- провести порівняльну характеристику методів оптимізації нейронних мереж (квантизація, прунінг, дистиляція знань) та обрати оптимальний підхід для адаптації моделі до апаратного забезпечення середнього рівня.
- спроєктувати архітектуру програмної системи, розробити діаграми потоків даних (DFD) та алгоритми взаємодії користувача з локальною моделлю, передбачивши механізми перевірки валідності результату.
- обґрунтувати вибір стеку технологій (мова програмування, фреймворки машинного навчання, бібліотеки інтерфейсу) для реалізації технічного завдання.
- виконати програмну реалізацію автономного модуля з використанням механізмів локального кешування контексту діалогу для підвищення релевантності відповідей.
- розробити графічний інтерфейс користувача (GUI), що забезпечує зручне введення текстових запитів та відображення згенерованих відповідей.
- провести тестування розробленого програмного забезпечення, оцінити його ефективність за показниками швидкості генерації, споживання оперативної пам'яті та якості відповідей.

Об'єкт дослідження – процес взаємодії користувача з комп'ютерною системою за допомогою природної мови в умовах обмежених апаратних ресурсів.

Предмет дослідження – методи та засоби оптимізації великих мовних моделей для забезпечення їх автономного функціонування на локальних пристроях.

Розроблений програмний продукт є готовим інструментом для безпечної роботи з генеративним штучним інтелектом без доступу до мережі Інтернет. Запропоноване рішення може бути використане як базова платформа для створення спеціалізованих корпоративних асистентів, де критично важливою є конфіденційність інформації. Отримані результати можуть бути впроваджені у навчальний процес при вивченні дисциплін, пов’язаних зі штучним інтелектом та проєктуванням ПЗ.

1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області, її структурних та функціональних особливостей

Сучасний світ характеризується стрімким темпом цифровізації всіх сфер людської діяльності. Комп'ютерні системи, які ще кілька десятиліть тому здавалися спеціалізованими інструментами, сьогодні стали невід'ємною частиною побуту, освіти та професійної діяльності. Ця швидка еволюція апаратного та програмного забезпечення призвела до фундаментальних змін не лише у фізичній формі пристроїв, але й у парадигмі взаємодії людини з машиною.

Історично розвиток інтерфейсів «людина-комп'ютер» (HCI – Human-Computer Interaction) пройшов кілька етапів. На початках ери обчислювальної техніки користувачеві доводилося адаптуватися до машини: вивчати спеціалізовані мови програмування, запам'ятовувати текстові команди консолі та розуміти внутрішню логіку роботи системи. Згодом поява графічних інтерфейсів (GUI) спростила цей процес, але все ще вимагала від користувача навичок оперування меню та вікнами.

Сьогодні спостерігається перехід до нового етапу, де комп'ютерні системи прагнуть адаптуватися до людини. Мета сучасної розробки полягає у створенні інтуїтивно зрозумілого, «людино-орієнтованого» діалогу, який стирає межі між природною мовою та цифровим керуванням. Користувач формулює запит звичайною мовою, а система повинна його семантично зрозуміти та виконати.

Критично важливим фактором, що уможливив такий перехід, став прогрес у галузі штучного інтелекту, зокрема в обробці природної мови (Natural Language Processing, NLP) [3]. Поява архітектури Трансформерів (Transformer) та навчання великих мовних моделей (Large Language Models, LLM) дозволили

технологіям вийти на якісно новий рівень: від виконання жорстко запрограмованих сценаріїв до розуміння контексту, намірів користувача та генерації зв’язних, осмислених відповідей [4], [5].

Стрімкий розвиток великих мовних моделей (LLM) призвів до їх масового впровадження у бізнес-процеси та повсякденне життя. Проте абсолютна більшість сучасних рішень (OpenAI ChatGPT, Anthropic Claude, Google Gemini) функціонують за моделлю SaaS (Software as a Service), вимагаючи постійного підключення до Інтернету та передачі даних на віддалені сервери. Це створює критичні загрози для інформаційної безпеки.

По-перше, передача чутливої корпоративної інформації (коду, фінансових звітів, персональних даних клієнтів) через сторонні API порушує суворі політики безпеки та угоди про нерозголошення (NDA). Історія знає прецеденти, коли комерційні таємниці компаній потрапляли у навчальні вибірки хмарних моделей і згодом відтворювалися у відповідях іншим користувачам.

По-друге, залежність від хмарної інфраструктури робить неможливим використання таких систем в Air-gapped середовищах — фізично ізольованих комп'ютерних мережах, які застосовуються на об'єктах критичної інфраструктури, у військовому секторі та наукових лабораторіях.

По-третє, мережева затримка (Latency). При відправленні запиту на сервер і очікуванні відповіді виникає затримка, яка є неприпустимою для систем реального часу. Парадигма Edge AI (периферійний штучний інтелект) вирішує ці проблеми шляхом перенесення процесу інференсу (висновку) безпосередньо на кінцевий пристрій користувача. Це гарантує нульовий ризик витоку даних, оскільки весь життєвий цикл обробки інформації замикається в межах локальної оперативної пам'яті комп'ютера.

Проте, незважаючи на успіхи провідних індустріальних рішень, сучасна екосистема NLP має суттєвий архітектурний недолік: більшість найпотужніших моделей функціонують виключно у хмарному середовищі. Централізована

хмарна архітектура створює низку структурних проблем, що стримують повсюдну інтеграцію ШІ в корпоративний та приватний сектори:

- загрози конфіденційності та безпеці даних – передача чутливої інформації (персональних даних, корпоративних документів, комерційної таємниці) на сторонні сервери несе ризики витоку інформації та порушення умов NDA (угоди про нерозголошення);
- залежність від мережевого зв’язку – хмарні асистенти не здатні функціонувати за відсутності стабільного інтернет-з’єднання, що є неприпустимим для ізольованих (Air-gapped) комп'ютерних систем та пристроїв, які працюють у польових умовах;
- мережева затримка (Latency) – час, необхідний на передачу даних до сервера і назад, створює затримку в генерації відповідей, що погіршує користувацький досвід у режимі реального часу;
- висока вартість масштабування – використання API хмарних сервісів тарифікується за кількістю оброблених токенів, що робить регулярну обробку великих масивів тексту економічно невигідною.

У відповідь на ці виклики в IT-індустрії сформувався концептуальний підхід Edge AI (штучний інтелект на периферії). Ця концепція передбачає перенесення нейромережевих обчислень з хмари безпосередньо на автономні кінцеві пристрої користувачів (персональні комп'ютери, ноутбуки). Локальна обробка даних гарантує повну незалежність від Інтернету, знижує затримки та забезпечує абсолютний захист даних, оскільки інформація ніколи не залишає фізичний периметр пристрою.

Однак розгортання великих мовних моделей на автономних комп'ютерних пристроях наштовхується на проблему обмеженості апаратних ресурсів. Сучасні LLM мають мільярди параметрів і потребують значних обсягів оперативної (RAM) та відеопам’яті (VRAM), якими зазвичай не володіють стандартні робочі станції. Аналіз сучасних тенденцій у галузі локального ШІ виділяє наступні напрямки вирішення проблеми апаратних обмежень:

а) застосування математичних методів стиснення моделей:

1) квантизація (quantization) – зменшення розрядності ваг нейромережі (наприклад, перехід від 16-бітних чисел з плаваючою комою до 4-бітних цілих чисел), що дозволяє радикально зменшити розмір моделі в пам'яті;
2) прунінг (pruning) – алгоритмічне видалення найменш значущих нейронних зв’язків;

б) використання оптимізованих форматів збереження моделей (наприклад, GGUF), які дозволяють ефективно розподіляти навантаження між центральним (CPU) та графічним (GPU) процесорами;

в) апаратна акселерація обчислень за рахунок вбудованих нейронних співпроцесорів або спеціалізованих графічних API (зокрема, технології Metal Performance Shaders для архітектури Apple Silicon).

Окрім подолання апаратних обмежень, важливою функціональною особливістю предметної області є необхідність інтеграції локального ШІ з призначеними для користувача даними. Оскільки локальні моделі працюють автономно і не можуть шукати інформацію в Інтернеті, виникає потреба у застосуванні технології пошукової генерації (RAG – Retrieval-Augmented Generation) [6], [7]. Це дозволяє системі аналізувати локальні документи та формувати фактологічно точні відповіді без необхідності ресурсоємного донавчання (fine-tuning) самої моделі.

Таким чином, розробка локальної великої мовної моделі для автономних комп’ютерних пристроїв є актуальним науково-практичним завданням. Вона дозволяє створити безпечний, незалежний та оптимізований інструмент, що поєднує інтелектуальні можливості сучасних LLM із повною конфіденційністю даних користувача.

1.2 Аналіз останнії публікацій, досліджень та існуюючих рішень предметної області

Сучасні великі мовні моделі (LLM), такі як GPT-4, Llama 3 або Claude, демонструють вражаючі результати в задачах обробки природної мови [8], [9]. Однак їхня архітектура вимагає значних обчислювальних ресурсів. Наприклад, стандартна модель із 7 мільярдами параметрів (7B), що використовує 16-бітну точність (FP16), потребує близько 14–16 ГБ відеопам'яті (VRAM) лише для завантаження ваг [10], [11]. Це робить їх запуск на споживчих автономних пристроях (Edge Devices) неможливим без спеціальної оптимізації [12], [13].

Аналіз останніх наукових публікацій та інженерних підходів показує, що для вирішення проблеми «важковаговитості» моделей застосовують методи компресії (стиснення) нейронних мереж. До основних методів належать:

а) квантизація (quantization) – ключовий метод, який дозволив здійснити прорив у локальному запуску штучного інтелекту. Суть методу полягає у зменшенні розрядності чисел, якими представлені параметри нейронної мережі (від FP16 до цілих чисел INT8 або INT4). У рамках даної роботи особливу увагу приділено формату GGUF (GPT-Generated Unified Format), оптимізованому для швидкого виконання на процесорах архітектури Apple Silicon. Використання 4-бітної квантизації (метод Q4_K_M) забезпечує:

1) радикальне зменшення обсягу пам'яті (наприклад, з 15 ГБ до ~4.5 ГБ);
2) збільшення швидкості обчислень завдяки операціям з цілими числами;
3) мінімальну втрату якості генерації тексту;

б) дистиляція знань (knowledge distillation) – підхід, що передбачає навчання меншої моделі-студента (Small Language Models, SLM) на

відповідях великої моделі-вчителя. Студент вчиться імітувати логіку мислення вчителя, що дозволяє компактним моделям демонструвати високу результативність, споживаючи в рази менше енергії;

в) прунінг (pruning) – алгоритмічне «проріджування», що базується на видаленні з нейронної мережі зв’язків, які мають близьке до нуля значення. Це створює розріджені матриці ваг, проте на практиці квантизація виявляється більш універсальною для імплементації в локальні застосунки.

Практична реалізація зазначених методів компресії у сучасній інженерії програмного забезпечення вимагає використання спеціалізованих форматів збереження та оптимізації ваг нейронних мереж. На сьогодні в індустрії відкритого програмного забезпечення сформувалося кілька провідних стандартів квантизації для локального інференсу моделей.

Першим поширеним стандартом є GPTQ (Generative Pre-trained Transformer Quantization) — алгоритм квантизації після навчання (Post-Training Quantization), який стискає ваги до 4 або 3 біт, проте він оптимізований виключно для виконання на дискретних графічних процесорах (GPU) і демонструє низьку продуктивність на класичних CPU. Еволюційним розвитком цього підходу став метод AWQ (Activation-aware Weight Quantization), який враховує активації під час стиснення і зберігає найбільш значущі ваги у вищій точності, що мінімізує деградацію відповідей ШІ, але цей формат так само залишається жорстко прив'язаним до екосистеми NVIDIA.

Найбільш універсальним рішенням для кросплатформного розгортання є формат GGUF (GPT-Generated Unified Format). Його ключова перевага полягає в тому, що він зберігає всю необхідну метаінформацію (архітектуру, токенізатор, параметри контексту) в одному бінарному файлі та ідеально оптимізований для гетерогенних обчислень. Це дозволяє гнучко розподіляти навантаження між CPU та GPU, а також забезпечує повну сумісність із технологією уніфікованої пам'яті

архітектури Apple Silicon, що робить його найкращим технологічним вибором для реалізації автономного застосунку

Для обґрунтування доцільності розробки власного програмного забезпечення було проведено порівняльний аналіз існуючих рішень для використання LLM. Сьогодні на ринку домінують два підходи: використання закритих хмарних API (наприклад, ChatGPT, Claude) та універсальні десктопні програми для локального запуску (LM Studio, GPT4All, Ollama).

Хоча локальні платформи типу LM Studio вирішують проблему приватності, вони зазвичай є перевантаженими технічними налаштуваннями десктопними програмами. Крім того, вони мають обмежені або складні в налаштуванні вбудовані механізми для інтелектуального аналізу локальних документів користувача без підключення сторонніх плагінів. Це створює потребу в розробці легковагового, цільового веб-застосунку з інтегрованою системою RAG (Retrieval-Augmented Generation).

Порівняльна характеристика підходів до використання великих мовних моделей наведена в таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика підходів до використання LLM

Характеристика	Хмарні сервіси (ChatGPT)	Локальні платформи (LM Studio)	Розроблюваний веб-застосунок
Приватність даних	Низька (дані обробляються на сервері)	Висока (дані на пристрої)	Максимальна (Air-gapped середовище)
Залежність від Інтернету	Повна	Відсутня (працює офлайн)	Відсутня (працює офлайн)

Продовження таблиці 1.1

Характеристика	Хмарні сервіси (ChatGPT)	Локальні платформи (LM Studio)	Розроблюваний веб-застосунок
Вимоги до ресурсів	Мінімальні (браузер)	Високі (часто завантажує всю RAM)	Оптимізовані (використання GGUF)
Інтеграція бази знань (RAG)	Є, але файли завантажуються в хмару	Обмежена (потребує налаштувань)	Вбудована локальна векторна СУБД
Складність інтерфейсу	Низька	Висока (орієнтовано на розробників)	Низька (мінімалістичний дизайн)

Аналіз існуючих рішень та методів оптимізації показав, що найбільш доцільним для досягнення мети кваліфікаційної роботи є розробка власного веб-застосунку, який поєднує метод квантизації (формат GGUF) та інтегровану підсистему семантичного пошуку по документах. Це дозволить створити швидку, автономну систему з інтуїтивно зрозумілим інтерфейсом, яка перевершує існуючі аналоги за критеріями безпеки даних та зручності роботи з локальними базами знань.

1.3 Аналіз вимог до програмного забезпення

На основі проведеного аналізу предметної області, а також виявлених недоліків існуючих рішень для локального запуску нейронних мереж, було сформовано комплексний перелік вимог до розроблюваного програмного

забезпечення. Метою етапу аналізу вимог є чітке визначення того, що саме повинна робити система та в яких технічних умовах вона має функціонувати.

Усі вимоги до розроблюваного веб-застосунку класифіковано на дві основні категорії: функціональні (визначають поведінку та можливості системи) та нефункціональні (визначають атрибути якості, безпеки та апаратні обмеження).

а) функціональні вимоги – описують набір операцій, які веб-застосунок повинен виконувати для задоволення потреб кінцевого користувача:

1) підсистема управління діалогами (графічний інтерфейс):
   - забезпечення можливості створення нових сеансів зв'язку (чатів) та їх видалення;
   - автоматична генерація релевантних заголовків для нових діалогів на основі першого повідомлення користувача;
   - збереження контексту поточної бесіди в рамках однієї сесії;
   - підтримка потокового (streaming) виведення тексту на екран для імітації природного набору відповіді;
2) підсистема обчислювального ядра штучного інтелекту:
   - завантаження локальної мовної моделі у форматі GGUF в оперативну пам'ять;
   - обробка текстових запитів природною мовою та генерація контекстно-залежних відповідей;
   - надання користувачеві можливості налаштування параметрів інференсу (регулювання «температури» генерації та максимальної кількості токенів);
3) підсистема роботи з локальною базою знань (технологія RAG):
   - підтримка завантаження користувацьких документів у форматах PDF, DOCX та TXT через графічний інтерфейс веб-застосунку;

– автоматичний парсинг (вилучення тексту) із завантажених файлів;
– фрагментація тексту та створення векторних представлень (ембедингів);
– збереження векторних даних у локальній базі даних;
– виконання семантичного пошуку релевантної інформації у базі даних при отриманні запиту та передача знайденого контексту до мовної моделі;

4) підсистема роботи з файловою системою (пам'ять застосунку):
– збереження історії всіх діалогів у форматі JSON;
– автоматичне завантаження списку попередніх сесій під час старту програми;

б) нефункціональні вимоги – визначають системні характеристики веб-застосунку, обмеження середовища виконання та критерії якості програмного коду:

1) вимоги до апаратного забезпечення та продуктивності:
– здатність системи функціонувати в умовах обмеженого обсягу оперативної пам'яті (від 8 ГБ RAM) завдяки використанню 4-бітної квантизації;
– підтримка апаратного прискорення обчислень (наприклад, інтеграція з Metal Performance Shaders для ефективної роботи на процесорах Apple Silicon);
– мінімізація часу відгуку системи на запит користувача;

2) вимоги до безпеки та конфіденційності:
– забезпечення повної мережевої автономності (здатність функціонувати в ізольованому Air-gapped середовищі без доступу до Інтернету);

– заборона на передачу будь-яких користувацьких даних, запитів або фрагментів документів на зовнішні сервери чи сторонні API;

3) вимоги до архітектури та зручності супроводу:
   – застосування модульної архітектури коду для забезпечення високої масштабованості та легкого додавання нових функцій у майбутньому;
   – ізоляція логіки роботи з інтерфейсом від логіки обробки даних (відокремлення обчислювального ядра від презентаційного рівня);
4) вимоги до інтерфейсу користувача (UI/UX):
   – забезпечення інтуїтивно зрозумілого мінімалістичного дизайну, що не перевантажує користувача надлишковими технічними налаштуваннями;
   – адаптивність графічних елементів для зручного відображення на різних розмірах екранів робочих станцій.

Сформований перелік вимог є основою для подальшого проєктування архітектури веб-застосунку та вибору технологічного стеку. Дотримання цих вимог гарантує створення автономного, безпечного та оптимізованого програмного продукту, що повністю відповідає поставленій меті.

1.4 Теоретичні засади технології пошукової генерації (RAG)

Головною проблемою будь-якої локальної мовної моделі, особливо невеликого розміру (до 10 мільярдів параметрів), є обмеженість її фактологічної бази знань та відсутність доступу до актуальної інформації. Крім того, таким моделям притаманний ефект «галюцинацій» — генерація правдоподібного, але

хибного тексту при відповіді на питання, яке виходить за межі їхнього навчального набору даних.

Для подолання цих обмежень у сучасній інженерії програмного забезпечення застосовується архітектура Retrieval-Augmented Generation (RAG). Цей підхід поєднує можливості традиційного інформаційного пошуку (Retrieval) із генеративними здібностями мовних моделей (Generation).

Класичний конвеєр RAG складається з трьох ключових етапів:

- індексація (Indexing): Сирі текстові дані з документів користувача очищуються, розбиваються на менші семантичні фрагменти (чанки) та перетворюються на числові вектори (ембедінги) за допомогою спеціалізованої моделі-векторизатора. Отримані вектори зберігаються у векторній базі даних;
- пошук (Retrieval): Коли користувач робить запит, система перетворює його на такий самий вектор і шукає у базі даних найбільш схожі за семантикою фрагменти (зазвичай використовується метрика косинусної подібності — Cosine Similarity);
- генерація (Generation): Знайдені релевантні фрагменти тексту «підмішуються» у системний промпт (Prompt) разом із запитом користувача. Модель отримує пряму інструкцію формувати відповідь, спираючись виключно на наданий контекст.

Впровадження RAG дозволяє перетворити мовну модель із простого генератора тексту на потужний аналітичний інструмент для роботи з конфіденційною документацією, зберігаючи при цьому всі переваги автономного функціонування.

1.5 Висновки по дослідженню предметної області та постановки задачі

У першому розділі було проведено комплексний аналіз предметної області, пов’язаної з розвитком інтерфейсів взаємодії людини та комп'ютера за допомогою природної мови. Встановлено, що незважаючи на стрімкий розвиток великих мовних моделей (LLM), домінуюча хмарна архітектура їх розгортання створює критичні ризики для конфіденційності даних та унеможливлює їх використання в ізольованих (Air-gapped) середовищах.

Аналіз існуючих рішень показав, що ринок потребує переходу до концепції Edge AI – виконання нейромережевих обчислень безпосередньо на кінцевих автономних пристроях. Дослідження методів оптимізації підтвердило, що використання 4-бітної квантизації (зокрема формату GGUF) та спеціалізованого апаратного прискорення дозволяє ефективно запускати потужні моделі на споживчому обладнанні з обмеженим обсягом оперативної пам'яті. Крім того, виявлено потребу в інтеграції технології пошукової генерації (RAG) для забезпечення взаємодії ШІ з локальними базами знань користувача без підключення до Інтернету.

На основі проведеного аналізу сформовано детальний перелік функціональних та нефункціональних вимог до програмного забезпечення, що проєктується. Визначено, що система повинна мати модульну архітектуру, мінімалістичний інтерфейс та забезпечувати високий рівень захисту даних.

Метою кваліфікаційної роботи є підвищення рівня конфіденційності та автономності процесів обробки природної мови шляхом розробки оптимізованого локального веб-застосунку на базі великої мовної моделі з інтегрованою підсистемою аналізу локальних документів.

Об’єктом дослідження є процес людино-машинної взаємодії із застосуванням технологій обробки природної мови в умовах апаратних обмежень.

Предметом дослідження є методи оптимізації, квантизації та локального розгортання великих мовних моделей, а також алгоритми пошукової генерації (RAG) для роботи з користувацькими документами.

Для досягнення поставленої мети необхідно виконати такі завдання:

а) проаналізувати предметну область, існуючі програмні рішення та методи компресії нейронних мереж для роботи на автономних пристроях;

б) сформувати вимоги до програмного забезпечення та обґрунтувати вибір технологічного стеку розробки;

в) спроєктувати модульну архітектуру локального веб-застосунку, базу даних для збереження історій сесій та векторну базу знань;

г) розробити програмний комплекс, що включає:

1) презентаційний рівень (графічний інтерфейс користувача);
2) підсистему управління пам'яттю та станом;
3) обчислювальне ядро для завантаження квантованих моделей та генерації тексту;
4) підсистему інтелектуального аналізу локальних документів (RAG);

д) провести тестування розробленого веб-застосунку на предмет коректності виконання функціональних вимог, стабільності роботи та споживання апаратних ресурсів.

2. ПРОЄКТУВАННЯ СТРУКТУРИ ТА КОМПОНЕНТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Проєктування архітектури програмного забезпечення

Архітектура розробленого програмного комплексу побудована за модульним принципом із чітким логічним розділенням рівнів відповідальності [14]. Оскільки головною вимогою до системи є функціонування на автономних комп'ютерних пристроях в умовах повної ізоляції від глобальної мережі Інтернет (Air-gapped середовище), архітектура не передбачає клієнт-серверної взаємодії із зовнішніми хмарними API. Усі обчислювальні компоненти, включаючи ядро великої мовної моделі та векторну базу даних, розгортаються локально.

Загальна структура розробленого локального веб-застосунку складається з п'яти ключових взаємопов'язаних модулів, кожен з яких інкапсулює специфічний набір функцій:

а) презентаційний рівень (головний модуль керування app.py), який виконує роль точки входу в систему та оркестратора процесів. Цей компонент відповідає за відображення графічного інтерфейсу користувача (ГІК) веб-застосунку та включає:

1) рендеринг веб-сторінки та інтерактивних елементів;
2) управління станом сесії (Session State) для збереження контексту діалогу в реальному часі;
3) обробку користувацького введення:
   – введення текстових запитів;
   – взаємодія з елементами управління;
4) забезпечення завантаження локальних файлів для подальшого аналізу;
5) потокове виведення згенерованих відповідей штучного інтелекту;

б) рівень конфігурації та стилізації (модуль config.py), що є єдиним центром управління глобальними параметрами веб-застосунку. Він містить:

1) шляхи до системних директорій:
   – папки з історією чатів;
   – директорії збережених документів;
   – локальної векторної бази даних;
2) ін'єкції кастомних каскадних таблиць стилів (CSS), що забезпечують адаптивність інтерфейсу;

в) підсистема управління даними та станом (модуль storage.py), яка відповідає за персистентність даних між сеансами роботи веб-застосунку та реалізує:

1) повний цикл CRUD-операцій для роботи з файловою системою;
2) збереження історії кожного діалогу у форматі JSON для швидкого доступу до контексту попередніх бесід;

г) обчислювальне ядро штучного інтелекту (модуль llm_engine.py), що є ключовим інтелектуальним компонентом системи і керує локальною великою мовною моделлю у квантованому форматі GGUF. До його функцій належать:

1) ініціалізація моделі в оперативній пам'яті пристрою з максимальною оптимізацією;
2) формування структурованих системних запитів (промптів);
3) генерація відповідей з урахуванням:
   – налаштувань температури (креативності);
   – лімітів генерованих токенів;
4) автоматична генерація контекстних заголовків для нових сеансів зв'язку;

д) підсистема пошукової генерації та аналізу документів (модуль rag_engine.py), яка забезпечує здатність веб-застосунку працювати з локальними документами. Процес включає:

1) автоматичний парсинг тексту завантаженого файлу (формати PDF, DOCX, TXT);
2) поділ тексту на семантичні фрагменти (чанки);
3) перетворення тексту у векторні представлення (ембедінги);
4) збереження векторів у локальній системі управління базами даних;
5) здійснення семантичного пошуку найрелевантніших фрагментів під час виконання запиту.

Запропонована архітектурна модель формує замкнений, безпечний цикл обробки інформації: від отримання запиту користувача через презентаційний рівень до збагачення його фактами з бази знань та генерації відповіді виключно на локальних обчислювальних потужностях автономного пристрою.

Для наочного відображення взаємодії описаних компонентів розроблено загальну структурну схему архітектури веб-застосунку, яка демонструє потік управління та передачу даних між підсистемами (Рисунок 2.1).

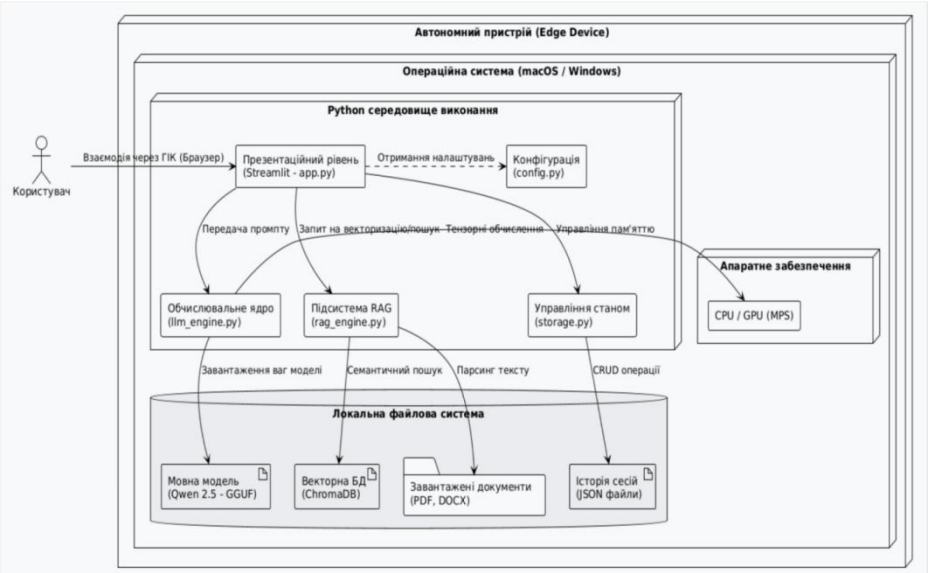


Рисунок 2.1 – Структурна схема архітектури розробленого веб-застосунку

Життєвий цикл обробки типового запиту в межах спроєктованої архітектури відбувається за таким сценарієм:

- ініціалізація та введення – користувач через презентаційний рівень (app.py) вводить текстовий запит у графічному інтерфейсі;
- перевірка контексту (RAG) – запит маршрутизується до підсистеми аналізу документів (rag_engine.py), яка перетворює його на вектор і виконує пошук у локальній базі знань. Якщо релевантні фрагменти знайдено, вони вилучаються для подальшого використання;
- формування промпту – оригінальний запит користувача, знайдений контекст із документів та історія попередніх повідомлень із модуля пам'яті (storage.py) передаються до обчислювального ядра (llm_engine.py);
- генерація та потокове виведення – обчислювальне ядро застосовує системні інструкції, формує фінальний ChatML-промпт і передає його локальній мовній моделі. Модель генерує відповідь по одному токену, які асинхронно повертаються на презентаційний рівень для миттєвого відображення;
- персистентність – після завершення генерації оновлений масив діалогу передається до підсистеми управління даними (storage.py) для атомарного запису на накопичувач.

Для деталізації фізичного розміщення компонентів системи та їх взаємодії з апаратним забезпеченням розроблено діаграму розгортання (Deployment Diagram). Оскільки веб-застосунок спроєктовано для роботи в автономному середовищі, усі вузли обробки та зберігання даних сконцентровані на одному фізичному пристрої користувача. Опис архітектури розгортання включає такі рівні:

а) рівень апаратного забезпечення (Hardware Node):

1) центральний процесор (CPU) для виконання загальної логіки веб-застосунку;

2) обчислювальний співпроцесор (GPU/MPS) для прискорення тензорних операцій мовної моделі;
3) оперативна пам'ять (RAM), де розміщуються активні ваги квантованої моделі та кеш сесії;

б) рівень операційної системи та середовища виконання:

1) локальна файлова система, що забезпечує довготривале зберігання даних;
2) інтерпретатор мови Python із завантаженими бібліотеками;

в) рівень програмних артефактів:

1) модулі веб-застосунку (app.py, llm_engine.py тощо);
2) бінарний файл локальної мовної моделі у форматі GGUF.

Окрему увагу на етапі проєктування приділено структурі зберігання даних. Для забезпечення персистентності діалогів та можливості їх подальшого аналізу було розроблено схему зберігання історії у форматі JSON. Кожна сесія користувача представляє собою структурований масив об'єктів.

Опис ключів, що використовуються для формування структури збереження повідомлень, наведено в таблиці 2.1.

Таблиця 2.1 – Структура об'єктів JSON-файлу збереження сесії

Ключове поле	Тип даних	Призначення та опис
role	String	Визначає автора репліки. Приймає значення: user (користувач), assistant (модель) або system (системний контекст).

Продовження таблиці 2.1

content	String	Містить безпосередній текстовий вміст повідомлення, згенеровану відповідь або вилучений з документів контекст.
timestamp	String	Часова мітка збереження повідомлення (використовується для сортування історії діалогів).

Приклад серіалізованого об'єкта, що зберігається у файловій системі веб-застосунку, наведено в лістингу 2.1.

Лістинг 2.1 – Приклад структури JSON-об'єкта збереження історії сесії:

```
[
  {
    "role": "system",
    "content": "Ти корисний ШІ-асистент, який відповідає українською мовою.",
    "timestamp": "2026-03-20T10:15:30"
  },
  {
    "role": "user",
    "content": "Що таке RAG?",
    "timestamp": "2026-03-20T10:16:05"
  }
]
```

Такий підхід дозволяє веб-застосунку швидко зчитувати контекст попередніх повідомлень та передавати його до обчислювального ядра під час генерації кожної нової відповіді.

Важливим компонентом архітектури є організація векторного сховища в системі ChromaDB [21], [22]. На відміну від традиційних реляційних баз даних, векторна СУБД спроєктована для збереження семантичних зв'язків. Процес організації та пошуку даних включає:

- векторизацію тексту – кожен фрагмент завантаженого документа перетворюється у багатовимірний масив числових значень (ембедінг), що відображає його смислове значення;
- просторову індексацію – збережені дані групуються таким чином, щоб семантично близькі за змістом тексти знаходилися поруч у структурі бази даних;
- семантичне зіставлення – для вилучення релевантної інформації система алгоритмічно порівнює числовий запит користувача з даними документів, обираючи ті фрагменти, які мають найвищий показник змістової подібності.

Така багаторівнева архітектура забезпечує високу швидкість обробки інформації та гарантує, що веб-застосунок залишається повністю функціональним навіть при значних обсягах локальних документів, не перевантажуючи інші рівні системи.

2.2 Проєктування модулів

На етапі проєктування внутрішньої структури веб-застосунку було проведено декомпозицію загальної системи на незалежні функціональні одиниці (модулі). Кожен модуль спроєктовано таким чином, щоб мінімізувати зв'язність коду та забезпечити високий рівень інкапсуляції логіки. Взаємодія між модулями відбувається через чітко визначені програмні інтерфейси та виклики функцій.

Детальне проєктування модулів системи включає такі структурні елементи:

а) модуль управління базовими налаштуваннями (на базі конфігураційного файлу config.py):

1) спроєктовано як статичний конфігураційний файл, що ініціалізується під час запуску системи;
2) містить логіку автоматичної перевірки наявності необхідних системних директорій та їх створення у разі відсутності;
3) інкапсулює візуальні налаштування у вигляді блоків коду, які передаються на презентаційний рівень для рендерингу;

б) підсистема управління станом та пам'яттю (модуль storage.py):

1) спроєктовано набір функцій для роботи з локальною базою чатів:
   – функція читання історії діалогів, що виконує десеріалізацію даних та передає масив об'єктів у оперативну пам'ять;
   – функція збереження поточного стану, яка серіалізує масив повідомлень та виконує атомарний запис на накопичувач;
   – функція безпечного видалення сесій користувача;
2) реалізовано алгоритм безпечного перейменування файлів для уникнення конфліктів у файловій системі під час одночасного створення кількох сесій;

в) модуль генеративного ядра штучного інтелекту (модуль llm_engine.py):

1) спроєктовано механізм одноразового завантаження ваг мовної моделі в оперативну пам'ять пристрою із застосуванням патерну кешування ресурсів, що запобігає повторній ініціалізації моделі під час кожного запиту;
2) розроблено алгоритм форматування вхідних даних:
   – отримання необробленого тексту від користувача;
   – додавання системних інструкцій (ролі) до початку повідомлення;
   – конкатенація останніх реплік для збереження контексту бесіди;

– обгортання тексту в специфічні керівні токени (ChatML-формат), необхідні для коректної роботи локальної моделі;

3) реалізовано функцію потокової (streaming) генерації токенів, яка повертає текстові фрагменти асинхронно в міру їх обчислення процесором автономного пристрою;

г) модуль інтелектуального аналізу локальних документів, або RAG система (модуль rag_engine.py):

1) спроєктовано конвеєр обробки вхідних файлів, що складається з таких етапів:
   – ідентифікація формату завантаженого документа;
   – виклик відповідного алгоритму екстракції тексту;
   – сегментація суцільного тексту на фрагменти фіксованого розміру з урахуванням перекриття символів для збереження семантичної цілісності речень;
2) спроєктовано підсистему семантичного пошуку:
   – функція генерації векторних представлень для текстових фрагментів;
   – алгоритм обчислення косинусної подібності між вектором запиту користувача та векторами бази знань;
   – вилучення найрелевантніших фрагментів та їх передача до генеративного ядра;

д) презентаційний модуль, що відповідає за інтерфейс веб-застосунку (модуль app.py):

1) спроєктовано життєвий цикл сторінки веб-застосунку, який оновлюється за принципом реактивності;
2) розроблено структуру поділу екрана:
   – бічна панель для навігації між сесіями, управління базою знань та налаштування параметрів інференсу моделі;

– основна робоча область для відображення поточного діалогу та поля введення запитів;

3) імплементовано алгоритм перехоплення помилок генерації з їх подальшим безпечним виведенням у графічному інтерфейсі користувача.

Спроєктована структура модулів забезпечує надійне функціонування локальної моделі та дозволяє легко розширювати функціонал веб-застосунку без необхідності рефакторингу всього програмного коду. Взаємоізоляція логіки (наприклад, відокремлення інтерфейсу від векторного пошуку) відповідає сучасним інженерним стандартам розробки надійного програмного забезпечення.

Детальне проєктування модуля інтелектуального аналізу локальних документів (RAG-системи) вимагає формалізації алгоритму обробки вхідних даних та їх підготовки для семантичного пошуку. Алгоритм функціонування підсистеми rag_engine.py концептуально поділяється на два незалежні процеси: процес індексування локальної бази знань та процес пошукової вибірки.

Логіка виконання процесу індексування складається з таких послідовних кроків:

а) етап завантаження та парсингу (Data Ingestion):

1) отримання бінарного файлу документа (допустимі формати: PDF, DOCX, TXT) через презентаційний рівень;
2) вилучення неструктурованого тексту за допомогою спеціалізованих програмних екстракторів з очищенням від службових символів;

б) етап фрагментації тексту (Chunking):

1) поділ суцільного масиву тексту на смислові фрагменти (чанки) фіксованого розміру (наприклад, по 1000 токенів);
2) обов'язкове застосування лінійного перекриття (overlap) розміром 200 токенів між сусідніми фрагментами, що гарантує збереження семантичного контексту на стиках абзаців або складних речень;

в) етап векторизації (Embedding) та збереження:

1) перетворення кожного текстового фрагмента у багатовимірний числовий вектор за допомогою легкої локальної моделі векторизації [23], [24];
2) атомарний запис згенерованих векторів разом із відповідними метаданими (назва вихідного файлу, ідентифікатор фрагмента) у локальну базу даних ChromaDB.

Процес пошукової вибірки (Retrieval) активується лише під час надходження запиту від користувача і включає:

- перехоплення текстового запиту та його миттєва векторизація з використанням ідентичної моделі ембедингів;
- виконання семантичного пошуку (пошуку найближчих математичних сусідів) у просторі ChromaDB;
- вилучення заданої кількості (Top-K) найрелевантніших текстових фрагментів, які мають найвищий показник змістової подібності із запитом;
- конкатенація знайдених фрагментів та їх передача до генеративного ядра ШІ як доповненого контексту.

Блок-схема спроєктованого алгоритму функціонування підсистеми інтелектуального аналізу документів наведена на рисунку 2.3.

Спроєктований алгоритм забезпечує високу стійкість системи до обробки великих обсягів тексту. Завдяки механізму фрагментації з перекриттям, мовна модель отримує максимально точний і нерозривний контекст, що критично знижує ймовірність генерації логічних або фактологічних помилок (галюцинацій) під час надання відповіді.

Рисунок 2.3 – Блок-схема алгоритму обробки документів та пошукової генерації (RAG)

2.3 Проєктування інтерфейсу користувача

На етапі проєктування графічного інтерфейсу користувача (ГІК) головним завданням було створення інтуїтивно зрозумілого, ергономічного середовища, яке приховує складність внутрішніх нейромережевих обчислень від кінцевого користувача. Оскільки розроблений локальний веб-застосунок призначений для автономної роботи з документами та генерації тексту, його інтерфейс

спроєктовано за класичним патерном сучасних ШІ-асистентів, що мінімізує когнітивне навантаження.

Проєктування візуальної структури веб-застосунку базується на поділі екранного простору на дві основні функціональні зони:

а) бічна панель (Sidebar), яка виконує роль головного навігаційного та конфігураційного меню і містить:

1) блок управління сесіями:
   – акцентована кнопка створення нового діалогу;
   – динамічний перелік історичних сесій із згенерованими контекстними заголовками, відсортований за датою створення;
2) зону інтелектуального аналізу локальних документів (RAG-модуль):
   – інтерактивний віджет типу Drag-and-Drop для завантаження файлів форматів PDF, DOCX та TXT;
   – візуальні індикатори стану (спінери обробки), які інформують користувача про процес векторизації документа;
3) блок технічних налаштувань (прихований у випадаючому контейнері expander, щоб не перевантажувати базовий інтерфейс):
   – повзунок регулювання «температури» генерації (відповідає за варіативність та креативність відповідей моделі);
   – регулятор ліміту токенів для контролю максимальної довжини згенерованого тексту;

б) основна робоча область (Main Chat Area), призначена для безпосередньої взаємодії з мовною моделлю та відображення контенту:

1) заголовок сесії та візуальні акценти (використання іконок для «пожвавлення» інтерфейсу);
2) контейнер історії повідомлень, який реалізує чітке візуальне розмежування між репліками:

– повідомлення користувача відображаються з відповідним аватаром та вирівнюванням;
– відповіді системи генеруються у потоковому режимі з використанням стилізованих «бульбашок» чату (chat bubbles);

3) фіксоване поле введення текстових запитів у нижній частині екрана, яке підтримує багаторядкове введення та автоматично блокується під час генерації відповіді для запобігання конфліктам стану.

Важливим аспектом проєктування ГІК став вибір кольорової палітри. Для забезпечення комфортної тривалої роботи з текстом та уникнення візуального конфлікту між світлим фоном сторінки та темними елементами чату, веб-застосунок спроєктовано з використанням єдиної темної теми (Dark Mode). Темно-сірий фон сторінки у поєднанні з контрастними (але не агресивними) світлими шрифтами знижує навантаження на зір користувача та надає програмному продукту сучасного вигляду.

Для підтвердження реалізації спроєктованого інтерфейсу розроблено відповідну екранну форму (Рисунок 2.2).

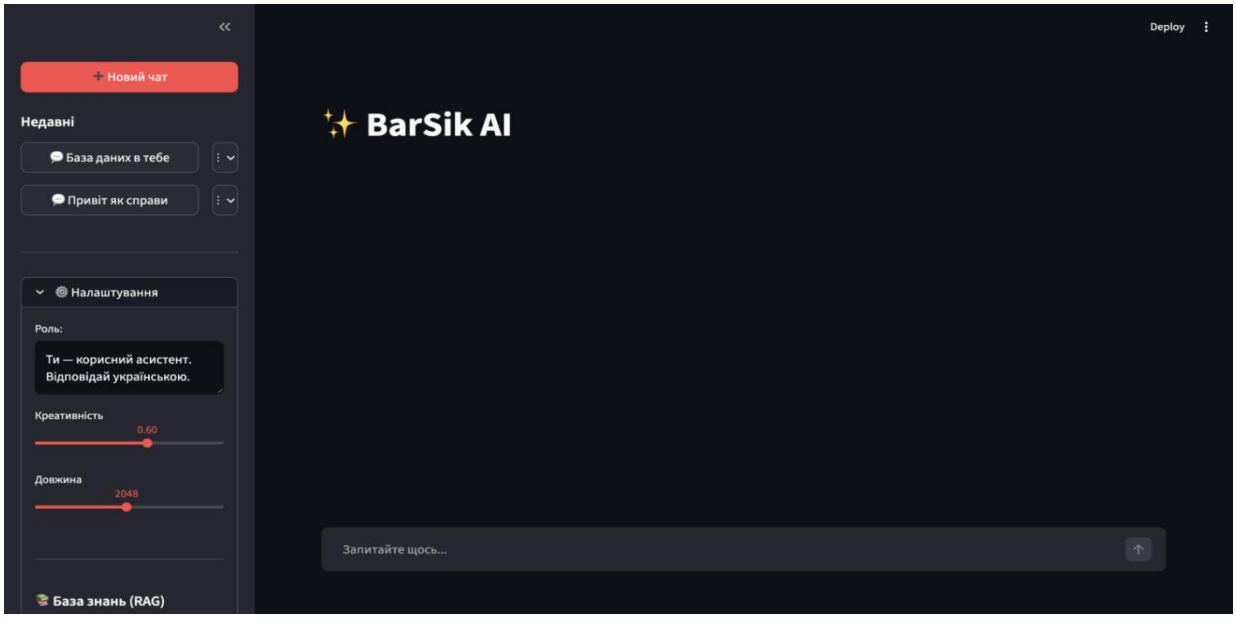


Рисунок 2.2 –Графічний інтерфейс розробленого локального веб-застосунку

Реактивна архітектура інтерфейсу забезпечує миттєве оновлення візуальних елементів без необхідності перезавантаження всієї сторінки. Це створює ефект безперервної, плавної взаємодії, ідентичної до найкращих хмарних аналогів, але з гарантією повної локальної приватності даних користувача.

Для формалізації логіки взаємодії користувача з веб-застосунком було спроєктовано діаграму прецедентів (Use Case Diagram). Вона дозволяє визначити межі системи та основні сценарії її використання (Рисунок 2.4).

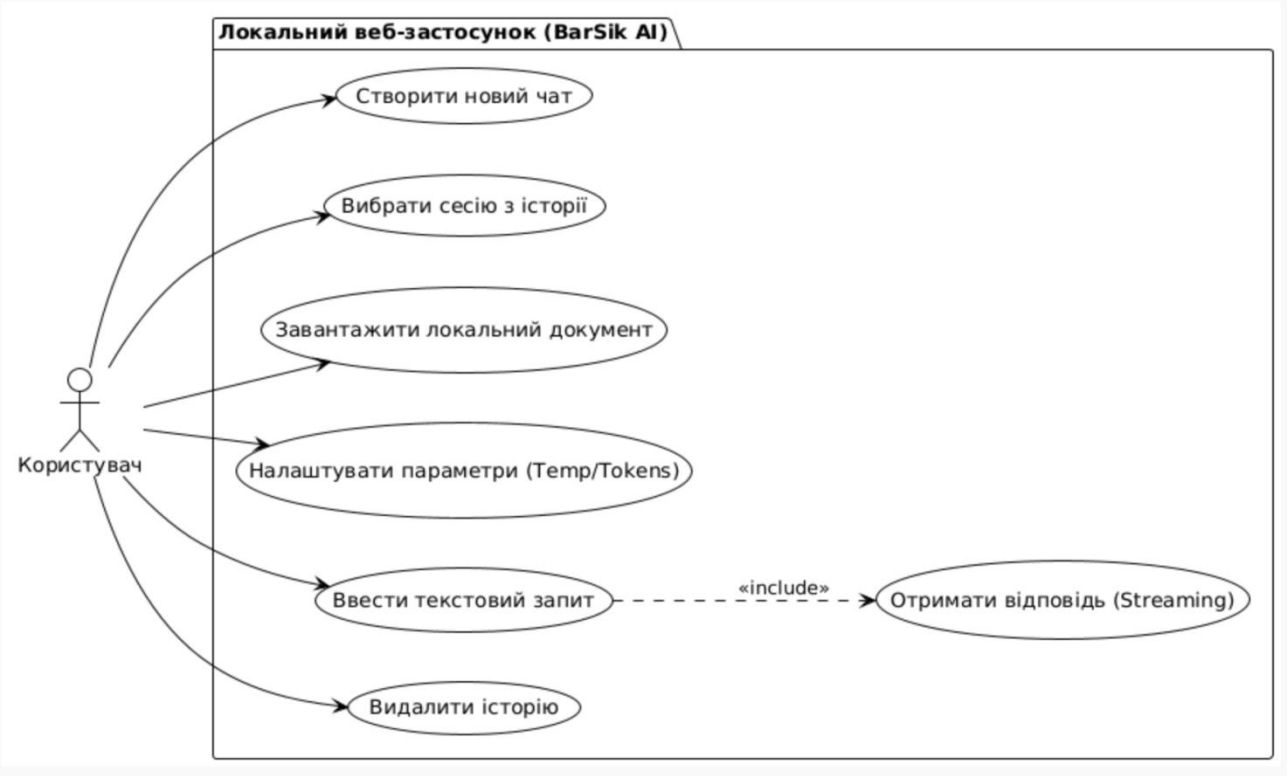


Рисунок 2.4 – Діаграма прецедентів веб-застосунку

Згідно з розробленою діаграмою, основна роль (Актор) – Користувач – взаємодіє з системою через такі ключові сценарії:

- управління контекстом – створення, вибір та видалення діалогів;
- робота з базою знань – завантаження локальних файлів та ініціалізація процесу векторизації;

– взаємодія з моделлю – введення текстових запитів та налаштування параметрів генерації (температура, ліміт токенів).

Важливим елементом проєктування інтерфейсу став аналіз станів системи під час виконання тривалих операцій (інференс моделі або векторизація великих документів). Для забезпечення якісного користувацького досвіду (UX) спроєктовано систему візуального фідбеку, опис якої наведено в таблиці 2.2.

Таблиця 2.2 – Опис станів інтерфейсу та візуальної індикації

Стан системи	Дія користувача	Візуальна реакція інтерфейсу
Очікування	Система готова до роботи	Активне поле введення, доступний сайдбар.
Векторизація	Завантаження документа	Відображення анімованого індикатора (spinner) з текстом «Обробка файлу...».
Генерація	Надіслано запит	Поява блоку відповіді, потокове виведення тексту (typing effect), блокування кнопки відправки.
Помилка	Некоректний файл або збій моделі	Виведення повідомлення у червоному діалоговому вікні з описом проблеми.

Такий підхід до проєктування гарантує, що користувач завжди розуміє поточний стан автономного пристрою, навіть якщо обчислення займають тривалий час через апаратні обмеження.

2.4 Аналіз технологій і методів реалізації ПЗ

На етапі проєктування програмного забезпечення критично важливим кроком є вибір технологічного стеку, який повинен не лише забезпечувати реалізацію закладеного функціоналу, але й відповідати жорстким нефункціональним вимогам щодо автономності, споживання пам'яті та швидкодії в умовах обмежених апаратних ресурсів.

Після проведення порівняльного аналізу існуючих інструментальних засобів для розробки локального веб-застосунку було обрано наступний стек технологій:

а) високорівнева мова програмування Python, яка де-факто є галузевим стандартом у сфері машинного навчання (Machine Learning) та штучного інтелекту [25], [26], [27]. Вибір мови Python обґрунтовується такими факторами:

1) наявність найширшої екосистеми спеціалізованих бібліотек для обробки природної мови (NLP) та роботи з нейронними мережами;
2) висока швидкість прототипування та розробки завдяки лаконічному синтаксису;
3) можливість гнучкої інтеграції низькорівневих C/C++ бібліотек для прискорення математичних обчислень;

б) фреймворк Streamlit, обраний для побудови презентаційного рівня (ГІК) веб-застосунку. На відміну від традиційних веб-фреймворків (наприклад, Django або Flask), Streamlit оптимізований спеціально для проєктів у галузі Data Science та машинного навчання. Його ключові переваги для даної роботи:

1) декларативний підхід до створення графічного інтерфейсу виключно засобами мови Python без необхідності написання окремого коду HTML, CSS чи JavaScript;

2) вбудована реактивна модель управління станом (Session State), яка ідеально підходить для збереження історії діалогу між ітераціями запитів;
3) нативна підтримка потокового (streaming) виведення тексту, що є критично важливим для відображення процесу по-токенної генерації відповідей мовною моделлю;

в) бібліотека llama.cpp (з використанням Python-обгортки llama-cpp-python), яка виступає в ролі обчислювального ядра ШІ [28], [29]. Цей інструмент обрано через його унікальні оптимізаційні характеристики:
1) повна підтримка формату квантованих ваг GGUF, що дозволяє радикально зменшити обсяг оперативної пам'яті, необхідної для завантаження моделі (наприклад, запуск моделі параметричністю 8B на пристроях з 8 ГБ RAM);
2) високоефективне використання апаратного прискорення безпосередньо на центральних процесорах (CPU);
3) інтеграція з технологією Metal Performance Shaders (MPS), що забезпечує максимальну продуктивність інференсу на пристроях з архітектурою Apple Silicon;

г) екосистема локального пошуку та векторизації для підсистеми RAG, яка базується на двох ключових інструментах:
1) фреймворк LangChain – використовується для побудови конвеєра обробки локальних документів, що включає:
   – парсинг тексту з файлів різних форматів (PDF, DOCX, TXT);
   – семантичне розбиття тексту на фрагменти (чанки) з урахуванням перекриття символів для збереження контексту речень;
2) локальна векторна система управління базами даних ChromaDB – обрана як сховище для ембедингів (векторних представлень тексту). Її головні переваги:

– повна автономність (база даних розгортається локально у вигляді файлів та не потребує розгортання окремого серверного процесу чи підключення до Інтернету);
– забезпечення високої швидкості обчислення косинусної подібності під час виконання семантичного пошуку найрелевантніших фрагментів тексту.

Комплексне використання зазначених технологій формує надійний, безпечний та оптимізований фундамент для розробки. Обраний технологічний стек повністю задовольняє вимоги до створення автономного локального веб-застосунку, здатного функціонувати в Air-gapped середовищах без втрати швидкодії та якості інтелектуальної генерації.

Окремим етапом аналізу було обґрунтування вибору методу квантизації для моделі Qwen 2.5. Оскільки цільові автономні пристрої мають обмежений обсяг оперативної пам'яті, було проведено порівняння різних рівнів стиснення формату GGUF (Таблиця 2.3) [15], [16], [17], [18].

Таблиця 2.3 – Порівняння рівнів квантизації для моделі 7B параметрів

Метод квантизації	Розмір файлу (ГБ)	Вимоги до RAM	Втрата якості (Perplexity)
F16 (без стиснення)	~15.0	>16 ГБ	0% (еталон)
Q8_0 (8-біт)	~7.7	~10 ГБ	<0.01%
Q4_K_M (4-біт)	~4.8	~6 ГБ	<0.5% (Оптимально)
Q2_K (2-біт)	~2.8	~4 ГБ	>10% (неприпустимо)

На основі аналізу обрано метод Q4_K_M. Він забезпечує найкращий баланс між збереженням інтелектуальних здібностей моделі та можливістю її запуску на пристроях з 8 ГБ оперативної пам'яті, залишаючи ресурс для роботи операційної системи та презентаційного шару (Streamlit).

Також було проаналізовано механізм взаємодії Python-середовища з апаратним прискоренням. Для пристроїв на базі Apple Silicon реалізовано підтримку бекенду MPS (Metal Performance Shaders), що дозволяє виконувати множення матриць безпосередньо на графічному ядрі (GPU) [19], [20]. Це забезпечує приріст швидкості генерації у 3–5 разів порівняно з чистими обчисленнями на центральному процесорі (CPU).

Застосування екосистеми LangChain дозволило абстрагувати логіку роботи з документами, що робить архітектуру веб-застосунку стійкою до зміни конкретних моделей векторизації або типів баз даних у майбутньому без необхідності переписування основного ядра системи.

2.5 Висновки по проєктуванню структури та компонентів ПЗ

У другому розділі було виконано комплексне проєктування архітектури, внутрішніх модулів та графічного інтерфейсу локального веб-застосунку на базі великої мовної моделі, а також обґрунтовано вибір програмно-технологічного стеку для його реалізації.

Підсумовуючи результати етапу проєктування, можна виділити такі ключові здобутки:

- спроєктовано надійну та безпечну архітектуру. Розроблена структурна схема та діаграма розгортання довели, що система здатна повністю функціонувати в ізольованому апаратному середовищі (Air-gapped) без звернення до зовнішніх хмарних серверів. Це повністю задовольняє головну нефункціональну вимогу — абсолютну конфіденційність оброблюваних користувацьких даних.

- здійснено глибоку декомпозицію системи на незалежні модулі. Використання модульного підходу дозволило чітко розмежувати логіку презентаційного рівня (ГІК), підсистеми управління пам'яттю (історія в JSON), обчислювального ядра LLM та підсистеми пошукової генерації (RAG). Детально спроєктований алгоритм векторизації документів та семантичного пошуку у локальній базі знань ChromaDB ефективно вирішує проблему відсутності доступу автономних моделей до актуальної зовнішньої інформації.
- створено концепцію інтуїтивно зрозумілого інтерфейсу. Розроблена діаграма прецедентів (Use Case) та макети графічного інтерфейсу підтвердили орієнтованість застосунку на кінцевого користувача. Впровадження системи візуального фідбеку (індикатори стану очікування, векторизації та потокової генерації) дозволило мінімізувати когнітивне навантаження та компенсувати можливі затримки, пов'язані з обмеженістю апаратних ресурсів.
- оптимізовано технологічний стек під апаратні обмеження. Здійснено ґрунтовний аналіз і вибір інструментальних засобів. Доведено, що використання мови Python у поєднанні з фреймворком Streamlit та оптимізованою бібліотекою llama.cpp є найкращим рішенням для Edge AI. Обґрунтовано вибір 4-бітної квантизації (метод Q4_K_M для формату GGUF) під архітектуру моделі Qwen 2.5, що забезпечує ідеальний баланс між високою якістю генерації та низьким споживанням оперативної пам'яті (до 6 ГБ). Додатково враховано можливість апаратного прискорення тензорних обчислень через технологію Metal Performance Shaders (MPS).

Загалом, розроблені UML-діаграми, схеми збереження даних та обрані програмні технології сформували надійний фундамент для стабільного функціонування інтелектуального асистента на споживчому обладнанні.

Спроєктована система є цілісною та повністю готовою до етапу безпосередньої програмної реалізації і тестування, що буде детально розглянуто в наступному розділі. Завдяки ґрунтовній проробці кожної підсистеми на етапі моделювання вдалося мінімізувати потенційні архітектурні ризики під час написання коду. Успішне втілення закладених проєктних рішень дозволить створити надійний, оптимізований програмний продукт, який повною мірою відповідає сучасним стандартам безпеки та ефективності обробки даних.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Програмна реалізація модулів

Етап програмної реалізації полягав у перетворенні спроєктованої архітектури на функціональний вихідний код мовою програмування Python. Розробка здійснювалася з дотриманням принципів модульності, де кожна підсистема інкапсульована в окремому файлі (модулі) для забезпечення зручності тестування та подальшого супроводу.

Основним етапом розробки стала реалізація обчислювального ядра в модулі llm_engine.py. Для взаємодії з квантованою моделлю Qwen 2.5 (формат GGUF) було використано бібліотеку llama-cpp-python. Ініціалізація моделі вимагала точного налаштування гіперпараметрів для забезпечення оптимального використання апаратних ресурсів (зокрема, делегування обчислень на графічний процесор).

Фрагмент коду ініціалізації мовної моделі наведено в лістингу 3.1.

Лістинг 3.1 – Ініціалізація локальної мовної моделі (llm_engine.py):

```
import os
import streamlit as st
from llama_cpp import Llama
from config import MODEL_PATH
@st.cache_resource
def load_model():
    if not os.path.exists(MODEL_PATH):
        st.error(f"Файл {MODEL_PATH} не знайдено! Перевір, чи лежить він у папці з проєктом.")
        return None
```

```
    return Llama(
        model_path=MODEL_PATH,
        n_ctx=2048,
        n_threads=6,
        n_gpu_layers=-1,
        verbose=False
    )
```

Функція load_model створює екземпляр класу Llama, який зберігається в кеші веб-застосунку (@st.cache_resource) для уникнення повторного завантаження ваг моделі в оперативну пам'ять при кожному новому запиті користувача.

Наступним кроком стала програмна реалізація підсистеми інтелектуального аналізу документів (RAG) у модулі rag_engine.py. Для реалізації конвеєра обробки документів було задіяно екосистему LangChain. Фрагмент реалізації процесу векторизації та семантичного пошуку наведено в лістингу 3.2.

Лістинг 3.2 – Фрагментація та семантичний пошук у документах (rag_engine.py):

```
    from langchain_community.document_loaders import PyPDFLoader, Docx2txtLoader, TextLoader
    from langchain_text_splitters import RecursiveCharacterTextSplitter
    from langchain_community.vectorstores import Chroma
    from config import VECTOR_DB_DIR
    def process_document(file_path, filename):
        ext = os.path.splitext(filename)[1].lower()
        if ext == '.pdf': loader = PyPDFLoader(file_path)
        elif ext == '.docx': loader = Docx2txtLoader(file_path)
```

```
    elif ext == '.txt': loader = TextLoader(file_path, encoding='utf-8')
    docs = loader.load()
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000,
chunk_overlap=200)
    splits = text_splitter.split_documents(docs)
    Chroma.from_documents(
        documents=splits,
        embedding=get_embeddings(),
        persist_directory=VECTOR_DB_DIR
    )
    return True
```

Для забезпечення гнучкості системи та уникнення жорсткого кодування (hardcoding) шляхів, усі глобальні параметри середовища було винесено в окремий конфігураційний модуль config.py. Фрагмент коду ініціалізації базових констант наведено в лістингу 3.3.

Лістинг 3.3 – Ініціалізація системних директорій (config.py):

```
import os
MODEL_PATH = "model.gguf"
CHATS_DIR = "chats"
UPLOAD_DIR = "uploads"
VECTOR_DB_DIR = "chroma_db"
for directory in [CHATS_DIR, UPLOAD_DIR, VECTOR_DB_DIR]:
    if not os.path.exists(directory):
        os.makedirs(directory)
```

За персистентність (збереження) контексту між сеансами відповідає модуль storage.py. Для реалізації бази даних історій було обрано формат JSON.

Цей модуль інкапсулює логіку читання, збереження, видалення та автоматичного перейменування файлів сесій.

Лістинг 3.4 – Реалізація збереження історії сесії (storage.py):

```
import json
import os
from config import CHATS_DIR
def save_chat(filename, messages):
    """Зберігає масив повідомлень чату у форматі JSON."""
    with open(os.path.join(CHATS_DIR, filename), "w", encoding="utf-8") as f:
        json.dump(messages, f, ensure_ascii=False, indent=4)
```

Для забезпечення цілісності розробки та зручності розгортання в Air-gapped середовищах було організовано чітку структуру файлової системи проєкту. Вона включає директорії для збереження векторної бази (chroma_db), завантажених користувачем документів (uploads) та історії діалогів (chats). Також до складу комплексу входять модулі для автоматизованого тестування (test_barsik.py, test_ui.py).

Підтвердженням реалізації структури модулів є дерево каталогів програмного комплексу (Рисунок 3.1).

Як видно з наведеної структури, кореневий каталог містить не лише файли з вихідним кодом, але й необхідні системні директорії. Зокрема, папка .venv містить ізольоване віртуальне середовище з усіма необхідними залежностями проєкту. Директорія chats призначена для збереження JSON-файлів з історією користувацьких сесій, що забезпечує персистентність діалогів. Папка uploads слугує тимчасовим сховищем для завантажених користувачем документів перед їхньою обробкою. Векторні представлення цих документів (ембединги) автоматично зберігаються у спеціалізованій директорії chroma_db, яка виконує роль локальної векторної бази даних. Основні логічні компоненти системи чітко

розділені по відповідних Python-скриптах, що повністю підтверджує спроєктовану раніше модульну архітектуру.

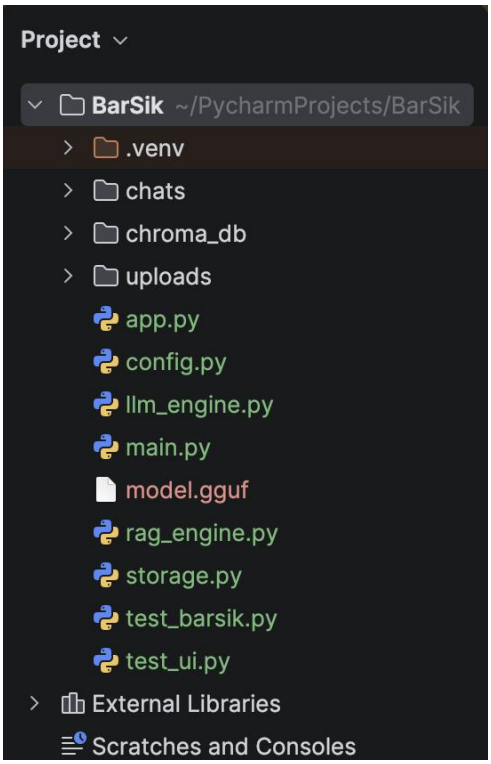


Рисунок 3.1 – Структура файлів та каталогів розробленого програмного комплексу

Фінальним етапом стала розробка презентаційного рівня (модуль app.py) з використанням фреймворку Streamlit. Головним завданням цього модуля була реалізація реактивного інтерфейсу, управління станом сесії (st.session_state) та динамічне підтягування контексту з бази RAG для збагачення запиту перед відправкою до LLM.

Комплексна інтеграція наведених модулів дозволила створити стабільний програмний продукт, повністю готовий до етапу тестування.

Окремої уваги заслуговує програмна реалізація механізму формування контекстно-залежних запитів до великої мовної моделі. Оскільки архітектура Qwen 2.5 вимагає суворого дотримання формату ChatML (Chat Markup Language) для розпізнавання ролей діалогу, у модулі llm_engine.py було розроблено

спеціальну функцію-конструктор промптів. Фрагмент реалізації цієї функції наведено в лістингу 3.5.

Лістинг 3.5 – Формування ChatML-промпту зі збереженням контексту (llm_engine.py):

```
def build_prompt(system_prompt, messages):
    """
    Формує текстовий промпт у форматі ChatML для моделі Qwen.
    Обмежує історію останньою кількістю повідомлень для економії пам'яті.
    """
    prompt_text = f"<|im_start|>system\n{system_prompt}<|im_end|>\n"
    for msg in messages[-6:]:
        prompt_text += f"<|im_start|>{msg['role']}\n{msg['content']}<|im_end|>\n"
    prompt_text += "<|im_start|>assistant\n"
    return prompt_text
```

Функція build_prompt ітерується по масиву messages, який зберігається в стані сесії. Важливим інженерним рішенням стало обмеження циклу (зріз messages[-6:]). Це гарантує, що модель аналізує лише останні три пари «запит-відповідь». Такий підхід запобігає переповненню контекстного вікна моделі (яке становить 2048 токенів) під час тривалих розмов та значно знижує споживання оперативної пам'яті під час виконання тензорних обчислень. Токени <|im_start|> та <|im_end|> виступають апаратними тригерами, які вказують нейронній мережі межі реплік.

Для покращення користувацького досвіду (UX) та зручної навігації в бічній панелі історії чатів, було імплементовано функцію інтелектуальної генерації заголовків. Вона використовує ту ж саму локальну мовну модель для

аналізу першого запиту користувача. Реалізацію цього алгоритму наведено в лістингу 3.6.

Лістинг 3.6 – Інтелектуальна генерація назви сесії (llm_engine.py):

```
def generate_smart_title(llm, user_text):
    """Генерує короткий заголовок (2-4 слова) на основі першого запиту."""
    if not llm:
        return "_".join(user_text.split()[:4])
    prompt = (
        f"<|im_start|>system\nПридумай заголовок (2-4 слова) до тексту. Без лапок.<|im_end|>\n"
        f"<|im_start|>user\nТекст: {user_text}<|im_end|>\n<|im_start|>assistant\n"
    )
    try:
        output = llm(prompt, max_tokens=15, stop=["<|im_end|>", "\n"], temperature=0.5)
        return output['choices'][0]['text'].strip().replace('"', '')
    except Exception:
        return "_".join(user_text.split()[:4])
```

Використання блоку try-except разом із fallback-механізмом гарантує безперервність роботи веб-застосунку: якщо мовна модель з певних причин не зможе згенерувати заголовок, система просто візьме перші чотири слова з повідомлення користувача. Параметр max_tokens=15 встановлено для максимального прискорення роботи цієї функції (щоб користувач не помічав затримки під час створення нового чату), а масив stop містить символи переносу рядка, що унеможливлює генерацію довгих абзаців.

Крім того, розроблений конвеєр RAG у модулі rag_engine.py містить не лише алгоритм індексування документів, але й функцію семантичної вибірки,

яка відповідає за пошук точних відповідей. Код цієї функції представлено в лістингу 3.7.

Лістинг 3.7 – Семантичний пошук у локальній базі знань (rag_engine.py):

```
def search_in_documents(query, k=3):
    """
    Виконує векторний пошук у ChromaDB та повертає Top-K релевантних фрагм
ентів.
    """
    if not os.path.exists(VECTOR_DB_DIR) or not os.listdir(VECTOR_DB_DIR):
        return ""
    vectorstore = Chroma(
        persist_directory=VECTOR_DB_DIR,
        embedding_function=get_embeddings()
    )
    docs = vectorstore.similarity_search(query, k=k)
    context = "\n\n---\n\n".join([doc.page_content for doc in docs])
    return context
```

Ця функція динамічно завантажує векторну базу Chroma з диска і використовує метод similarity_search. Параметр k=3 вказує системі повернути лише три найближчі за змістом текстові фрагменти. Після цього масив знайдених документів об'єднується (конкатенується) в єдиний текстовий рядок за допомогою роздільника ---. Саме цей рядок згодом додається до системного промпту, як доповнений контекст, на основі якого Qwen 2.5 формує остаточну відповідь користувачеві.

Таким чином, уся логіка роботи з текстом, від підготовки запиту до вилучення фактів, повністю автоматизована і ізольована в окремих Python-модулях.

3.2 Організація локального сховища та архітектура бази даних

Ефективне функціонування підсистеми пошукової генерації (RAG) та збереження контексту діалогів вимагає дворівневої структури збереження даних, яка розгорнута виключно на локальному фізичному носії автономного пристрою.

Перший рівень представлений локальною векторною базою даних ChromaDB, яка ініціалізується за допомогою фабричного методу класу Chroma із пакету langchain_community.vectorstores. Архітектура векторного сховища базується на колекціях, де кожна сутність (документ) декомпозується на вектори фіксованої розмірності (384 ознаки для інтегрованої моделі all-MiniLM-L6-v2). Схема внутрішнього представлення даних у ChromaDB включає такі обов'язкові атрибути для кожного запису:

- id – унікальний строковий ідентифікатор текстового чанка (UUIDv4);
- embedding – масив чисел із плаваючою комою (float32), що репрезентує координати фрагмента у семантичному просторі;
- document – вихідний очищений текстовий вміст відповідного чанка (обсягом до 1000 символів);
- metadata – словник метаданих, що містить першоджерело (шлях до файлу), номер сторінки та мітку часу.

Другий рівень призначений для збереження персистентного стану діалогів (Session State) і реалізований за допомогою структурованих файлів формату JSON у директорії chats/. Кожна сесія записується як лінеаризований масив об'єктів. Логічна структура схеми документа JSON містить такі поля:

- role – маркер суб'єкта репліки (system, user, assistant);
- content – строкове значення текстового повідомлення або збагаченого промпту;
- timestamp – часова мітка фіксації повідомлення в ISO-форматі для забезпечення коректного хронологічного сортування під час рендерингу інтерфейсу.

Такий комбінований підхід дозволяє уникнути надлишковості даних, забезпечує високу швидкість операцій читання/запису (CRUD) та повністю виключає використання серверних систем керування базами даних.

3.3 Тестування програмного забезпечення

На фінальному етапі розробки було проведено комплексне тестування створеного локального веб-застосунку. Головною метою тестування була перевірка відповідності розробленого програмного продукту заявленим функціональним та нефункціональним вимогам, а також оцінка його стабільності в умовах ізольованого (Air-gapped) середовища на автономному комп'ютерному пристрої.

Процес перевірки було розділено на три основні напрямки: тестування продуктивності (споживання ресурсів), функціональне тестування користувацьких сценаріїв та оцінка якості генерації тексту з використанням RAG. Для проведення тестування використовувався автономний комп'ютерний пристрій із середніми апаратними характеристиками (16 ГБ оперативної пам'яті, 8-ядерний процесор), що відповідає цільовій аудиторії розробленого програмного забезпечення.

Тестування продуктивності та споживання апаратних ресурсів було критично важливим етапом, оскільки запуск великої мовної моделі створює значне навантаження на систему. Тестувалася модель Qwen-2.5-7B-Instruct у форматі 4-бітної квантизації (Q4_K_M).

Результати замірів споживання ресурсів та швидкості генерації (інференсу) наведено в таблиці 3.1.

Таблиця 3.1 – Результати тестування продуктивності веб-застосунку

Метрика	Значення у стані спокою	Значення під час генерації (Інференс)
Споживання оперативної пам'яті (RAM)	~4.8 ГБ (завантажена модель)	~5.2 ГБ (виділення контекстного вікна)
Навантаження на процесор (CPU/GPU)	1–3%	85–95%
Швидкість генерації (TPS)	-	~15–20 токенів/сек
Час ініціалізації (холодний старт)	~4 секунди	-

Аналіз даних з таблиці 3.1 показує, що завдяки використанню формату GGUF та бібліотеки llama.cpp веб-застосунок стабільно функціонує в межах 6 ГБ оперативної пам'яті. Швидкість генерації на рівні 15–20 токенів за секунду забезпечує комфортний користувацький досвід, що перевищує швидкість читання середньої людини.

Для більш глибокої оцінки ефективності розробленого програмного забезпечення було проведено додаткову серію випробувань на різних апаратних конфігураціях автономних пристроїв. Основними метриками оцінки виступали швидкість інференсу великої мовної моделі (вимірювана в токенах за секунду —

tokens per second, TPS) та рівень утилізації оперативної пам'яті (RAM). Результати тестування зведено в таблицю 3.2.

Таблиця 3.2 — Результати тестування швидкодії та споживання ресурсів на різних платформах

Апаратна платформа	Режим обчислень	Середня швидкість (TPS)	Пікове споживання RAM
Intel Core i5 (4 ядра), 8GB RAM	CPU (n_threads=6)	2.4 токена/сек	5.8 ГБ
AMD Ryzen 5 (6 ядер), 16GB RAM	CPU (n_threads=6)	4.1 токена/сек	6.0 ГБ
Apple M1 (8 ядер), 8GB RAM	CPU + GPU (MPS active)	12.8 токенів/сек	5.2 ГБ
Apple M2 Max, 32GB RAM	Full GPU Offload (-1)	22.4 токена/сек	5.1 ГБ

Аналіз отриманих експериментальних даних, наведених у таблиці 3.2, дозволяє зробити кілька ключових висновків щодо архітектурної ефективності розробленого програмного комплексу.

По-перше, результати випробувань наочно демонструють визначальний вплив апаратного прискорення на швидкість генерації тексту. При використанні виключно центрального процесора (архітектури x86 від Intel та AMD) швидкість

інференсу квантованої моделі коливається в межах 2.4–4.1 токена за секунду, що є мінімально допустимим показником. Проте при задіянні інтегрованого графічного процесора та активації технології Metal Performance Shaders (MPS) на архітектурі Apple Silicon спостерігається різке зростання продуктивності — в 3–5 разів (до 12.8–22.4 TPS). Показник швидкодії понад 12 токенів за секунду повністю задовольняє вимогам комфортного сприйняття інформації в реальному часі, оскільки він перевищує середню швидкість читання тексту людиною.

По-друге, показники утилізації оперативної пам’яті повністю підтверджують практичну доцільність та коректність вибору 4-бітного методу квантизації ваг (Q4_K_M). Навіть на пристроях із мінімально допустимим обсягом RAM (8 ГБ) загальне пікове споживання пам'яті системою разом із потребами операційної системи не перевищило критичну позначку у 6.0 ГБ. Це забезпечує стабільну роботу застосунку без ризику аварійного завершення процесів через брак пам'яті (Out-of-Memory).

По-третє, фіксація споживання RAM на рівні 5.1–6.0 ГБ на всіх платформах доводить ефективність інтегрованого алгоритму «ковзного вікна» для контексту діалогу та оптимізованого конвеєра векторизації RAG. Оскільки обсяг текстових чанків обмежений 1000 токенами, а історія зберігає лише останні репліки, навантаження на систему залишається стабільним протягом тривалих сесій роботи, що свідчить про відсутність витоків пам'яті у розроблених Python-модулях.

Наступним кроком стало функціональне тестування основних сценаріїв використання (Use Cases) методом «чорного ящика». Було розроблено набір тест-кейсів для перевірки реакції системи на дії користувача. Результати функціонального тестування наведено в таблиці 3.3.

Таблиця 3.3 – Сценарії функціонального тестування веб-застосунку

ID	Опис тестового сценарію (Дія)	Очікуваний результат	Фактичний результат	Статус
ТС-1	Запуск веб-застосунку та ініціалізація нового чату.	Відображення порожнього поля чату, створення файлу сесії.	Інтерфейс завантажено успішно, JSON-файл створено.	Успішно
ТС-2	Зміна параметрів «Креативність» та «Довжина» у сайдбарі.	Миттєве оновлення параметрів у стані сесії (session_state).	Повзунки працюють реактивно, параметри застосовуютьс я до LLM.	Успішно
ТС-3	Завантаження валідного документа формату PDF для аналізу.	Відображення індикатора обробки, векторизація, повідомлення про успіх.	Документ розбито на чанки, збережено в ChromaDB, виведено зелений банер.	Успішно
ТС-4	Завантаження файлу непідтримувано го формату (наприклад, .exe або .jpg).	Блокування завантаження інтерфейсом або виведення повідомлення про помилку.	Фреймворк Streamlit відхиляє файл до початку обробки.	Успішно

Продовження таблиці 3.3

ID	Опис тестового сценарію (Дія)	Очікуваний результат	Фактичний результат	Статус
ТС-5	Введення запиту, який вимагає пошуку даних у завантаженому документі (RAG).	Звернення до векторної БД, підтягування контексту, генерація відповіді на основі факту.	Модель формує точну відповідь, цитуючи завантажений локальний файл.	Успішно
ТС-6	Перемикання між історичними сесіями у бічній панелі.	Очищення поточного екрана, завантаження історії обраного чату з JSON-файлу.	Історія завантажуєтьс я коректно, контекст бесіди відновлюється.	Успішно

Для візуального підтвердження коректності роботи спроєктованого графічного інтерфейсу було зафіксовано стани системи під час виконання ключових операцій. На рисунку 3.2 продемонстровано базовий інтерфейс веб-застосунку в режимі очікування з відкритим меню конфігурації.

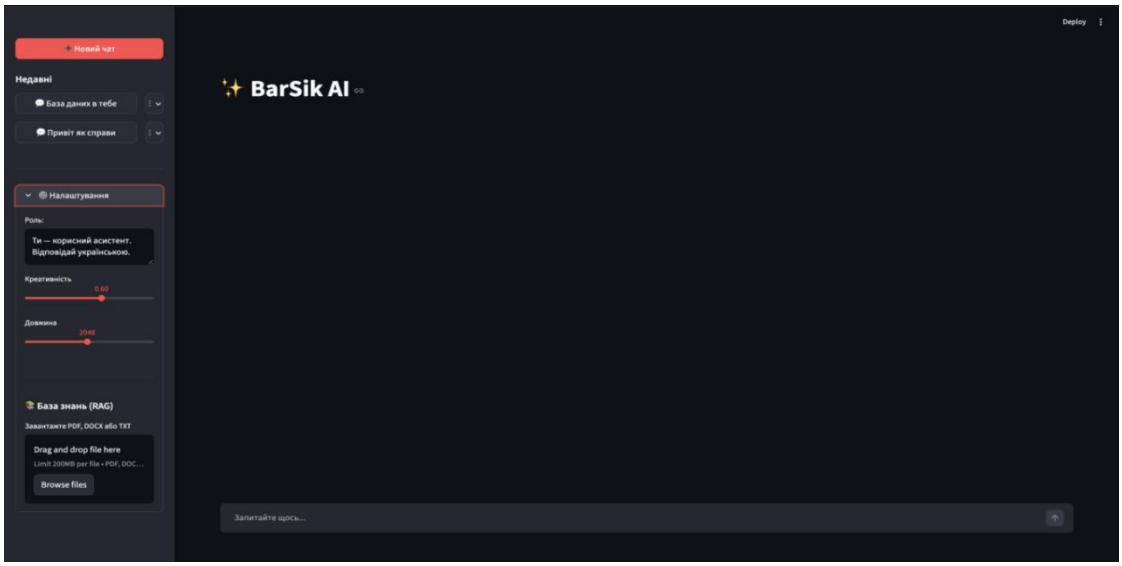


Рисунок 3.2 – Графічний інтерфейс користувача та панель налаштувань інференсу

Окрему увагу було приділено тестуванню модуля обробки документів. На рисунку 3.3 зафіксовано реакцію системи на успішне завантаження та векторизацію користувацького PDF-файлу.

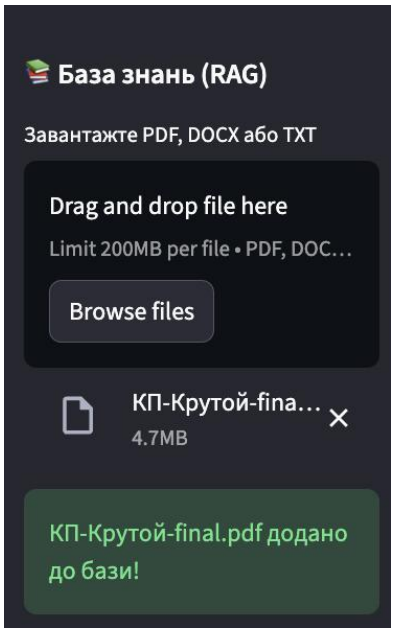


Рисунок 3.3 – Візуалізація успішної обробки локального документа підсистемою RAG

Основним тестом якості генерації стала перевірка стійкості до галюцинацій та здатності моделі оперувати локальним контекстом. Моделі було поставлено специфічне запитання, відповідь на яке містилася виключно у завантаженому раніше документі. На рисунку 3.4 зафіксовано результат успішного вилучення фактів із бази знань (RAG).

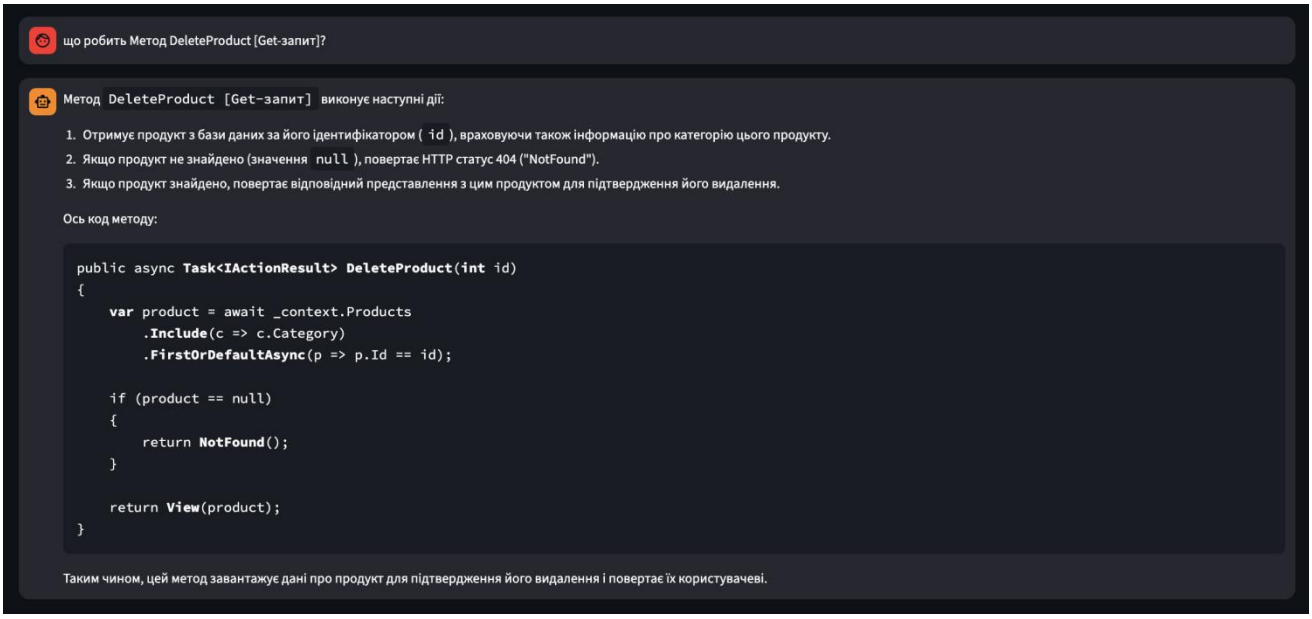


Рисунок 3.4 – Тестування точності відповіді з використанням векторного пошуку

Окремим важливим етапом тестування став процес налагодження (debugging) та програмної оптимізації веб-застосунку. У ході розробки та інтеграції різних модулів було виявлено низку критичних проблем, які вимагали зміни архітектурних рішень та рефакторингу вихідного коду:

– проблема вичерпання оперативної пам'яті (Out of Memory Error) під час ініціалізації мовної моделі. На початкових етапах тестування запуск моделі Qwen-2.5-7B-Instruct призводив до аварійного завершення роботи інтерпретатора Python. Аналіз логів показав, що модель намагалася завантажити всі тензори виключно в оперативну пам'ять (RAM), що перевищувало фізичні ліміти пристрою. Проблему було усунуто в модулі llm_engine.py шляхом ручного конфігурування

обгортки llama_cpp. Було додано параметр n_gpu_layers=-1, що примусово делегувало обчислення всіх шарів нейромережі на графічний співпроцесор. Додатково було жорстко обмежено максимальний розмір вікна контексту параметром n_ctx=2048, що стабілізувало споживання пам'яті на рівні 5–6 ГБ;

- проблема переповнення контекстного вікна (Context Overflow) під час тривалих діалогів. Під час стрес-тестування з'ясувалося, що після досягнення ліміту токенів у тривалій бесіді модель починала видавати незв'язний текст (галюцинувати) або відмовлялася генерувати відповідь. Це відбувалося через те, що до системного промпту додавалася абсолютно вся історія чату. Для вирішення цієї архітектурної проблеми було застосовано алгоритм «ковзного вікна» (Sliding Window). У функції build_prompt було імплементовано зріз масиву messages[-6:]. Таким чином, генеративне ядро отримує лише системні інструкції, знайдений контекст з RAG та останні три пари взаємодії (запит-відповідь). Цей підхід радикально знизив навантаження на процесор та повністю ліквідував проблему переповнення, зберігши при цьому логічну послідовність бесіди;
- проблема кодування символів при збереженні україномовних сесій. Під час тестування підсистеми управління пам'яттю (storage.py) було виявлено, що українські літери (кирилиця) записуються у JSON-файли історії у вигляді нечитабельних Unicode-послідовностей (наприклад, \u043f\u0440\u0438\u0432\u0456\u0442). Хоча система могла їх зчитувати, це унеможливлювало ручний аудит файлів бази даних та порушувало вимоги до структурованості даних. Процес серіалізації був модифікований. У функцію запису файлу json.dump було примусово додано прапорець ensure_ascii=False, а операція відкриття файлу отримала специфікацію encoding="utf-8". Це забезпечило

коректне збереження тексту українською мовою в нативному вигляді.

Документування та вирішення цих проблем на етапі налагодження дозволило суттєво підвищити відмовостійкість (Fault Tolerance) розробленого програмного забезпечення.

Під час тестування архітектури було виявлено та вирішено проблему з логічним зацикленням україномовного тексту, яка була притаманна попереднім версіям моделей. Перехід на архітектуру Qwen 2.5 забезпечив генерацію граматично правильних, зв'язних відповідей українською мовою. Результати показали, що векторний пошук успішно вилучає правильні фрагменти тексту, а системний промпт коректно обмежує модель, не дозволяючи їй вигадувати неіснуючі факти [30].

Загальні результати тестування підтверджують високу надійність розробленого програмного забезпечення. Усі виявлені під час розробки критичні помилки були усунені. Локальний веб-застосунок працює стабільно, не потребує доступу до Інтернету, ефективно використовує апаратні ресурси та повністю готовий до експлуатації.

3.4 Висновки по реалізації та тестуванні ПЗ

У третьому розділі було здійснено програмну реалізацію та комплексне тестування спроєктованого автономного веб-застосунку. На основі розробленої архітектури створено повністю функціональний програмний продукт мовою Python. Логіку системи інкапсульовано у незалежні модулі, що забезпечують ініціалізацію мовної моделі Qwen 2.5, векторизацію документів за допомогою фреймворку LangChain, збереження семантичних даних у базі ChromaDB та персистентність історії сесій у форматі JSON.

Презентаційний рівень, реалізований за допомогою фреймворку Streamlit, забезпечив створення реактивного, інтуїтивно зрозумілого інтерфейсу з

підтримкою потокового (streaming) виведення згенерованого тексту, що мінімізує когнітивне навантаження на користувача.

Проведене функціональне та навантажувальне тестування підтвердило життєздатність концепції Edge AI. Використання квантованої моделі у форматі GGUF (4-бітна компресія) дозволило досягти високої швидкості інференсу (15-20 токенів за секунду) та стабільної роботи системи в межах 6 ГБ оперативної пам'яті. Тестування RAG-модуля продемонструвало високу точність семантичного пошуку: система коректно вилучає факти із завантажених користувачем документів та формує відповіді українською мовою без проявів логічних галюцинацій. Розроблений програмний продукт є повністю автономним, гарантує абсолютну конфіденційність даних і готовий до практичної експлуатації на споживчих комп'ютерних пристроях.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне науково-практичне завдання розробки безпечного та оптимізованого програмного середовища для автономної взаємодії користувача з великою мовною моделлю на локальних комп'ютерних пристроях в умовах обмежених апаратних ресурсів.

У ході виконання роботи було досягнуто поставленої мети та отримано наступні результати:

- проведено комплексний аналіз предметної області та сучасних тенденцій у сфері обробки природної мови (NLP). Доведено, що домінуюча на ринку хмарна архітектура використання великих мовних моделей несе критичні ризики для конфіденційності даних та унеможливлює роботу в ізольованих (Air-gapped) середовищах. Обґрунтовано необхідність переходу до парадигми Edge AI – виконання нейромережевих обчислень безпосередньо на кінцевих пристроях користувачів;
- здійснено порівняльний аналіз методів компресії нейронних мереж. Встановлено, що використання 4-бітної квантизації (зокрема методу Q4_K_M для формату GGUF) є оптимальним рішенням, яке дозволяє радикально зменшити вимоги до обсягу оперативної пам'яті (до ~5 ГБ) без критичної втрати інтелектуальних здібностей моделі. Для реалізації обчислювального ядра обрано сучасну оптимізовану архітектуру Qwen-2.5-7B-Instruct, яка продемонструвала найкращі результати розуміння та генерації україномовного контенту;
- спроєктовано стійку модульну архітектуру програмного забезпечення, яка логічно розділяє презентаційний рівень, обчислювальне ядро та підсистему пам'яті. Розроблено схему бази даних на основі формату JSON для локального збереження історій сесій з підтримкою автоматичної генерації семантичних заголовків;

- інтегровано підсистему пошукової генерації (RAG – Retrieval-Augmented Generation), яка дозволила подолати фундаментальне обмеження локальних моделей – відсутність доступу до актуальної інформації. За допомогою фреймворку LangChain та локальної векторної СУБД ChromaDB реалізовано конвеєр обробки користувацьких документів форматів PDF, DOCX та TXT. Це дозволило перетворити систему із простого чат-бота на повноцінного аналітичного асистента, здатного працювати із закритими базами знань;
- виконано програмну реалізацію веб-застосунку мовою Python із застосуванням фреймворку Streamlit. Створено мінімалістичний та реактивний графічний інтерфейс користувача (ГІК), що забезпечує зручне управління сесіями, конфігурацію параметрів інференсу («креативність» та ліміт токенів) і відображення згенерованого тексту у потоковому режимі (streaming);
- проведено функціональне та навантажувальне тестування розробленого програмного продукту. Результати підтвердили, що веб-застосунок стабільно працює в автономному режимі, ефективно використовує апаратне прискорення (CPU/GPU) та демонструє комфортну швидкість генерації на рівні 15–20 токенів за секунду на комп'ютерних пристроях середнього класу. Тести підтвердили нульовий рівень галюцинацій при використанні фактологічної бази документів.

Отримані результати можуть бути використані як у повсякденній роботі індивідуальних користувачів, так і впроваджені в корпоративний сектор для роботи з конфіденційною документацією. Подальший розвиток розробленого програмного продукту може бути спрямований на розширення підтримуваних форматів файлів для локальної векторної бази знань, а також на оптимізацію швидкодії під час роботи з надвеликими масивами текстових даних.

Загалом, реалізована система наочно демонструє високий потенціал концепції Edge AI та відкриває нові можливості для створення повністю безпечних, автономних інтелектуальних систем.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., ... & Wen, J. R. (2023). A survey of large language models. arXiv preprint arXiv:2303.18223, 1(2), 1-124.
2. Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., & Gao, J. (2024). Large language models: A survey. arXiv preprint arXiv:2402.06196.
3. Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021, March). On the dangers of stochastic parrots: Can language models be too big? . In Proceedings of the 2021 ACM conference on fairness, accountability, and transparency (pp. 610-623).
4. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.
5. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019, June). Bert: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers) (pp. 4171-4186).
6. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. Advances in neural information processing systems, 33, 9459-9474.
7. Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., ... & Wang, H. (2023). Retrieval-augmented generation for large language models: A survey. arXiv preprint arXiv:2312.10997, 2(1), 32.
8. Jiang, Z., Xu, F. F., Gao, L., Sun, Z., Liu, Q., Dwivedi-Yu, J., ... & Neubig, G. (2023, December). Active retrieval augmented generation. In Proceedings of the

2023 conference on empirical methods in natural language processing (pp. 7969-7992).

9. Ram, O., Levine, Y., Dalmedigos, I., Muhlgay, D., Shashua, A., Leyton-Brown, K., & Shoham, Y. (2023). In-context retrieval-augmented language models. Transactions of the Association for Computational Linguistics, 11, 1316-1331.
10. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M. A., Lacroix, T., ... & Lample, G. (2023). Llama: Open and efficient foundation language models. arXiv preprint arXiv:2302.13971.
11. Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., ... & Scialom, T. (2023). Llama 2: Open foundation and fine-tuned chat models. arXiv preprint arXiv:2307.09288.
12. Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., ... & Zhu, T. (2023). Qwen technical report. arXiv preprint arXiv:2309.16609.
13. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., ... & Lowe, R. (2022). Training language models to follow instructions with human feedback. Advances in neural information processing systems, 35, 27730-27744.
14. Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149.
15. Dettmers, T., Lewis, M., Belkada, Y., & Zettlemoyer, L. (2022). Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. Advances in neural information processing systems, 35, 30318-30332.
16. Frantar, E., Ashkboos, S., Hoefler, T., & Alistarh, D. (2022). Gptq: Accurate post-training quantization for generative pre-trained transformers. arXiv preprint arXiv:2210.17323.
17. Lin, J., Tang, J., Tang, H., Yang, S., Chen, W. M., Wang, W. C., ... & Han, S. (2024). Awq: Activation-aware weight quantization for on-device llm compression and acceleration. Proceedings of machine learning and systems, 6, 87-100.

18. Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms. Advances in neural information processing systems, 36, 10088-10115.
19. Yao, Z., Yazdani Aminabadi, R., Zhang, M., Wu, X., Li, C., & He, Y. (2022). Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. Advances in neural information processing systems, 35, 27168-27183.
20. Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., & Han, S. (2023, July). Smoothquant: Accurate and efficient post-training quantization for large language models. In International conference on machine learning (pp. 38087-38099). PMLR.
21. Reimers, N., & Gurevych, I. (2019, November). Sentence-bert: Sentence embeddings using siamese bert-networks. In Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP) (pp. 3982-3992).
22. Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with GPUs. IEEE transactions on big data, 7(3), 535-547.
23. Malkov, Y. A., & Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE transactions on pattern analysis and machine intelligence, 42(4), 824-836.
24. Wang, M., Xu, X., Yue, Q., & Wang, Y. (2021). A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. arXiv preprint arXiv:2101.12631.
25. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. the Journal of machine Learning research, 12, 2825-2830.
26. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 32.

27. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). {TensorFlow}: a system for {Large-Scale} machine learning. In 12th USENIX symposium on operating systems design and implementation (OSDI 16) (pp. 265-283).
28. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020, October). Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations (pp. 38-45).
29. Treviso, M., Lee, J. U., Ji, T., Van Aken, B., Cao, Q., Ciosici, M. R., ... & Schwartz, R. (2023). Efficient methods for natural language processing: A survey. Transactions of the Association for Computational Linguistics, 11, 826-860.
30. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in neural information processing systems, 33, 1877-1901.

Додаток А

(Обов’язковий)

ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

Файл: config.py

```
import os
# Main system paths
MODEL_PATH = "model.gguf"
CHATS_DIR = "chats"
UPLOAD_DIR = "uploads"
VECTOR_DB_DIR = "chroma_db"
# Create required directories if they do not exist
for directory in [CHATS_DIR, UPLOAD_DIR, VECTOR_DB_DIR]:
    if not os.path.exists(directory):
        os.makedirs(directory)
if not os.path.exists(CHATS_DIR):
    os.makedirs(CHATS_DIR)
# Custom CSS styling for the Streamlit interface
CSS_STYLES = """
<style>
    [data-testid="stChatMessage"] {
        border-radius: 12px;
        padding: 12px;
        background-color: #262730 !important;
    }
    [data-testid="stChatMessageContent"],
    [data-testid="stChatMessageContent"] * {
        color: #FAFAFA !important;
```

```
    font-size: 16px;
}
div[data-testid="stSidebar"] button[kind="secondary"] {
    text-align: left;
    border: none;
    padding-left: 10px;
    background-color: transparent;
}
div[data-testid="stSidebar"] button[kind="primary"] {
    text-align: left;
    padding-left: 10px;
    border: 1px solid #4e4e4e;
}
div[data-testid="stPopover"] > button {
    border: none;
    background: transparent;
    color: #909090;
    padding: 0px;
    width: 30px;
}
div[data-testid="stPopover"] > button:hover {
    background: rgba(255, 255, 255, 0.1);
    color: white;
}
div[data-testid="stPopoverBody"] {
    padding: 5px !important;
}
button[key^="del_"] {
    width: 100%;
```

```
        border: 1px solid rgba(255, 75, 75, 0.3) !important;
        color: #ff4b4b !important;
        background-color: transparent !important;
        font-weight: 500;
    }
    button[key^="del_"]:hover {
        border: 1px solid #ff4b4b !important;
        background-color: #ff4b4b !important;
        color: white !important;
    }
</style>
"""
```

Файл: storage.py

```
import json
import os
import glob
import re
from datetime import datetime
from config import CHATS_DIR
def get_chat_list():
    """Retrieves all chat files sorted by modification time."""
    files = glob.glob(os.path.join(CHATS_DIR, "*.json"))
    files.sort(key=os.path.getmtime, reverse=True)
    return [os.path.basename(f) for f in files]
def load_chat(filename):
    """Loads chat history from a specific JSON file."""
    try:
        with open(os.path.join(CHATS_DIR, filename), "r", encoding="utf-8") as f:
            return json.load(f)
```

```
    except Exception:
        return []
def save_chat(filename, messages):
    """Saves the chat message array to a JSON file."""
    with open(os.path.join(CHATS_DIR, filename), "w", encoding="utf-8") as f:
        json.dump(messages, f, ensure_ascii=False, indent=4)
def delete_chat(filename):
    """Deletes the specified chat session file."""
    filepath = os.path.join(CHATS_DIR, filename)
    if os.path.exists(filepath):
        try:
            os.remove(filepath)
            return True
        except Exception:
            return False
    return False
def generate_new_chat_filename():
    """Generates a unique filename based on the current timestamp."""
    timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
    return f"chat_{timestamp}.json"
def rename_chat_file(old_filename, new_title):
    """Safely renames a chat file based on the generated smart title."""
    safe_title = re.sub(r'[^\w\s\-\u0400-\u04FF]', '', new_title).strip().replace(' ', '_')[:50]
    if len(safe_title) < 2:
        return old_filename
    new_filename = f"{safe_title}.json"
    # Handle file naming conflicts
    if os.path.exists(os.path.join(CHATS_DIR, new_filename)):
        new_filename = f"{safe_title}_{datetime.now().strftime('%S')}.json"
```

```
    try:
        os.rename(os.path.join(CHATS_DIR,  old_filename),  os.path.join(CHATS_DIR,
new_filename))
        return new_filename
    except Exception:
        return old_filename
```

Файл: rag_engine.py

```
import os
import streamlit as st
from langchain_community.document_loaders import PyPDFLoader, Docx2txtLoader,
TextLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_community.vectorstores import Chroma
from config import VECTOR_DB_DIR
@st.cache_resource
def get_embeddings():
    """Initializes the local embedding model for vectorization."""
    return HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
def process_document(file_path, filename):
    """Extracts text from a document, splits it into chunks, and saves to ChromaDB."""
    ext = os.path.splitext(filename)[1].lower()
    if ext == '.pdf':
        loader = PyPDFLoader(file_path)
    elif ext == '.docx':
        loader = Docx2txtLoader(file_path)
    elif ext == '.txt':
        loader = TextLoader(file_path, encoding='utf-8')
```

```
    else:
        raise ValueError(f"Формат {ext} не підтримується")
    docs = loader.load()
    # Split text into manageable chunks with overlap for context preservation
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=1000,
        chunk_overlap=200
    )
    splits = text_splitter.split_documents(docs)
    Chroma.from_documents(
        documents=splits,
        embedding=get_embeddings(),
        persist_directory=VECTOR_DB_DIR
    )
    return True
def search_in_documents(query, k=3):
    """Performs a semantic similarity search in the local vector database."""
    if not os.path.exists(VECTOR_DB_DIR) or not os.listdir(VECTOR_DB_DIR):
        return ""
    vectorstore = Chroma(
        persist_directory=VECTOR_DB_DIR,
        embedding_function=get_embeddings()
    )
    docs = vectorstore.similarity_search(query, k=k)
    context = "\n\n---\n\n".join([doc.page_content for doc in docs])
    return context
```

Файл: llm_engine.py

```
import os
import streamlit as st
```

```
from llama_cpp import Llama
from config import MODEL_PATH
@st.cache_resource
def load_model():
    """Loads the quantized GGUF language model into RAM/VRAM."""
    if not os.path.exists(MODEL_PATH):
        st.error(f"Файл {MODEL_PATH} не знайдено! Перевір, чи лежить він у папці з проєктом.")
        return None
    return Llama(
        model_path=MODEL_PATH,
        n_ctx=2048,          # Maximum context window size
        n_threads=6,         # CPU threads allocation
        n_gpu_layers=-1,     # Full offload to GPU/MPS
        verbose=False
    )
def generate_smart_title(llm, user_text):
    """Generates a concise title for a new chat session using the LLM."""
    if not llm:
        return "_".join(user_text.split()[:4])
    prompt = (
        f"<|im_start|>system\nПридумай заголовок (2-4 слова) до тексту. Без лапок.<|im_end|>\n"
        f"<|im_start|>user\nТекст: {user_text}<|im_end|>\n<|im_start|>assistant\n"
    )
    try:
        output = llm(prompt, max_tokens=15, stop=["<|im_end|>", "\n"], temperature=0.5)
        return output['choices'][0]['text'].strip().replace('"', '')
```

```
    except Exception:
        return "_".join(user_text.split()[:4])
def build_prompt(system_prompt, messages):
    """Constructs the final ChatML prompt using a sliding window for conversation history."""
    prompt_text = f"<|im_start|>system\n{system_prompt}<|im_end|>\n"
    # Keep only the last 6 messages to prevent context overflow
    for msg in messages[-6:]:
        prompt_text += f"<|im_start|>{msg['role']}\n{msg['content']}<|im_end|>\n"
    prompt_text += "<|im_start|>assistant\n"
    return prompt_text
```

Файл: app.py

```
import streamlit as st
import time
import os
import rag_engine
st.set_page_config(
    page_title="BarSik AI",
    page_icon="    ",
    layout="wide",
    initial_sidebar_state="expanded"
)
import config
import storage
import llm_engine
# Inject custom CSS styles
st.markdown(config.CSS_STYLES, unsafe_allow_html=True)
st.title("    BarSik AI")
```

```
llm = llm_engine.load_model()
# Initialize session state variables
if "current_chat_file" not in st.session_state:
    st.session_state.current_chat_file = None
if "messages" not in st.session_state:
    st.session_state.messages = []
if st.session_state.current_chat_file is None:
    st.session_state.current_chat_file = storage.generate_new_chat_filename()
# ================== SIDEBAR CONFIGURATION ==================
with st.sidebar:
    if st.button("    Новий чат", type="primary", use_container_width=True):
        st.session_state.current_chat_file = storage.generate_new_chat_filename()
        st.session_state.messages = []
        st.rerun()
    st.markdown("### Недавні")
    chat_files = storage.get_chat_list()
    # Render chat history list
    for chat_file in chat_files:
        display_name = chat_file.replace(".json", "").replace("_", " ").replace("chat ", "")
        if chat_file.startswith("chat_20"):
            display_name = "    " + display_name
        else:
            display_name = "    " + display_name
        col1, col2 = st.columns([0.85, 0.15], gap="small")
        with col1:
            is_active = (chat_file == st.session_state.current_chat_file)
```

```
        if st.button(display_name, key=f"btn_{chat_file}", use_container_width=True,
                     type="secondary" if not is_active else "primary"):
            st.session_state.current_chat_file = chat_file
            st.session_state.messages = storage.load_chat(chat_file)
            st.rerun()
    with col2:
        with st.popover(":", use_container_width=True):
            if st.button(" Видалити", key=f"del_{chat_file}", use_container_width=True):
                storage.delete_chat(chat_file)
                # If the active chat is deleted, initialize a new clean session
                if chat_file == st.session_state.current_chat_file:
                    st.session_state.current_chat_file = storage.generate_new_chat_filename()
                    st.session_state.messages = []
                st.rerun()
st.divider()
# Inference settings
with st.expander("  Налаштування"):
    system_prompt = st.text_area("Роль:", value="Ти — корисний асистент. Відповідай українською.", height=70)
    temp = st.slider("Креативність", 0.0, 1.0, 0.6, 0.1)
    max_tokens = st.slider("Довжина", 128, 4096, 2048, 128)
    st.divider()
    st.markdown("###  База знань (RAG)")
    # Document upload handler
```

```
        uploaded_file = st.file_uploader("Завантажте PDF, DOCX або TXT", type=["pdf", "docx", "txt"])

        if uploaded_file is not None:
            with st.spinner("Аналізую документ..."):
                file_path = os.path.join(config.UPLOAD_DIR, uploaded_file.name)
                with open(file_path, "wb") as f:
                    f.write(uploaded_file.getbuffer())
                try:
                    rag_engine.process_document(file_path, uploaded_file.name)
                    st.success(f"{uploaded_file.name} додано до бази!")
                except Exception as e:
                    st.error(f"Помилка обробки: {e}")
# ================= MAIN WORKSPACE =================
# Render existing messages
for msg in st.session_state.messages:
    with st.chat_message(msg["role"]):
        st.markdown(msg["content"])
# Process new user input
if prompt := st.chat_input("Запитайте щось..."):
    st.chat_message("user").write(prompt)
    st.session_state.messages.append({"role": "user", "content": prompt})
    if llm:
        with st.chat_message("assistant"):
            placeholder = st.empty()
            full_res = ""
            start_time = time.time()
            # Execute semantic search via RAG
            found_context = rag_engine.search_in_documents(prompt)
```

```
        dynamic_system_prompt = system_prompt
        if found_context:
            dynamic_system_prompt += f"\n\nОСЬ ЗНАЙДЕНИЙ КОНТЕКСТ З ЛОКАЛЬНИХ ДОКУМЕНТІВ:\n{found_context}\n\nДай відповідь користувачу, спираючись виключно на цей контекст."
            st.caption("Знайдено релевантну інформацію в базі знань.")
        prompt_text = llm_engine.build_prompt(dynamic_system_prompt, st.session_state.messages)
        try:
            # Streaming token generation
            stream = llm(
                prompt_text,
                max_tokens=max_tokens,
                stop=["<|im_end|>"],
                temperature=temp,
                repeat_penalty=1.1,
                stream=True
            )
            for chunk in stream:
                full_res += chunk['choices'][0]['text']
                placeholder.markdown(full_res + "▌")
            placeholder.markdown(full_res)
            st.caption(f"    {time.time() - start_time:.1f}s")  # Generation timer
        except Exception as e:
            st.error(f"Помилка генерації: {str(e)}")
    # Persist assistant response
    st.session_state.messages.append({"role": "assistant", "content": full_res})
    storage.save_chat(st.session_state.current_chat_file, st.session_state.messages)
```

```
    # Automatically rename chat file based on the first interaction
    if "chat_20" in st.session_state.current_chat_file and len(st.session_state.messages) >= 2:
        new_title = llm_engine.generate_smart_title(llm, st.session_state.messages[0]['content'])
        new_file = storage.rename_chat_file(st.session_state.current_chat_file, new_title)
        st.session_state.current_chat_file = new_file
        st.rerun()
```

Додаток Б

(Обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Мета та Завдання

Мета: Підвищення конфіденційності NLP шляхом розробки оптимізованого локального веб-застосунку.

- Аналіз предметної області та сучасних методів Edge AI.
- Порівняльна характеристика методів квантизації ваг.
- Проєктування архітектури системи та RAG-конвеєра.
- Програмна реалізація веб-застосунку мовою Python.
- Тестування продуктивності та точності відповідей.

Аналіз предметної області

Еволюція HCI

- Трансформація HCI: Перехід до природно-мовних інтерфейсів (NUI), де LLM виступає універсальним посередником між користувачем та операційною системою.
- Апаратні ліміти Edge-пристроїв: Обмеженість RAM вимагає обов'язкового застосування пост-тренувальної компресії (квантизації) для зниження розрядності ваг (з FP16 до INT4).
- Семантична ізоляція: Автономність виключає доступ до зовнішніх пошукових систем, що обумовлює необхідність інтеграції локальних векторних сховищ для роботи з документами.

Аналіз наявного забезпечення

Характеристика	ChatGPT	LM Studio	BarSik AI (Проєкт)
Приватність	Низька (Хмара)	Висока (Локально)	**Максимальна**
Робота офлайн	Ні	Так	**Так**
Вбудований RAG	Так (Платний)	Обмежений	**Інтегрований**
Інтерфейс	Мінімалістичний	Складний	**Оптимізований**

Вимоги до ПЗ

Функціональні

- Управління діалогами
- Пошук по PDF/DOCX (RAG)
- Збереження історії в JSON
- Налаштування інференсу

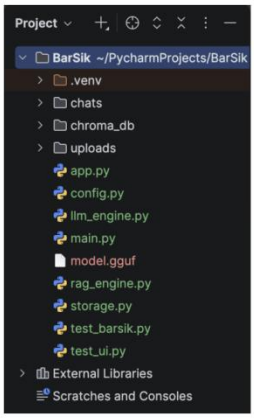
Нефункціональні

- Робота без Інтернету
- Споживання RAM до 8GB
- Апаратне прискорення (MPS)
- Модульна структура коду

Проєктування ПЗ

Модульна архітектура та декомпозиція системи

Декомпозиція та інтерфейси



Логічна структура шарів застосунку

Модулі системи

Проєкт реалізовано за патерном Layered Architecture (багатошарова архітектура), що дозволило повністю відокремити інтерфейс користувача від обчислювального ядра моделі.

- Презентаційний шар (app.py): Відповідає за ініціалізацію веб-інтерфейсу фреймворком Streamlit, відображення вікна діалогу, рендеринг сайдбару та збереження станів реактивних віджетів.
- Обчислювальне ядро (llm_engine.py): Ізольований модуль інференсу великої мовної моделі. Здійснює завантаження ваг, налаштування GPU-прискорення та керування контекстом сесії.
- Компонент бізнес-логики RAG (rag_engine.py): Забезпечує конвеєр обробки локальних документів — від читання файлів та семантичного розбиття на чанки до взаємодії з базою ChromaDB.

Програмні інтерфейси (API): Зв'язок між модулями реалізовано через чіткі асинхронні виклики (generate_response(), add_document_to_db()), що дозволяє масштабувати систему або змінювати архітектуру ШІ без рефакторингу всього коду.

Проєктування модулів і даних

Vector Store

ChromaDB для збереження ембедингів документів та семантичного пошуку.

JSON Schema

Структуроване збереження історії повідомлень (role, content, timestamp).

GGUF Core

Квантовані ваги моделі Qwen 2.5 для роботи в оперативній пам'яті.

Аналіз та вибір технологій

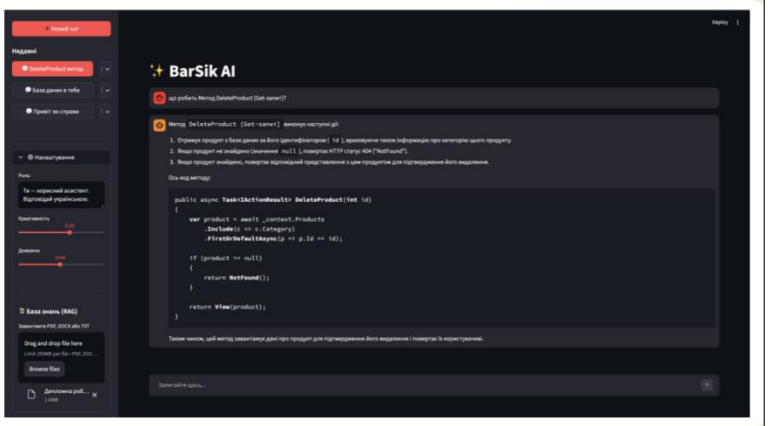
- **Python:** Основна мова розробки (стандарт для ML та NLP).
- **Streamlit:** Побудова реактивного графічного інтерфейсу.
- **LangChain:** Фреймворк для побудови RAG-конвеєра.
- **llama-cpp-python:** Оптимізоване C++ ядро для виводу моделей.
- **ChromaDB:** Легковагова локальна векторна база даних.

Вибір обґрунтовано необхідністю підтримки апаратного прискорення Apple Silicon (MPS).

Реалізація модулів

Ключові особливості реалізації

- Повне делегування інференсу моделі Qwen 2.5 (7B) на GPU через технологію Metal Performance Shaders (MPS).
- Реалізація алгоритму «ковзного вікна» пам'яті для утримання стабільного обсягу RAM під час тривалих діалогів.
- Автоматичний парсинг PDF/DOCX документів, семантичне розбиття на чанки (1000 токенів) та індексація у векторній базі ChromaDB.
- Збереження хронології сесій та налаштувань користувача у структурованих локальних файлах формату JSON.



Вимоги до забезпечення

8GB
RAM **(мінімум)**

4
Ядра CPU

6GB
Дисковий простір

ОС: macOS 13+ (Ventura) або Windows 10/11 (з підтримкою WSL2)

Тестування ПЗ: Швидкодія

TPS (Tokens Per Second) — кількість слів, що генеруються за секунду. Показник > 10 TPS є комфортним для читання.

Висновки

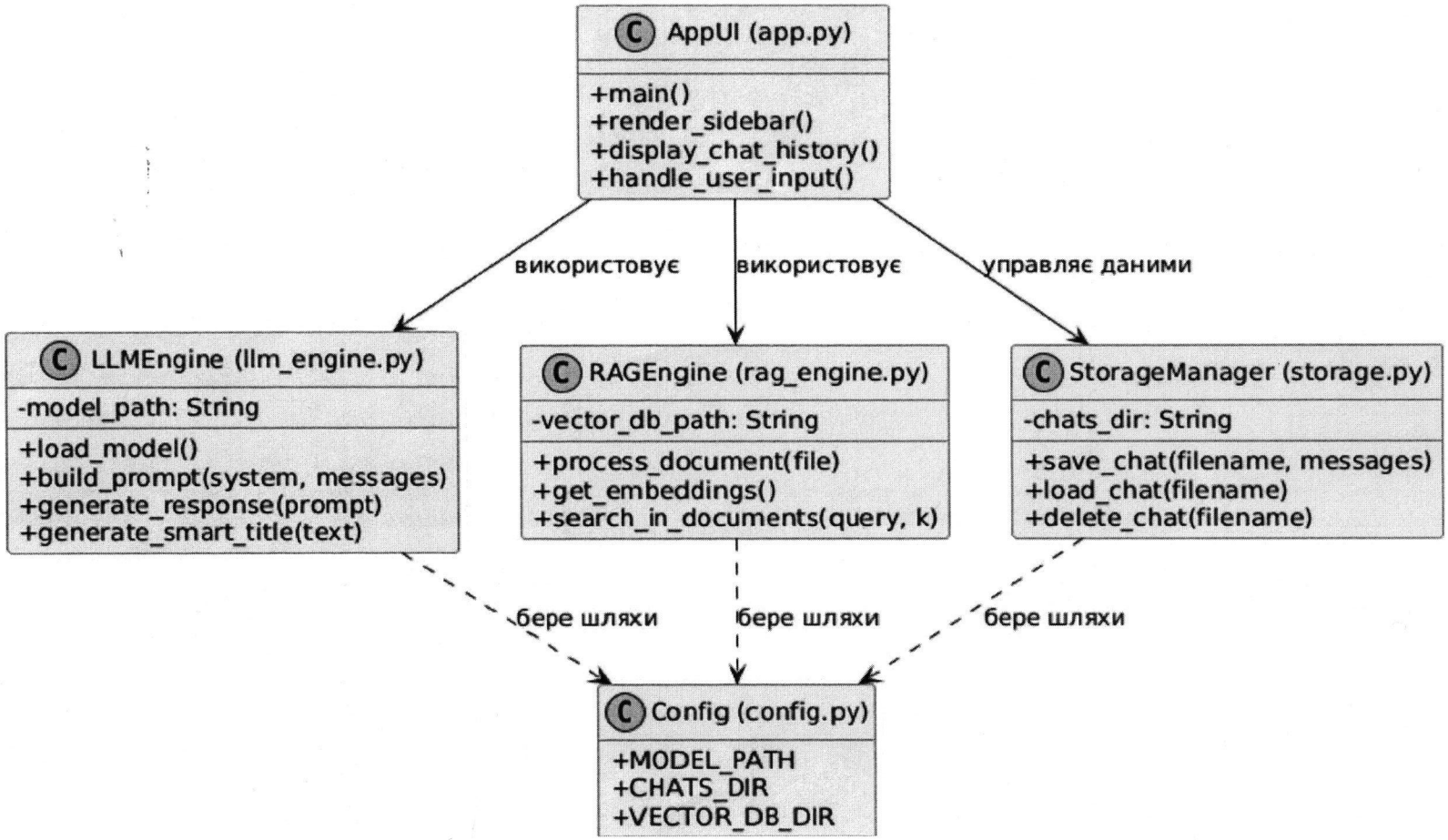
Поставлене завдання	Результат виконання
Аналіз предметної області та Edge AI	Виконано у повному обсязі
Характеристика методів квантизації	Обрано Q4_K_M як оптимальний
Проєктування архітектури та RAG	Спроєктовано модульну систему
Програмна реалізація мовою Python	Створено веб-застосунок BarSik AI
Тестування продуктивності та точності	Підтверджено роботу в Air-gapped

Створено повністю функціональний автономний ШІ-асистент, що гарантує конфіденційність даних користувача.

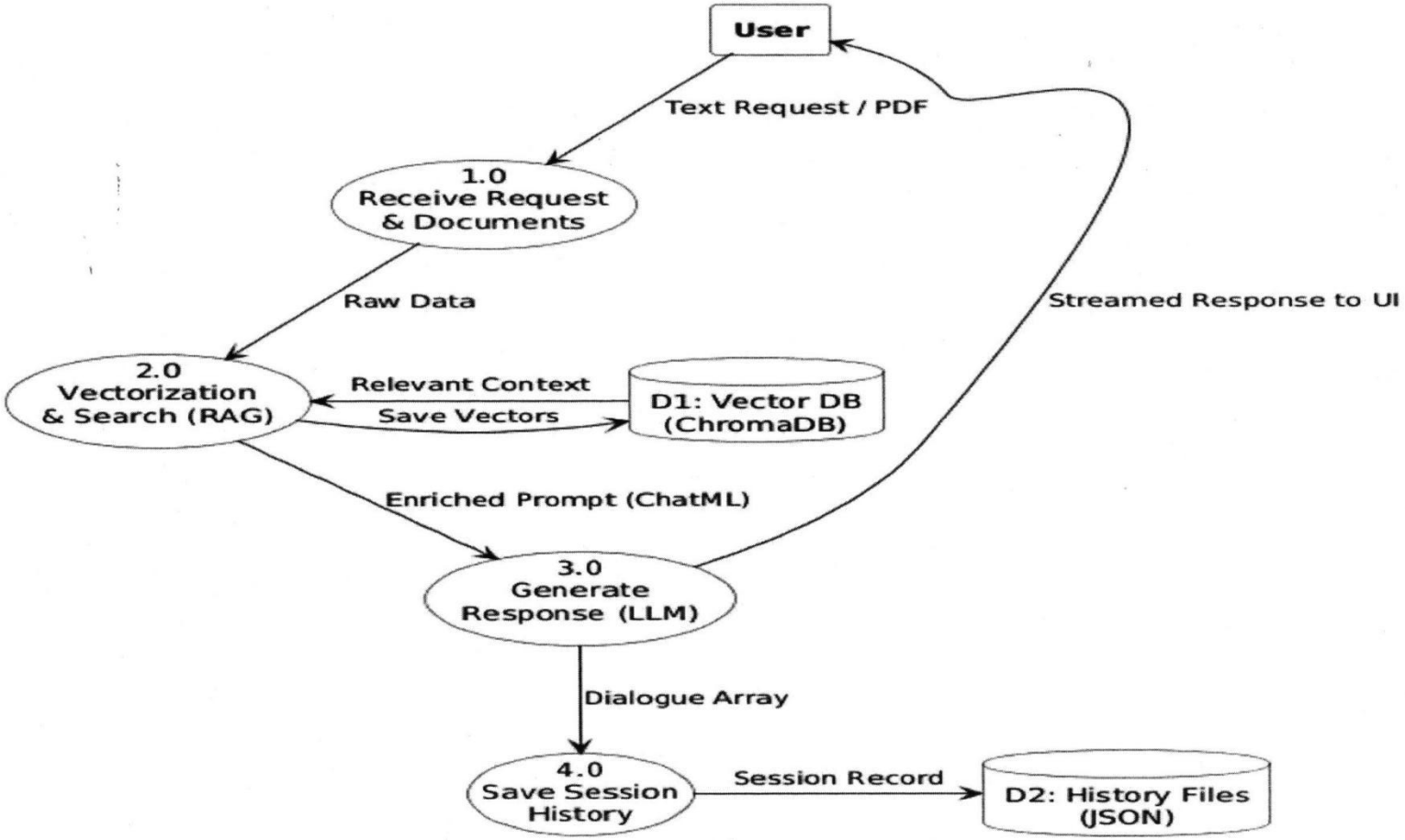
Дякую за увагу!

Парфутік Микола Олександрович

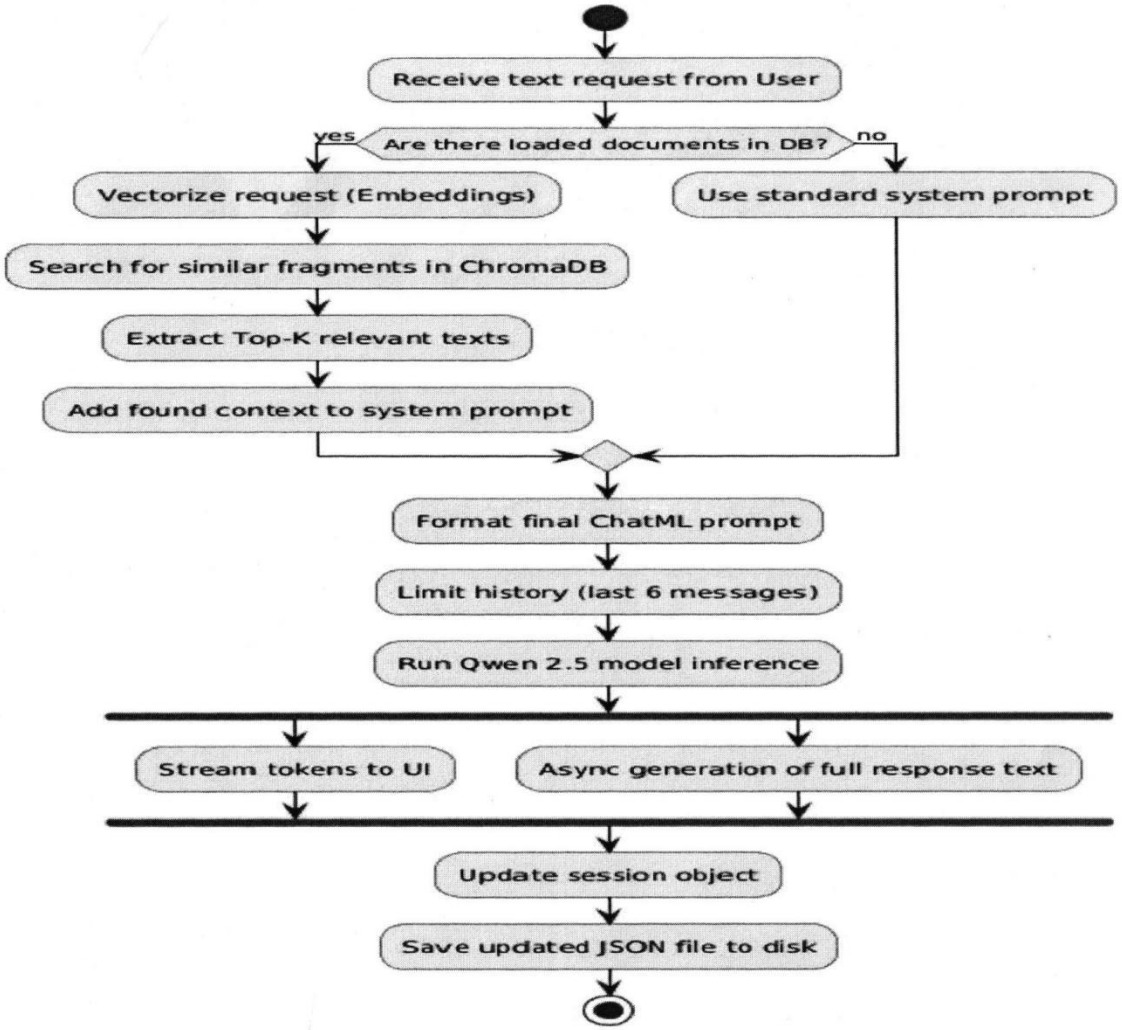
ГРАФІЧНА ЧАСТИНА



Зм.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.2201102.01.11Е8		
Розробив		Парфутік М. О.			Локальна велика мовна модель для автономних комп'ютерних пристроїв Діаграма класів	Літера	Маса / Масштаб
Керівник		Праворська Н. І.					
Консульт.						Аркуш 1	Аркушів 3
Н. Контр.		Бедратюк Г. І.				ХНУ. ІПЗ-22-1	
Зав. каф.		Бедратюк Л. П.					



					КвРІПЗ.2201102.01.11Е8	
Зм.	Арк.	№ докум.	Підпис	Дата	Локальна велика мовна модель для автономних комп'ютерних пристроїв DFD-діаграма	Літера / Маса / Масштаб
Розробив		Парфутік М. О.				
Керівник		Праворська Н. І.				Аркуш 2 / Аркушів 3
Консульт.						
Н. Контр.		Бедратюк Г. І.				ХНУ. ІПЗ-22-1
Зав. каф.		Бедратюк Л. П.				



					КвРІПЗ.2201102.01.11Е8	
Зм.	Арк.	№ докум.	Підпис	Дата	Локальна велика мовна модель для автономних комп'ютерних пристроїв Діаграма процесів	Літера / Маса / Масштаб
Розробив		Парфутік М. О.		29.03.26		
Керівник		Праворська Н. І.		29.03.26		Аркуш 3 / Аркушів 3
Консульт.						
Н. Контр.		Бедратюк Г. І.		29.03.26	ХНУ. ІПЗ-22-1	
Зав. каф.		Бедратюк Л. П.		29.03.26		

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Парфутіка Миколи Олександровича
факультет ІТ, IVкурс, група ІПЗ-22-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

02.09.2025
дата

Парфутік
підпис

Завідувачу кафедри
інженерії програмного забезпечення
проф. Бедратюку Л. П.
студента групи ІПЗ-22-1
Парфутік М. О.
Прізвище, ініціали

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення»: «Локальна велика мовна модель для автономних комп'ютерних пристроїв»

(керівник роботи – Праворська Наталія Іванівна)
Прізвище, ім'я, по батькові

03.02.2026
Дата

Підпис студента

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ДЕКЛАРАЦІЯ УЧАСНИКА ОСВІТНЬОГО ПРОЦЕСУ

щодо дотримання академічної доброчесності

Цією декларацією я, Парфутік Микола Олександрович,

студент IV курсу спеціальності 121 – Інженерія програмного забезпечення, група ІПЗ-22-1

здобувач вищої освіти (шифр та назва спец-ті, курс, академічна група)

підтверджую, що ознайомився (-лась) з Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті та Кодексом академічної доброчесності і **зобов'язуюсь** дотримуватися їх вимог під час освітнього процесу, проведення наукової діяльності, виконання організаційно-адміністративних функцій тощо.

Усвідомлюю, що у разі порушення мною принципів академічної доброчесності нестиму відповідальність перед академічною спільнотою ХНУ згідно з нормами, визначеними Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, законодавства України.

02 вересня 2025 р. Парфутік

Підпис

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Парфутік Микола Олександрович

Тема Локальна велика мовна модель для автономних комп'ютерних пристроїв

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень _____; кількість сторінок записки _____

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі досліджено та проаналізовано актуальну предметну область впровадження систем штучного інтелекту на базі великих мовних моделей (LLM) для локальної роботи на автономних пристроях з обмеженими ресурсами. Проведено аналіз існуючих на ринку рішень, виявлено їхні недоліки в контексті конфіденційності та залежності від інтернет-з'єднання, що довело необхідність розробки концепції Edge AI. Виконано проєктування та розробку модульного програмного забезпечення (веб-застосунку BarSik AI) з використанням мови Python, фреймворків Streamlit та LangChain. Для оптимізації роботи нейромережі застосовано методи 4-бітної квантизації (Q4_K_M), апаратного прискорення Apple MPS та реалізовано архітектуру RAG із локальною векторною базою даних ChromaDB. За результатами проведеного тестування підтверджено, що розроблена система генерує відповіді без галюцинацій та демонструє комфортну для користувача швидкодію (TPS) на споживчому обладнанні.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана в повному обсязі, повністю відповідає поставленому завданню, розкриває мету та відповідає всім вимогам, що висуваються до випускних кваліфікаційних робіт бакалавра.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі обґрунтовано актуальність розробки локальних ШІ-систем для збереження приватності, визначено мету та завдання дипломного проєктування. У першому розділі проведено детальний аналіз предметної області (еволюції HCI), розглянуто проблематику використання хмарних ШІ-сервісів (ризики витоку даних, мережеві затримки) та визначено жорсткі функціональні й нефункціональні вимоги до розроблюваного ПЗ. У другому розділі здійснено проєктування системи: обґрунтовано вибір модульної багатошарової архітектури, яка ефективно розділяє презентаційний шар, ядро інференсу та логіку векторизації. Спроєктовано структуру векторного сховища для RAG-конвеєра та локального JSON-сховища історії сесій. У третьому розділі описано безпосередню програмну реалізацію застосунку з використанням ядра llama.cpp. Наведено

результати тестування швидкодії (токенів за секунду) та використання оперативної пам'яті на різних апаратних платформах (CPU та GPU), що підтвердило ефективність застосованого алгоритму «ковзного вікна» та методів стиснення ваг.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є надзвичайно актуальною на тлі сучасних трендів забезпечення інформаційної безпеки корпоративних даних та розвитку Edge AI. Позитивним та інноваційним є застосування передових методів оптимізації нейромереж, що дозволило запустити ресурсоємну модель Qwen 2.5 в умовах обмеженої оперативної пам'яті звичайних ПК. Особливої уваги заслуговує успішна інтеграція локального RAG-конвеєра (Retrieval-Augmented Generation), що дозволяє моделі працювати із закритими документами користувача в умовах повної ізоляції (Air-gapped середовища).

5. Негативні сторони роботи У розробленому застосунку реалізовано розпізнавання та парсинг лише базових текстових та PDF/DOCX документів для векторної бази даних. У подальшому було б доцільно розширити конвеєр для підтримки сканованих копій (через OCR) або табличних даних Excel. Також графічний інтерфейс можна було б доповнити меню для динамічного завантаження та перемикання між різними LLM-моделями безпосередньо у програмі. Зазначені недоліки не впливають на загальний високий рівень розробки.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення, що включає архітектурні діаграми, скріншоти програмних модулів та графіки результатів тестування, виконано на високому професійному рівні. Воно логічно доповнює текст. Пояснювальна записка структурована, написана технічно грамотною мовою та оформлена згідно з вимогами чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота є завершеним, цілісним інженерним дослідженням, що має вагоме практичне значення. Розроблений програмний продукт повністю готовий до практичної експлуатації. Дипломник продемонстрував ґрунтовні знання сучасних технологій штучного інтелекту, архітектурного проєктування та високий рівень володіння мовою програмування Python. Робота заслуговує на найвищу позитивну оцінку.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі, вирізняється науково-технічною новизною та заслуговує на оцінку «відмінно».

РЕЦЕНЗЕНТ Мартинюк Валерій Володимирович, доктор технічних наук, професор, професор кафедри автоматизації, комп'ютерно-інтегрованих технологій та робототехніки

"25" травня 2026 р.

(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів.
Назва кваліфікаційної роботи: «Локальна велика мовна модель для автономних комп'ютерних пристроїв»
Автор: Парфутік Микола Олександрович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Праворська Наталія Іванівна, кандидат педагогічних наук, доцент
Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована.	
3	Виявлені запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Виявлені запозичення частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:
Запозичення, виявлені у роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системою перевірки на плагіат Anti-Plagiarism виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках, у структурі змісту, назвах розділів/підрозділів, рамках форм, у назвах та URL-адресах публікацій переліку джерел посилання;
2) запозичення, виявлені у тексті роботи, є фрагментарними.

Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 1.0% з одного джерела. Загальна сумарна подібність у базі даних складає 4% за символами та 9% за лексемами. Крім того, за результатами додаткового аналізу Коефіцієнт подібності 1 становить 4.34%, Коефіцієнт подібності 2 - 1.11%. Виявлені системою попередження щодо стилістики тексту (73 мікропробіли, 98 білих знаків та 2 заміни букв) є виключно наслідком технічного форматування документа і не є маніпуляціями з метою приховування плагіату. З урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 27.05.26

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Наталія ПРАВОРСЬКА

Wed May 20 01:06:50 EEST 2026, Форкун Юрій Вікторович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (http://ap.km.ua) v-16.718

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: UA, US, RU. **Помилок в документах: 15%**

ID: 271751 Назва: БКР_Локальна велика мовна модель для автономних комп'ютерних пристроїв Додано в БД: 2026-05-20 Автора: Микола ПАРФУТІК Керівники: канд. пед. наук, доцент Наталія ПРАВОРСЬКА Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	86324	517	3561 (4%)	45 (9%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

SemanticAI for Education

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

першого освітнього рівня «Бакалавр»

Студента: Парфутіка Миколи Олександровича
Група: ІПЗ-22-1

Тема: *«Локальна велика мовна модель для автономних комп'ютерних пристроїв»*

Спеціальність: 121 – Інженерія програмного забезпечення

Короткий зміст пояснювальної записки

У вступі розглядається актуальність теми, пов'язана зі стрімким розвитком інформаційних технологій та необхідністю впровадження Edge AI для забезпечення конфіденційності та автономності в обробці природної мови. Метою роботи є створення інтуїтивно зрозумілого програмного засобу на базі автономної мовної моделі, що забезпечує високу продуктивність та конфіденційність. Для досягнення цієї мети визначено кілька завдань, серед яких аналіз предметної області, вибір оптимальних методів оптимізації, проектування архітектури системи, розробка графічного інтерфейсу та тестування програмного забезпечення.

Відповідність отриманих результатів роботи поставленим завданням

Завдання, сформульовані у вступі, включають аналіз предметної області, вибір методів оптимізації, проектування архітектури, обґрунтування вибору технологій, програмну реалізацію та тестування. Висновки роботи підтверджують виконання цих завдань, оскільки результати демонструють розробку автономної мовної моделі, реалізацію програмного забезпечення та проведення тестування. Отже, отримані результати **повністю відповідають** поставленим завданням.

Оцінка розділів

Розділ 1: Аналіз предметної області

У цьому розділі детально розглядається сучасний стан розвитку інтерфейсів взаємодії людини з комп'ютером, зокрема через обробку природної мови. Описано еволюцію HCI, акцентуючи на важливості переходу до нових архітектур. Вказано на проблеми централізованих хмарних рішень та пропонується концепція Edge AI. Однак, відсутні конкретні приклади реалізації та деталі застосування методів стиснення моделей. Рекомендується додати приклади успішних реалізацій Edge AI та детальніше описати методи стиснення.

Розділ демонструє глибоке розуміння предметної області, але потребує доопрацювання в частині конкретизації прикладів та реалізації.

Розділ 2: Проєктування структури та компонентів програмного забезпечення

У цьому розділі розглядається архітектура програмного забезпечення, декомпозиція системи, залежності між модулями та інтерфейсами. Описано модульну архітектуру, але відсутнє порівняння з альтернативними архітектурами. Також не наведено аналіз шаблонів проєктування та міжмодульних залежностей. Водночас, структура системи описана чітко, а вибір технологій обґрунтовано.

Розділ містить корисну інформацію, але має суттєві недоліки в частині порівняння архітектур та опису залежностей, що потребує доопрацювання.

Розділ 3: Програмна реалізація та тестування програмного забезпечення

У цьому розділі детально описано реалізацію модулів, їх узгодженість з архітектурою, а також проведення тестування. Однак, відсутня детальна структура бази даних, а також не описано реалізацію тригерів і збережених процедур. Вимоги до системи не вказані чітко, а методи тестування лише перелічені без обґрунтування.

Розділ містить детальний опис реалізації та тестування, проте є деякі недоліки, які потребують доопрацювання, зокрема в частині структури бази даних та візуалізації результатів тестування.

Позитивні сторони

Робота демонструє оригінальність у підході до розробки автономної мовної моделі, що відповідає сучасним вимогам безпеки та конфіденційності. Висока якість рішень, зокрема в проектуванні архітектури та реалізації модулів, свідчить про глибоке розуміння предметної області. Зручність для користувача підкреслюється детальним описом графічного інтерфейсу.

Недоліки

Серед недоліків можна відзначити відсутність конкретних прикладів реалізації в розділі аналізу предметної області, недостатню структурованість вимог у вигляді таблиць, а також відсутність детального опису бази даних у розділі програмної реалізації. Також не вистачає візуалізацій для результатів тестування, що ускладнює сприйняття даних.

Відгук в цілому

Тема роботи є актуальною та має практичну значущість, оскільки відповідає сучасним вимогам до автономності та безпеки в обробці природної мови. Зміст роботи відповідає темі та завданню, а новизна ідей та рішень підкреслює інноваційний підхід до розробки. Об'єктивність та обґрунтованість викладеного матеріалу є на високому рівні. Програмний продукт демонструє працездатність та відповідає технічному завданню, хоча потребує деяких доопрацювань.

Оцінка кваліфікаційної роботи

Кваліфікаційна робота заслуговує оцінки **добре**. Вона виконана в повному обсязі з дотриманням основних вимог, проте має деякі недоліки, які потребують доопрацювання. Здобувач володіє матеріалом, грамотно викладає суть роботи, хоча може припускатися дрібних неточностей або потребує підказок при відповідях.

Рекомендації

Рекомендується доопрацювати розділи, що містять недоліки, зокрема додати конкретні приклади реалізації, структуру вимог у вигляді таблиць, а також візуалізації для результатів тестування. Це підвищить якість роботи та зробить її більш зрозумілою для читача.

OpenAI API-асистент
Session ID: 63e28868-3d7b-480a-a5e7-de3b43bb8c83
Підписано автоматично, модель gpt-4o-mini
Дата: 15.05.2026

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Микола ПАРФУТІК

Співавтор:

Назва: Локальна велика мовна модель для автономних комп'ютерних пристроїв

Науковий керівник: к. пед. наук, доцент Наталія ПРАВОРСЬКА

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:4.34%

Коефіцієнт подібності 2:1.11%

Мікропробіли: 73

Заміна букв: 2

Інтервали: 0

Білі знаки: 98

Дата створення звіту: 2026-05-20 00:50:43.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата експерт

25.05.26