Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Метод інтеграції компонентів видобування знань (RAG) у мультиагентні програмні системи для забезпечення контекстно-залежного прийняття рішень

Рівень вищої освіти Другий (магістерський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.240163.01.10.ПЗ

Виконав студент 2 курсу, група ІПЗм-23-1 _____ Олександр КАТРЕВИЧ
Підпис Ім'я, ПРІЗВИЩЕ

Керівник д-р фіз.-мат. наук, професор _____ Леонід БЕДРАТЮК
Науковий ступінь, звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер ст. викладач _____ Ганна БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

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

Дата 15 грудня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Інженерії програмного забезпечення

Освітній рівень магістр

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного програмного забезпечення»

Освітня програма «Інженерія програмного програмного забезпечення»

ЗАТВЕРДЖУЮ:

Завідувач кафедри ІПЗ

Л. П. Бедратюк

01.09.2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Катревичу Олександру Богдановичу

Прізвище, ім'я, по батькові студента (у давальному відмінку)

1. Тема роботи Метод інтеграції RAG-компонентів у мультиагентні системи

Керівник роботи д-р фіз.-мат. наук, професор Бедратюк Л. П.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджено наказом ректора ХНУ від 25.08.2025 р. № 65

2. Строк подання студентом роботи на кафедру: 01.12.2025 р.

3. Вихідні дані до роботи Матеріали науково-дослідної практики

4. Зміст пояснювальної записки (перелік питань, що належить розробити)

1. Аналіз стану досліджень у галузі

2. Теоретичні основи методу

3. Проектування та реалізація мультиагентної системи

4. Програмна реалізація та експериментальна перевірка методу

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди)

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Антиплагіат	Форкун Ю. В.	17.11.2025	07.12.2025
Нормоконтроль	Бедратюк Г. І.		9.12.2025

7. Дата видачі завдання «01» вересня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вивчення предметної області, формулювання мети та задач дослідження, визначення об'єкта та предмета дослідження	20.10-26.10.2025	
2	Робота над розділом 1 – вивчення джерел, аналіз стану досліджень у предметній галузі, визначення задач, висновки	20.10-26.10.2025	
3	Робота над розділом 2 – дослідження теоретичних основ, визначення підходів до реалізації методу, висновки	27.10-02.11.2025	
4	Робота над матеріалами до науково-практичної конференції	03.11-09.11.2025	
5	Робота над розділом 3 – проектування системи, вибір архітектури. технологічної платформи та програмних засобів для реалізації	10.11-16.11.2025	
6	Робота над розділом 4 – робота над програмною реалізацією та	17.11-23.11.2025	

	експериментальною перевіркою методу		
7	Попередній захист кваліфікаційної роботи	24.11.2025-31.11.2025	
8	Узгодження постановки задач, отриманих результатів та висновків, написання вступу, загальних висновків, оформлення джерел та додатків, оформлення записки та графічних матеріалів відповідно до вимог	01.12-07.12.2025	
9	Перевірка роботи на наявність плагіату, нормоконтроль, брошурування пояснювальної записки, підготовка супровідних документів	08.12-14.12.2025	
10	Підготовка до захисту кваліфікаційної роботи	з 15.12.2025	

Студент _____ Катревич О. Б.
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ Бедратюк Л. П.
Підпис Ініціали, прізвище

РЕФЕРАТ

Тема кваліфікаційної роботи: Метод інтеграції RAG-компонентів у мультиагентні системи.

Автор роботи: Катревич Олександр Богданович.

Керівник роботи: Бедратюк Леонід Петрович.

Пояснювальна записка: 102 с., 20 рис., 3 дод., 41 джерело.

Графічна частина: 18 слайдів.

RAG, МУЛЬТИАГЕНТНА СИСТЕМА, ІНТЕГРАЦІЯ, GPT, ВИТЯГ ЗНАНЬ, LLM, LANGCHAIN, COORDINATION, RETRIEVAL, ЖИТТЄВИЙ ЦИКЛ, ІНТЕРФЕЙС, ПРОТОТИП, ТРАСОВАНІСТЬ.

Об’єктом дослідження є мультиагентні системи програмного забезпечення з вбудованими модулями генерації відповідей на основі зовнішніх джерел знань (RAG-компонентів).

Предметом дослідження є метод і засоби інтеграції RAG-компонентів у мультиагентні системи, що забезпечують цілісну поведінку агентів, релевантність відповідей, узгодженість даних і якісну взаємодію із зовнішніми базами знань.

Мета дослідження – розробка методу інтеграції компонентів генерації відповідей з доступом до зовнішніх джерел знань (RAG-компонентів) у мультиагентні системи програмного забезпечення з урахуванням принципів узгодженості, масштабованості та забезпечення якості функціонування програмного продукту.

У процесі виконання кваліфікаційної роботи було проаналізовано сучасні підходи до реалізації мультиагентних систем із використанням моделей великої мовної обробки та механізмів витягування знань (retrieval). Виокремлено типові архітектурні шаблони та проблемні місця при поєднанні RAG-компонентів із агентною координацією. Проведено порівняльний аналіз існуючих підходів і сформульовано вимоги до нового методу інтеграції, зокрема з урахуванням життєвого циклу ПЗ, повторного використання знань і стабільності відповідей.

У результаті дослідження було розроблено архітектурну модель агента з інтегрованим RAG-компонентом. На її основі реалізовано прототип мультиагентної системи, що дозволяє забезпечити структурну незалежність витягування знань, масштабованість агентної взаємодії та трасованість відповідей.

Для реалізації програмного прототипу використано мову Python та бібліотеки LangChain, FAISS, CrewAI, OpenAI API. Розроблена система пройшла перевірку на прикладних сценаріях, що охоплюють взаємодію агентів генерації, координації та перевірки відповіді з доступом до зовнішнього контексту.

Запропонований метод дозволяє ефективно інтегрувати LLM-знання у агентну архітектуру, підвищує адаптивність системи до змін у базі знань і забезпечує зростання якості відповіді у порівнянні з типовими реалізаціями.

Результати дослідження можуть бути використані у розробці інтелектуальних систем нового покоління, зокрема у задачах автоматизації бізнес-процесів, підтримки рішень, семантичного пошуку та ШІ-аналітики. Запропоноване рішення має потенціал до масштабування та подальшого розвитку в напрямі автономних агентних систем з гнучкою координацією та інтеграцією нових джерел знань.

08. 12. 25 р
Дата

Підпис

ABSTRACT

Thesis title: Method of Integrating RAG Components into Multi-Agent Systems.

Author: Katrevych Oleksandr Bohdanovych.

Supervisor: Bedratiuk Leonid Petrovych.

Explanatory note: 102 pages, 20 figures, 3 appendices, 41 references.

Keywords: RAG, MULTI-AGENT SYSTEM, INTEGRATION, GPT, KNOWLEDGE RETRIEVAL, LLM, LANGCHAIN, COORDINATION, RETRIEVAL, LIFE CYCLE, INTERFACE, PROTOTYPE, TRACEABILITY.

The object of the study is multi-agent software systems with embedded modules for generating responses based on external knowledge sources (RAG components).

The subject of the study is the method and means of integrating RAG components into multi-agent systems that ensure coherent agent behavior, response relevance, data consistency, and high-quality interaction with external knowledge bases.

The purpose of the research is to develop a method for integrating response generation components with access to external knowledge sources (RAG components) into multi-agent software systems, taking into account the principles of consistency, scalability, and quality assurance of software functioning.

During the execution of the thesis, modern approaches to the implementation of multi-agent systems using large language models (LLMs) and retrieval mechanisms were analyzed. Typical architectural patterns and bottlenecks in combining RAG components with agent coordination were identified. A comparative analysis of existing approaches was carried out, and requirements for a new integration method were formulated, taking into account the software life cycle, knowledge reuse, and response stability.

As a result of the study, an architectural model of an agent with an integrated RAG component was developed. Based on this model, a prototype of a multi-agent system was implemented, enabling structural independence of knowledge retrieval, scalability of agent interactions, and traceability of responses.

For the implementation of the software prototype, the Python programming language and libraries such as LangChain, FAISS, CrewAI, and OpenAI API were used.

The developed system was tested in application scenarios covering the interaction of generation, coordination, and validation agents with access to external context.

The proposed method makes it possible to effectively integrate LLM knowledge into agent architectures, increases the adaptability of the system to changes in the knowledge base, and improves response quality compared to typical implementations.

The results of the research can be applied to the development of next-generation intelligent systems, particularly in tasks of business process automation, decision support, semantic search, and AI analytics. The proposed solution has significant potential for scalability and further development towards autonomous agent systems with flexible coordination and integration of new knowledge sources.

08.12.25г
Date


Signature

ЗМІСТ

ВСТУП

Сучасний розвиток штучного інтелекту визначається широким поширенням великих мовних моделей (Large Language Models, LLM), які поступово перетворилися з експериментальних досліджень у фундаментальний інструмент для створення інтелектуальних програмних систем. Такі моделі застосовуються в пошукових сервісах, чат-ботах, аналітичних платформах та корпоративних рішеннях [3; 7; 12]. Проте ключовою проблемою залишається їхня обмежена здатність працювати з актуальними та достовірними знаннями. Відомо, що LLM часто продукують помилкові твердження (так звані “галюцинації”), що значно обмежує можливості їх практичного використання [18; 22].

Одним із найперспективніших рішень цієї проблеми стала архітектура Retrieval-Augmented Generation (RAG), яка поєднує генеративні можливості моделей із пошуковими механізмами та зовнішніми базами знань. Концепцію RAG уперше було системно описано в роботі Lewis та ін. [9], після чого вона набула широкого поширення у науковій і промисловій практиці. Сучасні дослідження демонструють, що використання RAG значно підвищує релевантність і достовірність відповідей моделей, зменшує кількість помилкових тверджень та забезпечує відтворюваність результатів у прикладних сценаріях [11; 17; 20].

Паралельно розвивається інший напрямок – мультиагентні системи (MAS), які мають глибокі традиції у програмній інженерії. Вони ґрунтуються на ідеї взаємодії автономних агентів, здатних координувати свої дії, співпрацювати та розподіляти завдання для досягнення спільної мети [5; 14]. З появою LLM ці системи отримали новий імпульс розвитку: сучасні проєкти AutoGen [25], CAMEL [26], CrewAI [27], MetaGPT [28] демонструють, що поєднання агентної архітектури з генеративними моделями дозволяє реалізовувати багатокрокові сценарії планування, аналітики та автоматизованої взаємодії з користувачем. Огляд сучасної літератури [29; 31; 34] показує, що мультиагентні LLM-системи розглядаються як основа для наступного покоління інтелектуальних програмних платформ.

Разом із тим постає проблема інтеграції RAG-компонентів у мультиагентні архітектури. Більшість існуючих рішень розраховані на роботу з одним агентом, тоді як у багатокомпонентному середовищі виникають нові виклики. Серед них – необхідність узгодженості дій агентів, що одночасно звертаються до баз знань, розмежування функціональної відповідальності між агентами (витягування, генерація, валідація, координація), забезпечення масштабованості при зростанні кількості агентів та обсягів даних, а також потреба у трасованості й відтворюваності результатів у рамках життєвого циклу програмного забезпечення [16; 21; 32]. У науковій літературі ці питання висвітлені лише частково: більшість робіт описує загальні підходи до інтеграції RAG або архітектуру конкретних агентних систем, але майже відсутні комплексні методи, орієнтовані саме на багатокомпонентні архітектури. У результаті виникає проблемна ситуація: сучасні мультиагентні системи з LLM демонструють високу гнучкість, але їхня інтеграція з RAG-компонентами є фрагментарною та не забезпечує гарантованої якості функціонування.

З огляду на зростаючий попит на надійні та масштабовані інтелектуальні системи, проблема інтеграції RAG у мультиагентні середовища є актуальною для інженерії програмного забезпечення. Вона безпосередньо стосується процесів розробки архітектури, забезпечення якості, супроводу й модернізації програмних продуктів. Вирішення цієї проблеми сприятиме створенню нового покоління програмних систем, які поєднують переваги генеративних моделей і мультиагентної координації.

Мета дослідження – розробка методу інтеграції RAG-компонентів у мультиагентні системи програмного забезпечення з урахуванням принципів узгодженості, масштабованості та забезпечення якості функціонування.

Для досягнення поставленої мети сформульовано такі задачі:

1) проаналізувати сучасні підходи до реалізації RAG-архітектури та мультиагентних систем, визначити їх переваги і обмеження;

2) виявити проблемні аспекти інтеграції RAG-компонентів у мультиагентні архітектури;

3) сформулювати вимоги до методу інтеграції, що враховують специфіку життєвого циклу ПЗ;

4) розробити архітектурну модель агента з інтегрованим RAG-компонентом;

5) реалізувати експериментальний прототип мультиагентної системи з RAG-компонентами;

6) оцінити ефективність методу шляхом порівняння з існуючими рішеннями на основі обґрунтованих метрик.

Об’єкт дослідження – мультиагентні системи програмного забезпечення з вбудованими модулями генерації відповідей на основі зовнішніх джерел знань (RAG-компонентів).

Предмет дослідження – метод і засоби інтеграції RAG-компонентів у мультиагентні системи, що забезпечують цілісну поведінку агентів, релевантність відповідей, узгодженість даних і якісну взаємодію із зовнішніми базами знань.

У роботі використано такі методи дослідження:

1) структурно-функціональний аналіз для вивчення існуючих архітектур MAS і RAG;

2) методи компонентного та архітектурного проєктування для побудови моделі інтеграції;

3) експериментальне моделювання та прототипування з використанням Python, LangChain, FAISS, CrewAI, OpenAI API;

4) порівняльне тестування для оцінки ефективності запропонованого методу за метриками релевантності, узгодженості та стабільності;

5) методи трасування та логування для перевірки відтворюваності результатів у життєвому циклі системи.

Наукова новизна отриманих результатів:

1) вперше запропоновано метод інтеграції RAG-компонентів у мультиагентні системи, що забезпечує структурну незалежність витягування та генерації знань із гарантованою релевантністю відповідей;

2) удосконалено архітектурну модель мультиагентної системи шляхом введення інтерфейсного рівня абстракції для підключення RAG-компонентів, що підвищує гнучкість і масштабованість системи;

3) отримано подальший розвиток підходу до оцінювання якості функціонування мультиагентних систем із RAG-компонентами через уточнення показників узгодженості, релевантності та стабільності.

Практичне значення отриманих результатів – розроблений метод інтеграції може бути використаний:

1) у створенні мультиагентних платформ для інтелектуальних сервісів підтримки рішень, бізнес-аналітики, автоматизації документообігу;

2) у вдосконаленні існуючих RAG-рішень (LangChain, AutoGen, CrewAI) через підвищення стабільності відповідей та масштабованості;

3) як основа для подальших досліджень у галузі програмної інженерії, пов’язаних із життєвим циклом систем, що інтегрують LLM і зовнішні бази знань.

Запропонований метод сприяє підвищенню ефективності процесу розробки й супроводу програмного забезпечення, зменшує ризики неконсистентності знань і забезпечує трасованість результатів. Практична апробація прототипу довела можливість його застосування у реальних сценаріях із перспективою подальшого масштабування.

1 АНАЛІЗ СТАНУ ДОСЛІДЖЕНЬ У ГАЛУЗІ МУЛЬТИАГЕНТНИХ СИСТЕМ ТА RETRIEVAL-AUGMENTED GENERATION

1.1 Загальні методологічні підходи до побудови інтелектуальних агентних систем

Історія розвитку мультиагентних систем у програмній інженерії відображає загальні тенденції становлення штучного інтелекту як дисципліни. Перші ідеї колективного використання обчислювальних ресурсів виникли ще у 1970–1980-х роках у рамках досліджень у сфері розподіленого штучного інтелекту. Тоді науковці прагнули навчитися розподіляти завдання між кількома обчислювальними вузлами та синхронізувати їхню роботу. У 1990-х роках поняття агента почало набувати сучасного змісту: агент визначався як автономна програмна сутність, здатна сприймати середовище, приймати рішення та діяти відповідно до поставлених цілей. Цей етап характеризувався появою перших універсальних платформ, зокрема JADE, які дозволяли створювати експериментальні прототипи для галузей електронної комерції, логістики чи управління ресурсами.

Початок 2000-х років ознаменувався переходом від невеликих дослідницьких систем до масштабованих і відкритих середовищ. Важливим кроком стало формування стандартів FIPA, які забезпечили єдиний підхід до опису поведінки агентів та їхніх комунікаційних протоколів. Це дало змогу різним розробникам створювати сумісні компоненти та поєднувати їх у гетерогенні системи. Пізніше, у 2010-х роках, мультиагентні системи почали активно інтегруватися з методами машинного навчання. Особливої ваги набули завдання адаптивності та самоорганізації, коли агенти повинні були не лише виконувати наперед визначені ролі, а й динамічно змінювати свою поведінку залежно від стану середовища чи дій інших агентів.

Сучасний етап розвитку пов’язаний із поєднанням мультиагентних підходів із технологіями великих мовних моделей. З’явився феномен генеративних агентів, здатних відтворювати елементи людської поведінки, зберігати довготривалу пам’ять та взаємодіяти між собою через природну мову. Якщо раніше агентні

системи здебільшого використовувалися для моделювання або симуляцій, то тепер вони стають практичним інженерним рішенням для автоматизації бізнес-процесів, управління знаннями та інтелектуальної підтримки прийняття рішень. У такий спосіб відбувається перехід від вузькоспеціалізованих програмних засобів до багатофункціональних когнітивних систем.

Фундаментальним підґрунтям інженерії мультиагентних систем є принципи, що визначають способи взаємодії окремих агентів. Перш за все, кожен агент розглядається як автономний елемент, який самостійно визначає свої дії на основі сприйняття середовища та власних цілей. Автономність дозволяє створювати системи, що не потребують постійного централізованого управління, а отже, мають вищу стійкість до збоїв і гнучкіше реагують на зміни. Проте ефективність досягається не лише завдяки незалежності, але й через кооперацію, адже у більшості практичних випадків складні завдання потребують узгоджених дій кількох агентів. Саме кооперація відкриває можливості для розподілу ролей, колективного планування та побудови коаліцій.

Не менш важливою є координація, яка дає змогу уникати конфліктів і забезпечувати узгодженість у поведінці системи загалом. Залежно від предметної області координація може мати різні форми: від простого синхронного обміну повідомленнями до складних алгоритмів оптимізації планів. Нарешті, основою будь-якої взаємодії виступає комунікація. Класичні мультиагентні системи використовували спеціалізовані мови, наприклад FIPA ACL, де кожне повідомлення мало чітку семантику. Нині все більшого значення набуває обмін повідомленнями природною мовою, що розширює можливості взаємодії, але водночас вимагає нових методів контролю узгодженості знань.

Одним із найбільш актуальних викликів у проєктуванні інтелектуальних агентів є інтеграція зовнішніх знань. Традиційно агенти отримували доступ до централізованих баз знань, які містили факти, правила чи онтології. Цей підхід був зручним для невеликих систем, але його недоліком виявилася обмежена гнучкість: оновлення знань потребувало централізованої підтримки. Подальший розвиток привів до децентралізованих моделей, де кожен агент зберігав власну базу знань і

міг обмінюватися даними з іншими. Така схема відкривала більше можливостей для самоорганізації, але створювала труднощі з перевіркою достовірності та узгодженості інформації.

Сучасні тенденції свідчать, що найбільш перспективним напрямом є використання зовнішніх інформаційних ресурсів у режимі реального часу. Це можуть бути веб-сервіси, корпоративні бази даних або графи знань. Актуальною стала потреба у створенні уніфікованих інтерфейсів доступу, механізмів перевірки якості даних та їхньої синхронізації з агентними протоколами. У цьому контексті особливе значення має підхід Retrieval-Augmented Generation, що поєднує мовні моделі з можливістю пошуку в зовнішніх джерелах. Завдяки RAG агент спочатку отримує релевантні документи, а потім використовує їх у процесі генерації відповіді, що значно підвищує точність та пояснюваність результатів.

Поєднання RAG з мультиагентними системами створює нові можливості: окремі агенти можуть спеціалізуватися на пошуку, інші – на аналізі чи валідації, тоді як генерація кінцевої відповіді здійснюється колективними зусиллями. Це формує новий рівень гнучкості та надійності інтелектуальних систем, де знання інтегруються не централізовано, а завдяки динамічній співпраці спеціалізованих агентів.

Загальні методологічні підходи до побудови інтелектуальних агентних систем базуються на поєднанні кількох напрямів: історично сформованих принципів автономності та кооперації, сучасних методів координації та комунікації, а також новітніх технологій інтеграції знань. Перехід від класичних моделей до когнітивних мультиагентних систем, здатних працювати з зовнішніми джерелами інформації у реальному часі, свідчить про якісно новий етап розвитку цієї галузі. Саме тут виникає потреба у вдосконаленні методів інтеграції RAG-компонентів, що і визначає подальшу логіку дослідження. На рисунку 1.1 показано етапи розвитку мультиагентних систем у програмній інженерії.

Рисунок 1.1 – Етапи розвитку мультиагентних систем у програмній інженерії

1.2 Архітектурні підходи до реалізації RAG-систем

Концепція Retrieval-Augmented Generation (RAG) виникла як відповідь на обмеження великих мовних моделей, пов’язані з їхньою схильністю до «галюцинацій» та неможливістю безпосередньо працювати з актуальними чи доменними знаннями. На відміну від класичних LLM, які ґрунтуються виключно на знаннях, закладених під час навчання, RAG-підхід передбачає поєднання генеративної моделі з зовнішнім механізмом пошуку інформації. Це означає, що перед формуванням відповіді модель отримує доступ до зовнішньої бази даних або документів, які забезпечують контекст і гарантують релевантність результату. У такий спосіб генерація стає «підкріпленою» зовнішніми фактами, що значно підвищує точність і довіру до системи.

Місце RAG у сучасних LLM-системах визначається його роллю як проміжного шару між користувачем і самою моделлю. З одного боку, генеративна модель залишається ядром, що відповідає за формування зв’язного тексту, здатного відображати складні логічні структури. З іншого боку, механізм пошуку виступає «мостом» до динамічного знання, яке не може бути зафіксоване у параметрах моделі. Саме ця комбінація робить RAG надзвичайно цінним у завданнях, де

потрібна висока актуальність: у правових системах, медичних застосунках, фінансовій аналітиці чи корпоративних чат-ботах. У програмній інженерії та при побудові мультиагентних систем RAG дає змогу суттєво зменшити ризик некоректних відповідей та забезпечити повторюваність результатів.

Архітектурні підходи до реалізації RAG-систем різняться залежно від того, як саме організовано пошук і як формується база знань. Одним із перших і найбільш поширених став підхід Dense Retrieval, що базується на використанні векторних подань тексту. Ідея полягає у тому, що кожен документ чи фрагмент тексту представляється у вигляді вектора у багатовимірному просторі ознак, отриманому завдяки ембедингам. Коли користувач формулює запит, він також перетворюється у вектор, після чого відбувається пошук найближчих сусідів. Цей підхід добре масштабується та дозволяє враховувати семантичну подібність, що робить його значно ефективнішим за традиційний пошук за ключовими словами.

Втім, Dense Retrieval має свої обмеження. По-перше, якість пошуку суттєво залежить від якості ембедингів, які можуть виявитися недостатньо точними у вузькоспеціалізованих предметних областях. По-друге, пошук у великій векторній базі даних потребує значних обчислювальних ресурсів. Саме ці виклики стимулювали появу гібридних підходів.

Hybrid Search поєднує можливості класичного пошуку на основі індексів і щільного пошуку у векторному просторі. Така комбінація дозволяє враховувати як точні збіги ключових термінів, так і семантичну близькість. Наприклад, у правових чи медичних системах важливо зберегти точність термінології, де навіть незначна відмінність у слові може змінити зміст. У цьому випадку гібридні алгоритми дають змогу досягти балансу між суворістю формальних критеріїв і гнучкістю семантичного пошуку.

Ключову роль у реалізації обох підходів відіграють векторні бази даних, що спеціалізуються на зберіганні та ефективному пошуку ембедингів. Найбільш відомими є FAISS від Facebook, Pinecone та Weaviate, які забезпечують високошвидкісний пошук навіть у масивах, що складаються з мільйонів векторів. Архітектурно такі бази часто включають механізми попереднього індексування,

багаторівневі структури для пошуку найближчих сусідів та можливості масштабування у хмарних середовищах. Саме завдяки ним стало можливим практичне застосування RAG на промисловому рівні, адже класичні реляційні чи документні бази не могли забезпечити необхідної продуктивності.

Таким чином, архітектури RAG-систем формують гнучкий інструментарій для інтеграції зовнішніх знань у роботу мовних моделей. Dense Retrieval виступає як ефективна семантична основа, Hybrid Search забезпечує баланс між точністю й узагальненням, а векторні бази даних роблять можливим масштабне зберігання та пошук інформації. Усі ці підходи поступово інтегруються у мультиагентні системи, де різні агенти можуть брати на себе функції пошуку, аналізу та генерації, використовуючи єдину RAG-архітектуру як базовий механізм доступу до знань. Це підтверджує актуальність подальших досліджень у напрямку інтеграції RAG-компонентів у багатокомпонентні інтелектуальні системи. Схематично, концепція Retrieval-Augmented Generation показана на рисунку 1.2.

Рисунок 1.2 – Концепція Retrieval-Augmented Generation

1.3 Методи та алгоритми інтеграції RAG у багатокомпонентні програмні системи

Інтеграція RAG-компонентів у багатокомпонентні програмні системи пов’язана з низкою проблем, серед яких найбільш актуальною є узгодженість відповідей. У великих інформаційних середовищах агенти та модулі часто отримують доступ до різнорідних джерел знань, що можуть містити суперечливі факти або відрізнятися рівнем актуальності. Наприклад, один агент може здійснювати пошук у корпоративній базі документів, тоді як інший – у відкритих інтернет-ресурсах. У такій ситуації навіть за однаковим запитом можна отримати несумісні дані, що без механізму узгодження призводить до конфліктів у прийнятті рішень.

Додатковою складністю є неоднорідність форм представлення знань. Векторні бази даних забезпечують пошук семантично подібних фрагментів, однак не гарантують логічної несуперечності результатів. Відповіді мовних моделей можуть бути стилістично правильними, але містити елементи «галюцинацій», якщо серед вибраних документів відсутній релевантний контекст. Це особливо критично для мультиагентних систем, у яких декілька агентів повинні колективно дійти узгодженого рішення.

У сучасних дослідженнях пропонуються різні підходи для вирішення цієї проблеми. Найбільш перспективними вважаються алгоритми консенсусу, де кілька агентів незалежно формують свої відповіді на основі RAG, після чого здійснюється їхнє порівняння і вибір узгодженого варіанта. Іншим шляхом є введення спеціалізованих агентів-модераторів, що оцінюють відповіді з погляду суперечливості й стабільності, використовуючи додаткові критерії якості. Крім того, дедалі частіше застосовується підхід багатокрокового уточнення, коли генерація не завершується одразу після першого пошуку, а повторюється з урахуванням попередніх відповідей та контексту всієї системи. Це дозволяє підвищити узгодженість, але водночас вимагає додаткових обчислювальних ресурсів та ускладнює архітектуру.

Таким чином, узгодженість відповідей постає ключовою проблемою при інтеграції RAG у складні архітектури, і її розв’язання передбачає комбінацію алгоритмічних підходів, структурного моделювання та впровадження механізмів контролю якості.

Сучасні багатокомпонентні системи зазвичай проектуються у модульній парадигмі, де кожен компонент відповідає за певний набір функцій і взаємодіє з іншими через чітко визначені інтерфейси. Це повною мірою стосується і RAG-компонентів, які можуть бути інтегровані у вигляді окремих служб для пошуку, індексації та генерації. Такий підхід забезпечує високу гнучкість архітектури: компоненти можуть масштабуватися незалежно один від одного, оновлюватися без зупинки всієї системи та налаштовуватися під специфічні потреби домену.

Особливо важливою є побудова розподілених RAG-компонентів, коли база знань і механізми пошуку розподіляються між кількома вузлами. Це дозволяє не лише обробляти великі обсяги даних у режимі реального часу, а й забезпечує відмовостійкість системи. Якщо один вузол стає недоступним, інші продовжують обробку запитів, що критично для високонавантажених систем. Додатковим плюсом є можливість географічного розподілу, коли вузли розташовані ближче до кінцевих користувачів, що знижує затримки при доступі до знань.

З інженерної точки зору, важливим завданням є визначення оптимального рівня гранулярності RAG-компонентів. З одного боку, надто дрібна декомпозиція призводить до надмірних витрат на координацію, з іншого – надто великі модулі знижують гнучкість і повторне використання. У практиці найчастіше застосовується мікросервісний підхід, де RAG-компоненти реалізуються як незалежні сервіси, що взаємодіють через API. Це дозволяє легко інтегрувати їх у мультиагентні системи, де різні агенти можуть звертатися до спільних сервісів пошуку та генерації, не дублюючи функціональність.

Отже, модульність і розподіленість виступають фундаментальними принципами побудови сучасних RAG-систем, які забезпечують масштабованість, гнучкість та відмовостійкість, необхідні для їхнього ефективного застосування у багатокомпонентних архітектурах.

Оцінка ефективності інтеграції RAG у складні програмні системи неможлива без використання формалізованих метрик якості. Найбільш поширеною і водночас ключовою є метрика релевантності, яка визначає, наскільки отримані результати відповідають інформаційним потребам користувача. У контексті RAG релевантність оцінюється як на рівні пошуку (якість вибраних документів), так і на рівні генерації (здатність моделі інтегрувати отриманий контекст у свою відповідь). Висока релевантність забезпечує не лише корисність системи, а й знижує ризик появи помилкових або вигаданих фактів.

Другою важливою характеристикою є стабільність. RAG-системи можуть давати різні відповіді на ідентичні або схожі запити залежно від того, які документи було знайдено під час пошуку, чи навіть від випадковості у виборі токенів під час генерації. Для користувачів це може виглядати як непередбачуваність системи, що підриває довіру до неї. Тому в практиці проектування вводяться спеціальні методи забезпечення стабільності: кешування результатів пошуку, фіксація генеративних параметрів, використання алгоритмів ансамблювання відповідей.

Не менш суттєвим є критерій трасованості. На відміну від класичних мовних моделей, у RAG-системах зберігається можливість посилатися на джерела інформації, що були використані для формування відповіді. Це дозволяє користувачам перевіряти достовірність відповідей і значно підвищує рівень прозорості системи. У корпоративних і правових застосунках трасованість стає фактично обов’язковою вимогою, оскільки без неї неможливо гарантувати юридичну відповідальність за прийняті рішення.

Таким чином, інтеграція RAG-компонентів має супроводжуватися впровадженням системи метрик, яка дозволяє не лише оцінювати ефективність, а й формувати довіру користувачів до системи. Релевантність забезпечує корисність, стабільність гарантує передбачуваність, а трасованість – прозорість і контрольованість. Разом вони формують базу для стандартизації та широкого впровадження RAG у промислових багатокомпонентних системах. Архітектура модульних і розподілених RAG-компонентів показана на рисунку 1.3.

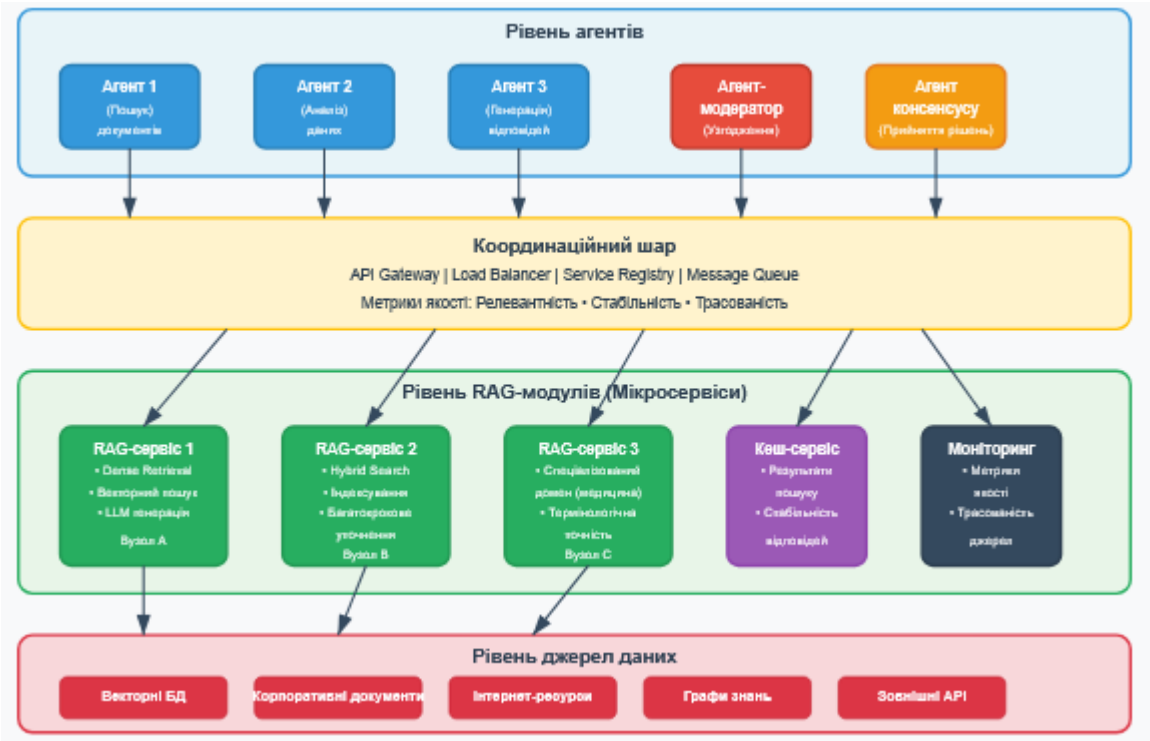

Рисунок 1.3 – Архітектура модульних і розподілених RAG-компонентів

1.4 Практичні аспекти використання RAG у мультиагентних системах

Сучасний розвиток мультиагентних систем (MAS) тісно пов’язаний із впровадженням методів Retrieval-Augmented Generation, оскільки саме вони дозволяють розширювати можливості агентів у роботі з великими обсягами динамічних даних. Якщо перші покоління агентних платформ зосереджувалися передусім на моделюванні кооперативної поведінки та механізмах узгодження дій, то новітні рішення роблять акцент на гнучкій інтеграції з мовними моделями і механізмах зовнішнього пошуку. Серед найбільш відомих мультиагентних платформ останніх років варто відзначити AutoGen, CrewAI, CAMEL та MetaGPT, які заклали основу для інженерії складних AI-систем, орієнтованих на роботу з відкритими та корпоративними знаннями.

Платформа AutoGen була однією з перших, яка надала розробникам інструменти для створення агентів на основі великих мовних моделей, що можуть взаємодіяти один з одним. Вона передбачає модульну архітектуру, де кожен агент має чітко визначені ролі, а комунікація відбувається у вигляді діалогів.

Впровадження RAG у таке середовище дозволяє значно підвищити рівень обґрунтованості рішень, оскільки агенти мають змогу підкріплювати свої аргументи зовнішніми фактами. CrewAI пішла далі у напрямку масштабованості, запропонувавши засоби організації колективів агентів, які можуть координуватися для виконання складних завдань. У такій системі RAG-компоненти часто реалізуються як спільні сервіси доступу до баз знань, що одночасно обслуговують декілька агентів. Це дозволяє уникати дублювання пошуку та підвищує узгодженість результатів. CAMEL, своєю чергою, демонструє приклад орієнтації на діалогові сценарії, де агенти взаємодіють у рамках симульованих ролей. Використання RAG у цьому випадку забезпечує реалістичність діалогів та зменшує ризик помилкових відповідей. MetaGPT є ще одним цікавим прикладом, де мультиагентна система використовується для автоматизації процесів розроблення програмного забезпечення: агенти беруть на себе ролі аналітиків, архітекторів, програмістів і тестувальників. У таких сценаріях RAG стає критично важливим, адже без доступу до актуальної документації та кодових прикладів агенти не змогли б ефективно виконувати свої функції.

Інтеграція зовнішніх знань у мультиагентні системи завжди була складним завданням. У класичних MAS агенти використовували спільні онтології та бази даних для обміну інформацією, однак такі рішення погано масштабувалися й потребували значних зусиль з підтримки. RAG-підхід змінив ситуацію, оскільки дозволив агентам працювати не зі статичними знаннями, а з динамічною інформацією, доступною через механізми пошуку. Завдяки векторним базам даних агенти отримують змогу не лише знаходити документи за ключовими словами, а й орієнтуватися у семантичному просторі, що особливо важливо при роботі з неоднорідними даними. Важливим аспектом є також підтримка трасованості: кожен агент може не тільки сформулювати відповідь, але й вказати джерело, на основі якого вона була отримана. Це суттєво підвищує довіру до системи, особливо у випадках прийняття критично важливих рішень.

Серед практичних прикладів застосування мультиагентних систем із RAG-компонентами варто відзначити кілька ключових напрямів. У бізнес-аналітиці такі

системи дозволяють автоматизувати збір і узагальнення даних з різних джерел – від внутрішніх корпоративних звітів до відкритих ринкових оглядів. Агент, що відповідає за пошук, забезпечує релевантність матеріалів, тоді як агент-аналітик формує висновки, використовуючи генеративні можливості LLM. У сфері підтримки прийняття рішень RAG-компоненти допомагають зменшити ризик суб'єктивних помилок, надаючи користувачам прозорі та перевірювані обґрунтування. Це особливо актуально для фінансових організацій, де неправильне рішення може призвести до значних втрат. Автоматизація документообігу є ще одним показовим прикладом: агенти здатні здійснювати пошук релевантних положень у договорах, створювати резюме документів і навіть формувати проєкти нових угод на основі попередньо знайдених зразків. Усі ці сценарії доводять, що інтеграція RAG у мультиагентні системи не є теоретичним експериментом, а вже має значний вплив на практику.

Таким чином, сучасні мультиагентні платформи, зокрема AutoGen, CrewAI, CAMEL та MetaGPT, демонструють можливість ефективного впровадження RAG у різні прикладні сфери. Особливості інтеграції зовнішніх знань полягають у переході від статичних онтологій до динамічного пошуку, а практичні приклади застосування підтверджують, що такі системи здатні підвищити якість аналізу, підтримку прийняття рішень та ефективність документообігу. Перспективи розвитку полягають у подальшій стандартизації архітектурних підходів і вдосконаленні метрик оцінювання, що дозволить зробити мультиагентні системи з RAG ще більш надійними та універсальними інструментами у різних галузях.

1.5 Проблеми та невирішені питання

Інтеграція підходів Retrieval-Augmented Generation у мультиагентні системи сьогодні розглядається як один із найперспективніших напрямів розвитку штучного інтелекту в інженерії програмного забезпечення. Поєднання можливостей великих мовних моделей із механізмами пошуку зовнішніх знань відкриває нові горизонти для застосування, починаючи від автоматизованої

підтримки бізнес-процесів і завершуючи інтелектуальними помічниками для наукових досліджень. Проте попри значні успіхи у цій галузі, існує низка проблем і невирішених питань, які потребують подальшого дослідження та формалізації. До найбільш актуальних належать недостатня методологічна база для інтеграції RAG у мультиагентні середовища, труднощі із масштабованістю та стабільністю відповідей при оновленні баз знань, а також відсутність зрілих рішень у сфері трасованості та відтворюваності результатів у межах життєвого циклу програмного забезпечення.

Недостатня формалізація методів інтеграції RAG у MAS. Попри те, що RAG-системи вже набули поширення у практичних додатках, методи їхньої інтеграції у мультиагентні архітектури залишаються слабо формалізованими. Більшість існуючих рішень мають характер інженерних прототипів, де основна увага приділяється роботі з конкретними бібліотеками чи платформами, такими як LangChain, LlamaIndex або AutoGen. Хоча ці інструменти забезпечують зручні засоби для створення RAG-компонентів, вони не пропонують цілісної методології інтеграції у системи з багатьма агентами, які взаємодіють у складному інформаційному середовищі.

Основна проблема полягає в тому, що мультиагентні системи мають власні принципи функціонування – автономність, координацію, комунікацію та кооперацію. Вбудовування RAG у таку архітектуру вимагає точного визначення ролей і відповідальності кожного агента, зокрема: чи є пошук зовнішніх знань колективною функцією, чи кожен агент має власний RAG-модуль? Чи повинні агенти узгоджувати джерела знань між собою, чи працювати незалежно? Як здійснюється розподіл навантаження між агентами, коли одночасно виникає багато запитів до бази знань? На ці питання сьогодні немає усталених відповідей.

Відсутність формалізації призводить до фрагментарності рішень: у деяких системах RAG виступає як допоміжний сервіс, у інших – як ядро всієї архітектури. Проте жоден підхід не є достатньо узагальненим, щоб вважатися стандартом або методологією. Це ускладнює повторне використання розробок, їхнє масштабування та адаптацію до нових предметних областей. Отже, проблема браку

формалізованих методів інтеграції RAG у MAS потребує подальшого наукового дослідження й обґрунтування, яке дозволило б перейти від окремих експериментів до стандартизованих інженерних практик.

Проблеми масштабованості та стабільності відповідей при оновленні баз знань. Другим критично важливим питанням є забезпечення масштабованості систем та стабільності їхніх відповідей у процесі роботи з постійно оновлюваними базами знань. RAG-системи мають справу з даними, що швидко змінюються: документи доповнюються новими версіями, векторні індекси оновлюються, джерела інформації з’являються та зникають. У такій ситуації результат пошуку може суттєво відрізнятися навіть для ідентичного запиту, виконаного у різний час.

Для мультиагентних систем це створює особливі труднощі. Якщо декілька агентів одночасно працюють із різними версіями бази знань, виникає проблема узгодженості: їхні висновки можуть суперечити одне одному. Складність полягає в тому, що механізми RAG за своєю природою ймовірнісні й не гарантують детермінованого результату навіть за незмінних даних. У поєднанні з постійними оновленнями це призводить до втрати стабільності відповідей, що знижує довіру користувачів до системи.

Проблема масштабованості проявляється ще й у зростанні обчислювальних витрат. Обробка великих векторних баз даних потребує значних ресурсів, і зі збільшенням кількості агентів навантаження зростає нелінійно. Навіть за використання сучасних індексаційних структур (наприклад, HNSW або IVF-PQ) пошук у надвеликих колекціях залишається дорогим. Для забезпечення реальної масштабованості необхідно розробляти механізми розподіленого пошуку, кешування результатів і балансування навантаження, але й ці рішення не усувають ризику втрати стабільності у випадку асинхронного оновлення даних.

Отже, забезпечення масштабованості та стабільності відповідей є одним із ключових викликів інтеграції RAG у мультиагентні системи. Поки що відсутні зрілі інженерні практики, які дозволили б поєднати ефективність роботи з великими обсягами даних із гарантіями узгодженості та повторюваності відповідей.

Питання трасованості та відтворюваності результатів у життєвому циклі ПЗ. Третім важливим аспектом, який залишається невирішеним, є забезпечення трасованості та відтворюваності результатів у межах життєвого циклу програмного забезпечення. Трасованість у контексті RAG означає здатність системи вказати, які саме документи чи джерела стали підґрунтям для генерації тієї чи іншої відповіді. У класичних інформаційних системах це питання вирішувалося завдяки збереженню посилань на джерела, однак у випадку генеративних моделей ситуація ускладнюється: модель може інтегрувати фрагменти даних у нові формулювання, які важко зіставити з початковим контентом.

Відсутність повної трасованості створює значні ризики, особливо в критичних доменах – юриспруденції, медицині, фінансах, де необхідно мати чітке обґрунтування кожного рішення. Якщо система не здатна пояснити, на яких документах ґрунтується її висновок, вона втрачає довіру й може стати неприйнятною для практичного використання.

Проблема відтворюваності результатів тісно пов’язана із трасованістю. Навіть за наявності однакових даних генеративні моделі можуть давати різні відповіді на ідентичні запити через стохастичну природу вибору токенів. Це створює труднощі для тестування, валідації та супроводу програмних систем. У класичних підходах до інженерії ПЗ відтворюваність результатів є ключовою вимогою до будь-якого модуля, тоді як у випадку RAG-компонентів вона часто порушується.

Для подолання цих проблем пропонуються різні підходи: зниження рівня стохастичності під час генерації, використання детермінованих параметрів, збереження історії пошукових запитів та відповідей, а також впровадження механізмів «audit trail», які дозволяють відстежувати увесь шлях формування відповіді. Проте ці рішення ще далекі від стандартизації й у більшості випадків залишаються на стадії експериментальних прототипів.

Попри значний прогрес у розвитку RAG та мультиагентних систем, існують суттєві проблеми, які обмежують їхнє практичне використання. Недостатня формалізація методів інтеграції не дозволяє створити єдину методологічну основу,

яка забезпечила б повторюваність і сумісність різних рішень. Проблеми масштабованості та стабільності відповідей при оновленні баз знань роблять такі системи вразливими у реальних сценаріях з великими даними. Нарешті, відсутність зрілих рішень у сфері трасованості та відтворюваності результатів створює ризики для застосування у критично важливих доменах. Саме ці невирішені питання визначають актуальність подальших досліджень і формують основу для розробки нових методів, спрямованих на інтеграцію RAG у мультиагентні системи з урахуванням вимог інженерії програмного забезпечення

1.6 Висновки. Постановка задачі

Проблематика інтеграції Retrieval-Augmented Generation у мультиагентні системи є надзвичайно актуальною для сучасної інженерії програмного забезпечення. Це пояснюється стрімким поширенням великих мовних моделей, які демонструють високі можливості у завданнях генерації тексту, проте залишаються обмеженими у роботі з актуальними та специфічними знаннями. Використання механізмів RAG дозволяє долати ці обмеження шляхом підключення зовнішніх баз даних та джерел знань, проте інтеграція таких компонентів у складні мультиагентні архітектури породжує нові проблеми. Зокрема, виникає суперечність між вимогами до стабільності, узгодженості та відтворюваності роботи системи й характером стохастичних моделей, які лежать в основі RAG.

Актуальність дослідження підсилюється й практичною значущістю теми. Багато сучасних сфер діяльності – від бізнес-аналітики до автоматизації документообігу чи підтримки прийняття рішень – потребують інтелектуальних систем, здатних працювати з великими обсягами даних у реальному часі. Проте у наявних реалізаціях мультиагентних платформ інтеграція RAG-компонентів часто залишається інженерним експериментом без належної методологічної бази. Це ускладнює створення масштабованих і надійних рішень, що здатні до супроводу в реальних умовах експлуатації. Таким чином, дослідження, спрямоване на

формалізацію підходів та розробку нового методу інтеграції, є як науково, так і практично значущим.

Об’єктом дослідження є мультиагентні системи програмного забезпечення з вбудованими модулями генерації відповідей на основі зовнішніх джерел знань (RAG-компонентів). Це той рівень, на якому проявляється проблемна ситуація: саме мультиагентні архітектури створюють складне середовище взаємодії, де координація, узгодженість і масштабованість мають критичне значення.

Предметом дослідження є метод і засоби інтеграції RAG-компонентів у мультиагентні системи, що забезпечують цілісну поведінку агентів, релевантність відповідей, узгодженість даних і якісну взаємодію із зовнішніми базами знань. Це більш вузький аспект, який фокусується безпосередньо на проблемних питаннях інтеграції: як організувати роботу агентів із зовнішніми джерелами так, щоб система залишалася стабільною та керованою.

Метою дослідження є розробка методу інтеграції компонентів генерації відповідей з доступом до зовнішніх джерел знань (RAG-компонентів) у мультиагентні системи програмного забезпечення з урахуванням принципів узгодженості, масштабованості та забезпечення якості функціонування програмного продукту. Іншими словами, йдеться про пошук такого інженерного рішення, яке дозволить поєднати гнучкість і потужність RAG з вимогами до життєвого циклу програмного забезпечення.

Для досягнення цієї мети необхідно вирішити ряд невирішених завдань, які й визначають зміст дослідження:

1) сформувати узагальнену методологічну базу для інтеграції RAG у мультиагентні архітектури, що досі відсутня у науковій літературі;

2) подолати проблему масштабованості та забезпечити стабільність відповідей системи при роботі з оновлюваними базами знань;

3) розробити підхід до трасованості та відтворюваності результатів, який дозволить валідувати і супроводжувати мультиагентні системи з RAG-компонентами упродовж їхнього життєвого циклу;

4) визначити критерії оцінки якості інтеграції RAG у MAS, які враховують специфіку програмної інженерії, а не лише продуктивність моделей;

5) спроєктувати і випробувати архітектурну модель, яка забезпечить можливість практичної реалізації запропонованого підходу.

Таким чином, постановка задачі у даному дослідженні полягає у необхідності створення науково обґрунтованого та практично придатного методу інтеграції RAG у мультиагентні системи, здатного забезпечити баланс між гнучкістю мовних моделей і вимогами до надійності та відтворюваності програмних систем.

2 ТЕОРЕТИЧНІ ОСНОВИ МЕТОДУ ІНТЕГРАЦІЇ RAG-КОМПОНЕНТІВ У МУЛЬТИАГЕНТНІ СИСТЕМИ

2.1 Формалізація задачі інтеграції RAG у мультиагентне середовище

Інтеграція зовнішніх джерел знань у мультиагентну програмну систему є складним завданням, яке передбачає поєднання Retrieval-Augmented Generation (RAG) з ієрархією агентів, що виконують розподілені функції. З формальної точки зору, можна розглядати таку систему як множину агентів $A = \{a_1, a_2, ..., a_n\}$, кожен з яких має доступ до підсистеми генерації, до зовнішньої бази знань K або до інших агентів через канали міжагентної комунікації. Мета полягає у тому, щоб забезпечити ефективний процес добування релевантної інформації з K, її аналізу і узгодженого використання у процесах генерації відповідей або прийняття рішень. У загальному випадку взаємодія агентів та знань описується функцією:

$$f : (q_i, K, C_i) \mapsto r_i, \tag{2.1}$$

де q_i – запит агента a_i, K – зовнішнє векторизоване або гібридне сховище знань, C_i – локальний або глобальний контекст, а r_i – результат генерації або проміжний результат reasoning-процесу. Важливо, що f реалізується як композиція пошуку (retrieval) та генерації (generation), з можливими додатковими кроками фільтрації, валідації або координації між агентами.

Однією з ключових складностей є розділення ролей між агентами: агент-пошуку, агент-генератор, агент-верифікатор, агент-контролер. Кожен з них реалізує частину загального ланцюга перетворення знань у відповідь. Наприклад, пошук може бути делегований окремому агенту, який спеціалізується на взаємодії з FAISS або Chroma; генерація – LLM-компоненту; оцінювання релевантності – агенту-критику.

Таким чином, формулювання задачі інтеграції потребує створення уніфікованої моделі комунікації між цими агентами, моделі узгодженого доступу

до знань, а також механізмів контролю якості, які враховують багатовимірний характер відповіді (релевантність, правдивість, повнота, обґрунтованість).

Загальна архітектура інтеграції RAG-компонента в мультиагентну систему зображена на рисунку 2.1, де показано послідовність взаємодії агентів із зовнішнім джерелом знань через спільний пошуковий інтерфейс.

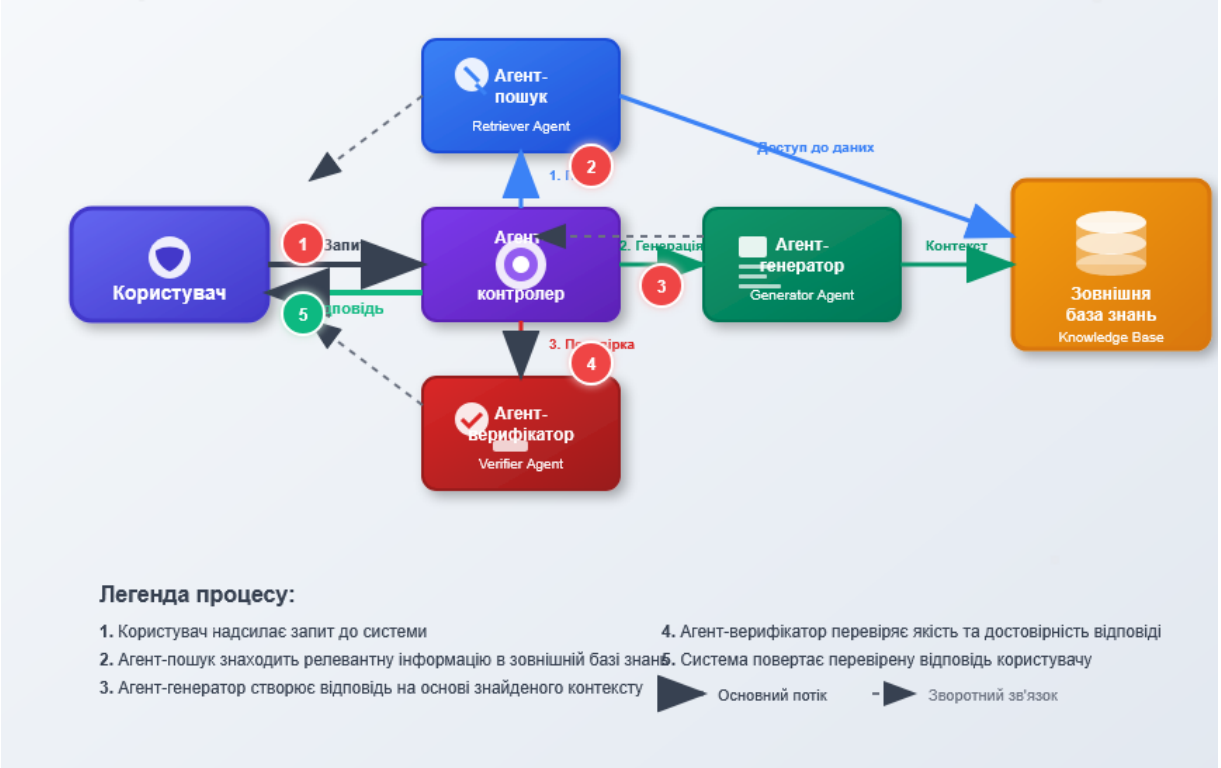

Рисунок 2.1 – Архітектура формалізованої інтеграції RAG у MAS

Однією з ключових складностей є розділення ролей між агентами: агент-пошуку, агент-генератор, агент-верифікатор, агент-контролер. Кожен з них реалізує частину загального ланцюга перетворення знань у відповідь. Наприклад, пошук може бути делегований окремому агенту, який спеціалізується на взаємодії з FAISS або Chroma; генерація – LLM-компоненту; оцінювання релевантності – агенту-критику.

Формулювання задачі інтеграції потребує створення уніфікованої моделі комунікації між цими агентами, моделі узгодженого доступу до знань, а також

механізмів контролю якості, які враховують багатовимірний характер відповіді (релевантність, правдивість, повнота, обґрунтованість).

Розробка методу інтеграції зовнішніх знань у мультиагентну систему потребує чіткого формулювання вимог до такого методу. На основі аналізу сучасних публікацій та практичного досвіду [4], [9], [15], можна виокремити чотири базові вимоги: масштабованість, узгодженість, стабільність та трасованість.

Масштабованість визначається як здатність системи підтримувати ефективність у разі зростання кількості агентів або обсягу зовнішньої бази знань. У практиці це означає, що час відповіді, якість результату та навантаження на ресурси мають зростати сублінійно. Наприклад, при збільшенні кількості запитів у 10 разів система повинна масштабувати обробку шляхом кешування, горизонтального шардінгу бази знань, розподілу навантаження між агентами або навіть динамічного створення агентів-клонів з делегованими задачами.

Узгодженість стосується необхідності підтримувати єдину модель знань серед усіх агентів. Це особливо актуально, якщо агенти працюють з оновлюваною базою знань або кешованими даними. Приклад проблеми – коли один агент витягує фрагмент, який уже є неактуальним, тоді як інший працює з оновленою версією. Методи вирішення: централізований менеджер контексту, CRDT-структури для синхронізації, періодичне оновлення наборів даних в кешах за push або pull-механізмом.

Стабільність вимагає, щоб за однакових вхідних умов результати залишалися інваріантними до випадкових факторів – флуктуацій у векторному пошуку, недетермінізму LLM тощо. Це може бути досягнуто введенням механізмів контролю версій бази знань, фіксацією seeds для LLM, використанням енсейблінгу моделей або стабілізацією через voting кількох агентів.

Трасованість є критичною у ситуаціях, коли потрібно пояснити походження відповіді. Це передбачає збереження шляху: запит → знайдені документи → вибрані фрагменти → промпт → відповідь. Лише при наявності цього шляху можна провести аудит, аналіз помилок, відповідати на регуляторні запити або генерувати human-in-the-loop feedback.

Формально, можна описати метод інтеграції як множину функцій і політик:

$$M=\{T_{retrieve},T_{generate},T_{validate},P_{sync},P_{trace}\}, \quad (2.2)$$

де T– трансформації, P – політики. Цей формалізм дозволяє експліцитно вказати, які саме модулі відповідають за виконання кожної з вимог.

Щоб інтеграційний метод був застосовним і відтворюваним, необхідно чітко визначити припущення, на яких він базується, та обмеження, що впливають на його використання. Такий аналіз не тільки знижує ризики під час впровадження, але й окреслює простір для подальших досліджень.

Серед ключових припущень:

1) якість зовнішніх знань – передбачається, що база знань (наприклад, індекси Chroma або FAISS) вже попередньо очищена, нормалізована та подрібнена на чанки зі збереженням семантичного змісту;

2) протоколи взаємодії між агентами – вважається, що агенти дотримуються загальних контрактів API або форматів обміну (наприклад, JSON-схем), що дозволяє забезпечити сумісність без потреби в глибокому семантичному узгодженні;

3) наявність механізму моніторингу – припускається, що всі транзакції з базою знань логуються або маркуються ID, що уможливлює трасування.

Серед обмежень, характерних для більшості реалізацій:

1) висока вартість динамічного оновлення бази знань: у практиці часто спостерігається ситуація, коли оновлення векторного сховища є операцією, несумісною з режимом реального часу, що знижує адаптивність системи;

2) обмеження на довжину запитів та промптів: архітектури LLM мають обмеження контекстного вікна, що зумовлює необхідність обмеження кількості витягнутих фрагментів або агрегації знань;

3) недостатня обґрунтованість довіри до зовнішніх знань: не завжди можна оцінити якість витягнутих фрагментів, особливо якщо використовуються агреговані бази з відкритих джерел.

Чітке формулювання припущень і обмежень дозволяє забезпечити передбачувану поведінку системи, зменшити кількість критичних помилок та оптимізувати механізми fallback у разі невдачі пошуку або генерації. Подальші етапи розробки передбачають урахування цих факторів під час проектування архітектури та вибору технологічних рішень.

2.2 Архітектурна модель мультиагентної системи з інтегрованими RAG-компонентами

У мультиагентних системах (MAS), що функціонують у динамічному інформаційному середовищі, дедалі більшої важливості набуває здатність агентів працювати з актуальними зовнішніми джерелами знань. У цьому контексті архітектурна інтеграція механізмів Retrieval-Augmented Generation (RAG) відкриває нові можливості для адаптивності та інтелектуальності поведінки агентів. Пропонована архітектурна модель включає чіткий поділ функцій між агентами різного типу, введення інтерфейсного рівня для взаємодії з базами знань та забезпечення відповідності принципам масштабованості, трасованості та стабільності відповідей.

У запропонованій моделі передбачено чотири основні типи агентів, кожен з яких виконує окрему роль у ланцюжку генерації відповідей:

1) агент витягування (Retriever Agent) відповідає за ініціалізацію запиту до векторного сховища знань або пошукової системи, отримуючи релевантні фрагменти тексту;

2) агент генерації (Generator Agent) приймає контекст із результатів витягування та формулює відповідь за допомогою великої мовної моделі;

3) агент координації (Coordinator Agent) виступає як керуюча ланка, яка визначає послідовність взаємодії інших агентів та відповідає за цілісність процесу;

4) агент валідації (Validator Agent) перевіряє відповідь на відповідність вимогам точності, повноти, а також може запитувати додаткові пояснення або підкріплення відповідей джерелами.

Узагальнена схема їх взаємодії представлена на рисунку 2.2, де зображено інформаційні потоки між агентами та зовнішніми джерелами знань. Такий розподіл ролей забезпечує функціональну декомпозицію системи, підвищує її гнучкість та дозволяє масштабувати окремі компоненти незалежно.

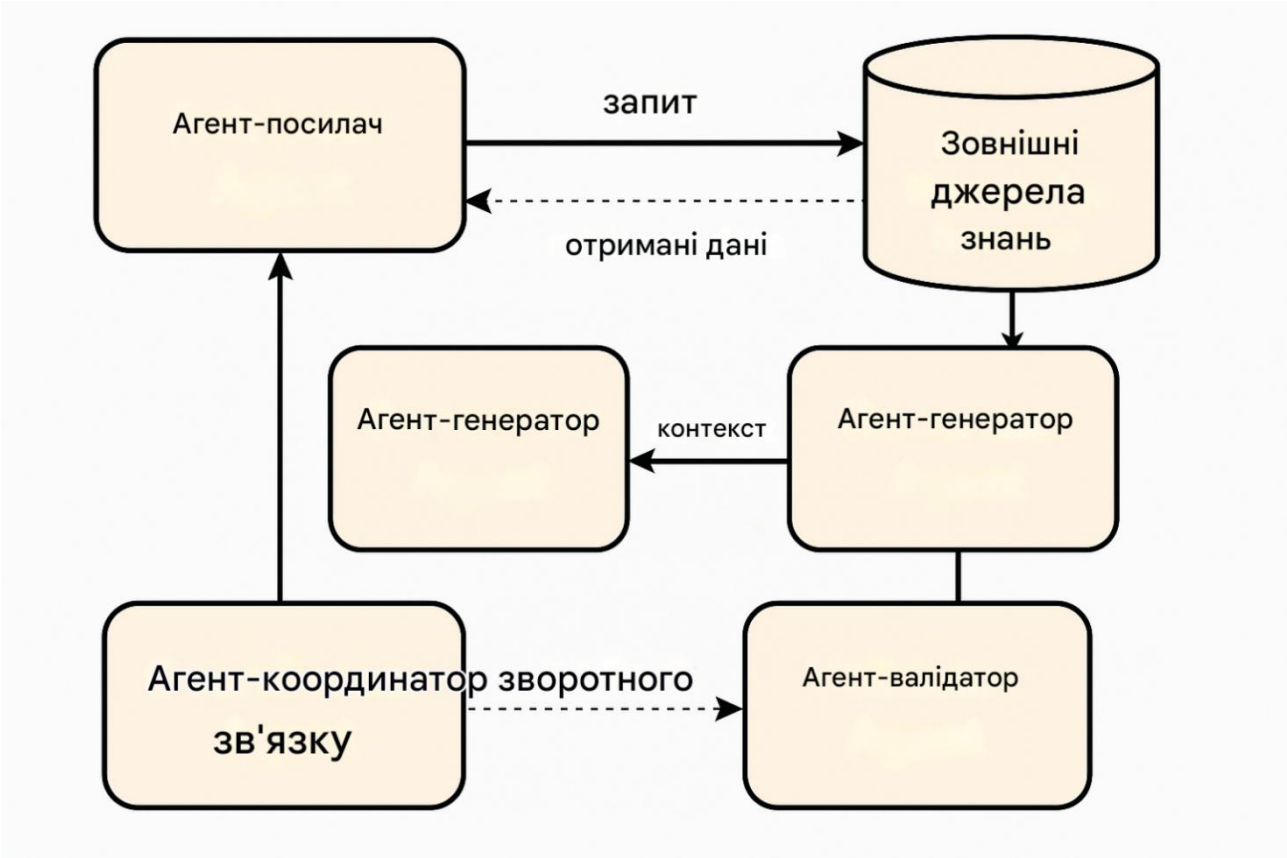

Рисунок 2.2 – Схема взаємодії агентів у системі з RAG-компонентами

2.2.1 Архітектурна модель багатоагентної RAG-системи з інтерфейсним рівнем

2.2.1.1 Концептуальні основи багатоагентного підходу в RAG-системах

Сучасний розвиток систем генерації з доповненим пошуком (Retrieval-Augmented Generation) вимагає переходу від монолітних архітектур до більш складних багатоагентних рішень. Запропонована архітектурна модель базується на принципі спеціалізації компонентів, де кожен агент відповідає за конкретний

аспект обробки запитів, що забезпечує підвищення загальної ефективності системи та спрощує процеси підтримки й масштабування.

Верхній рівень архітектури представлено чотирма спеціалізованими агентами. Retrieval Agent забезпечує семантичний пошук релевантних документів через векторну індексацію та оцінку релевантності контенту. Generation Agent відповідає за інтеграцію з великими мовними моделями, контекстну генерацію відповідей та оптимізацію промптів. Coordination Agent виконує функції оркестрації процесів, управління потоком операцій та балансування навантаження між компонентами. Validation Agent забезпечує перевірку достовірності згенерованих відповідей, фільтрацію нерелевантного контенту та оцінку якості фінальних результатів.

2.2.1.2 Інтерфейсний рівень як центральний компонент абстракції

Ключовою інновацією запропонованої архітектури є впровадження інтерфейсного рівня, який функціонує як централізований RAG Connector. Цей рівень реалізує концепцію "розумного посередника" між агентами та базовими RAG-компонентами, забезпечуючи абстрагування від конкретних реалізацій векторних сховищ та алгоритмів пошуку.

Interface Layer інкапсулює критично важливі компоненти системи, включаючи API Gateway як єдину точку входу для всіх запитів, Message Router для інтелектуальної маршрутизації повідомлень, Load Balancer для розподілу навантаження, а також підсистеми аутентифікації, моніторингу та централізованого логування. Така архітектура забезпечує високий рівень спостережуваності системи та дозволяє ефективно діагностувати проблеми продуктивності.

RAG Connector, як основний компонент інтерфейсного рівня, виконує функції попередньої обробки запитів через нормалізацію, валідацію вхідних даних та трансформацію текстових запитів у векторні представлення. Компонент забезпечує універсальну інтеграцію з різноманітними векторними сховищами,

включаючи FAISS для високопродуктивних обчислень, Chroma для швидкого прототипування, Milvus для корпоративних рішень, а також ElasticSearch для гібридного пошуку. Результати пошуку адаптуються через стандартизацію форматів відповідей, збагачення метаданими та підготовку оптимізованого контексту для генерації відповідей.

2.2.1.3 Базові компоненти системи та їх взаємодія

Нижній рівень архітектури складається з двох основних компонентів. Knowledge Base представляє векторне сховище, що містить структуровані векторні представлення знань, метадані для ефективного пошуку та підтримку множинних форматів даних від текстових документів до структурованих баз знань. RAG Module включає Retriever для швидкого приблизного пошуку найближчих сусідів, Generator як стандартизований інтерфейс до мовних моделей та Context Manager для управління та оптимізації контексту.

Система доповнюється спеціалізованими RAG Components, які включають Embedding Models і Tokenizers для векторизації тексту, Vector Stores і Document Loaders для зберігання та завантаження даних, LLM Connectors і Response Formatters для інтерфейсу та форматування, а також Quality Assessors і Cache Managers для оцінки якості та оптимізації продуктивності через кешування.

2.2.2 Архітектурні переваги та технічні характеристики

Запропонована архітектура забезпечує асинхронну взаємодію через паралельну обробку запитів декількома агентами одночасно та неблокуючі операції, що усуває простої під час очікування відповідей від зовнішніх сервісів. Масштабованість системи досягається через горизонтальне масштабування з можливістю легкого додавання нових екземплярів агентів та мікросервісну архітектуру, що дозволяє незалежне розгортання та оновлення окремих компонентів.

Надійність системи забезпечується через реалізацію fault tolerance механізмів, що забезпечують стійкість до відмов окремих компонентів, системи резервного копіювання для захисту від втрати критичних даних, а також комплексні механізми аутентифікації та авторизації для контролю доступу до ресурсів. Система моніторингу включає збір метрик продуктивності для відстеження ключових показників, детальне логування та трейсинг для діагностики проблем, а також адаптивну оптимізацію з автоматичним налаштуванням параметрів системи.

Впровадження Interface Layer як центрального RAG Connector надає системі винятковий рівень гнучкості та підтримуваності. Абстрагування від конкретних реалізацій дозволяє агентам працювати через уніфікований API, що спрощує розробку та підтримку коду. Легка заміна компонентів забезпечується можливістю оновлення векторних сховищ та алгоритмів пошуку без необхідності модифікації логіки агентів.

Централізоване управління через Interface Layer концентрує всі налаштування та конфігурації в єдиному компоненті, що значно спрощує адміністрування системи. Процеси тестування оптимізуються через можливість створення мок-об'єктів Interface Layer для юніт-тестування окремих агентів без залежності від зовнішніх сервісів.

Така архітектурна модель представляє продвинуту multi-agent систему нового покоління, здатну ефективно адаптуватися до змінних вимог бізнес-процесів та забезпечувати високі показники продуктивності в умовах зростаючих обсягів даних та складності запитів користувачів.

Архітектура з введеним інтерфейсним рівнем представлена на рисунку 2.3.

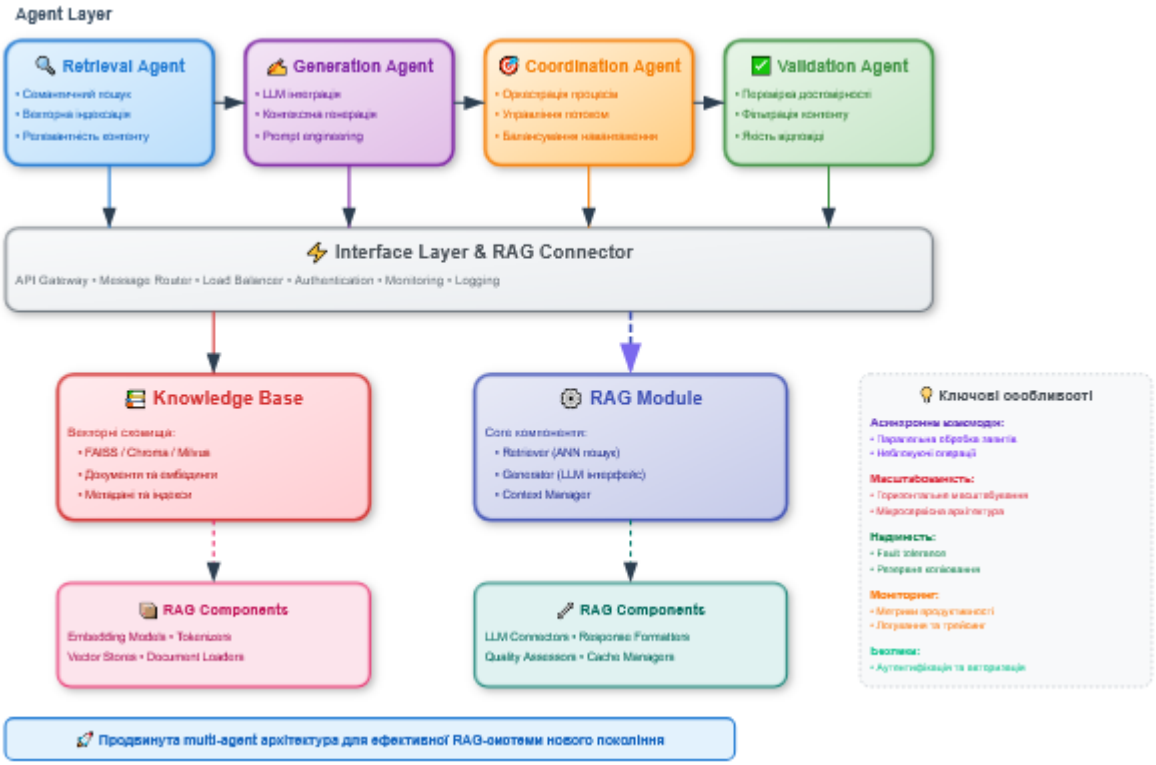

Рисунок 2.3 – Архітектура з інтерфейсним рівнем RAG Connector

Окрім технічної адаптації, інтерфейсний рівень також може бути використаний для реалізації контролю доступу, логування звернень до бази знань, а також для трасування походження кожної відповіді, що є критичним для перевірки якості системи.

2.2.3 Аналіз відмінностей від існуючих архітектур MAS

Традиційні мультиагентні архітектури (наприклад, Belief-Desire-Intention – BDI, або Actor-based) передбачають наявність внутрішньої бази знань у кожного агента, а адаптація до нових знань відбувається через навчання або ручне оновлення. У запропонованій моделі це обмеження знято: зовнішні джерела знань динамічно підключаються через RAG-механізм, що надає агентам здатність до знання на вимогу.

Ключова відмінність полягає у наступному:

1) джерело знань: у класичних MAS – локальне, статичне, у RAG-MAS – динамічне, зовнішнє;

2) адаптація до контексту: у класичних – обмежена, часто потребує перенавчання, у RAG-підході – здійснюється через оновлення бази знань без зміни агента;

3) трасованість: у RAG-підході можлива побудова explainable-пояснень шляхом повернення фрагментів знань, що вплинули на відповідь;

4) масштабованість: моделі, що використовують інтерфейсний рівень, легше масштабуються у хмарному середовищі, оскільки не зберігають повні бази знань локально.

Таким чином, запропонована архітектура поєднує сильні сторони MAS (декомпозиція, автономність, координація) із перевагами сучасних LLM-RAG систем (гнучкість, адаптивність, знання на вимогу), формуючи основу для створення мультиагентних рішень нового покоління.

2.3 Математична та об’єктно-орієнтована модель інтеграції

2.3.1 Формалізація процесів витягування та генерації знань

У рамках архітектури типу RAG (Retrieval-Augmented Generation) взаємодія між підсистемами витягування знань та генерації тексту має бути описана як послідовність перетворень над вхідними даними, з формальним урахуванням залежностей між етапами. Формально, нехай $x \in X$ – вхідний запит користувача, тоді процес витягування знань можна представити як функцію:

$$R : X \rightarrow C, \tag{2.3}$$

де $C = \{c_1, c_2, \ldots, c_k\}$ – множина релевантних контекстних фрагментів, що витягнуті з бази знань K. Надалі ці фрагменти передаються до модуля генерації, що реалізує функцію:

$$G : X \times C \rightarrow Y, \tag{2.4}$$

де Y – згенерована відповідь. Повна функціональна композиція RAG-системи описується як:

$$F = G \circ R : X \rightarrow Y, \text{ тобто } F(x) = G(x, R(x)). \tag{2.5}$$

Для мультиагентного середовища функції R і G можуть бути реалізовані різними агентами, які працюють асинхронно. Це потребує координації, що забезпечується агентом-координатором через оператор узгодження:

$$C' = U(C), \text{ де } U: C \rightarrow C'. \tag{2.6}$$

Для моделювання взаємодії між агентами введемо множину агентів:

$$A = \{A\_R, A\_G, A\_C, A\_V\}, \tag{2.7}$$

де A_R – агент витягування, A_G – агент генерації, A_C – агент координації, A_V – агент валідації.

Функція передачі повідомлень між агентами визначається через граф взаємодії

$$G = (A, E), \tag{2.8}$$

де $E \subseteq A \times A$ – множина напрямлених зв’язків. Кожен агент $A_i \in A$ виконує певну функцію

$$f_i : D_i \rightarrow R_i. \tag{2.9}$$

Для доступу до бази знань вводиться функція

$$K : Q \rightarrow C. \quad (2.10)$$

Оператор трасованості визначається як:

$$T : Y \rightarrow C. \quad (2.11)$$

2.3.2 Об’єктно-орієнтоване моделювання класів та інтерфейсів для інтеграції

Для програмної реалізації системи Retrieval-Augmented Generation доцільно застосувати об'єктно-орієнтований підхід з використанням агентної архітектури. В основі такої архітектури лежить абстрактний клас Agent, який визначає базову функціональність усіх агентів системи та включає фундаментальні методи receive(), process() та send() для обробки та передачі повідомлень.

Спеціалізовані агенти наслідують базову функціональність та розширюють її відповідно до своїх ролей у системі. RetrieverAgent відповідає за пошук релевантної інформації у векторних базах даних за допомогою методу retrieve(query), забезпечуючи ефективне знаходження контекстуально відповідних документів. GeneratorAgent здійснює генерацію відповідей на основі отриманих запитів та контекстної інформації через метод generate(query, context), інтегруючи знайдені дані з можливостями великих мовних моделей.

CoordinatorAgent виконує функції центрального координатора, керуючи взаємодією між різними компонентами системи та забезпечуючи правильну послідовність операцій. ValidatorAgent здійснює контроль якості генерованих результатів, перевіряючи відповідність відповідей заданим критеріям достовірності та релевантності.

Для забезпечення гнучкості підключення до різноманітних джерел знань використовується інтерфейс RAGConnector, який стандартизує взаємодію з векторними базами даних та іншими репозиторіями інформації. Цей інтерфейс

дозволяє легко інтегрувати нові типи сховищ даних без модифікації основної логіки системи.

Міжагентна комунікація реалізується через застосування патерну "Спостерігач" або системи черг повідомлень, таких як Redis чи ZeroMQ, що забезпечує асинхронну та масштабовану взаємодію між компонентами системи. Такий архітектурний підхід гарантує високу модульність, розширюваність та підтримуваність системи RAG.

UML-діаграма класів агентної архітектури RAG представлена на рисунку 2.4.

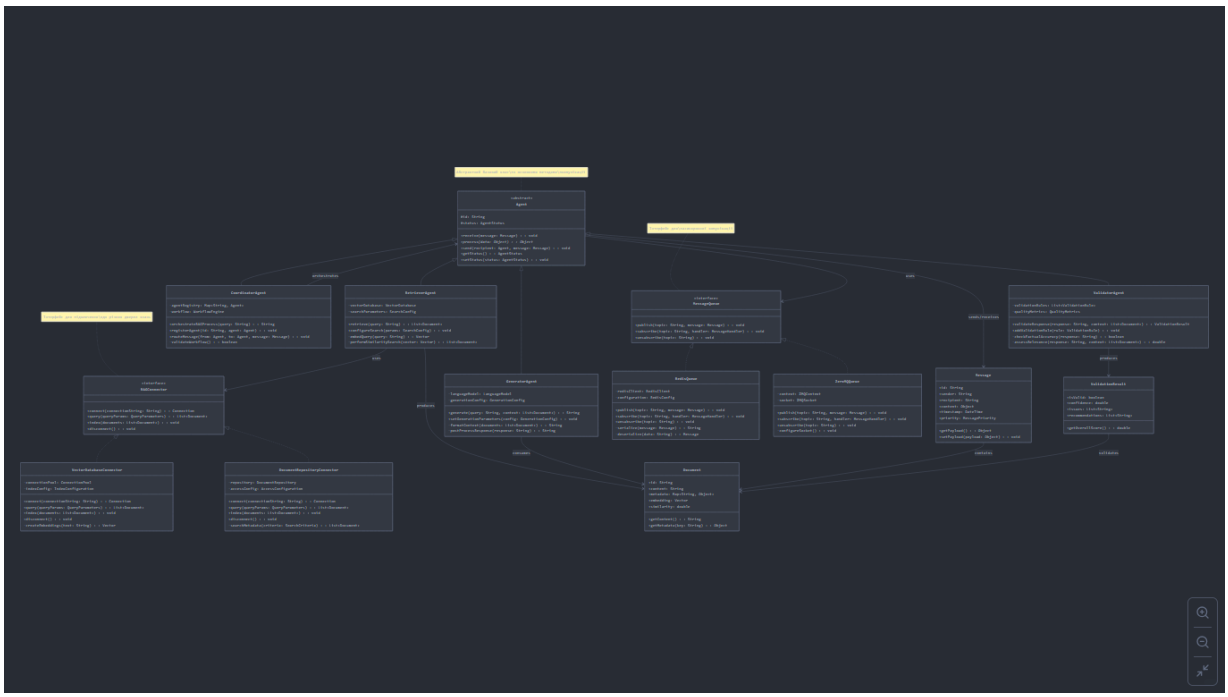

Рисунок 2.4 – UML-діаграма класів агентної архітектури RAG

2.4 Метод інтеграції RAG-компонентів: опис та властивості

2.4.1 Принципи роботи запропонованого методу

Запропонований метод інтеграції Retrieval-Augmented Generation (RAG) у мультиагентне середовище базується на поєднанні принципів модульності, прозорої маршрутизації запитів та динамічної адаптації до змін джерел знань. Основна ідея полягає в розділенні відповідальності між спеціалізованими агентами, де кожен компонент системи (витягування, генерація, фільтрація, валідація) виконується окремим агентом або групою агентів з чітко визначеним

інтерфейсом. Усі ці компоненти об'єднуються через рівень координації, який забезпечує маршрутизацію запитів, агрегацію результатів і гарантує логічну цілісність відповіді.

На відміну від монолітних інтеграцій, метод підтримує логіку “від запиту до результату” як послідовність незалежних операцій, реалізованих через проміжні повідомлення між агентами. Це дозволяє легко підміняти або оновлювати компоненти (наприклад, змінювати FAISS на Chroma, або замінювати GPT на іншу LLM), не порушуючи загальної архітектури. Таким чином, метод зберігає сумісність з принципами Service-Oriented Architecture (SOA) та Actor Model.

У центрі методу – проміжне подання запиту у вигляді об’єкта типу QueryTask, який містить уніфіковане представлення запиту, критерії релевантності, ідентифікатор контексту, а також параметри генерації. Це дозволяє агентам діяти над єдиною структурою, зменшуючи неоднозначність і забезпечуючи простежуваність результатів.

2.4.2 Механізми координації агентів при доступі до зовнішніх джерел знань

Одним із найважливіших аспектів запропонованого методу є координація між агентами, які здійснюють доступ до зовнішніх джерел знань. Оскільки бази знань можуть змінюватися, бути частково реплікованими або знаходитися в різних точках мережі, потрібні спеціальні протоколи для запобігання конфліктам, забезпечення узгодженості та уникнення перевантаження системи.

Координація реалізується через централізований або децентралізований диспетчер (наприклад, CoordinatorAgent), який аналізує вхідний запит і розподіляє підзапити серед агентів-добувачів (RetrieverAgents). У випадку гібридного пошуку диспетчер може ініціювати паралельний доступ до векторного та символічного індексу, а потім об'єднати результати з урахуванням ваг або довіри до джерел.

Щоб гарантувати узгодженість відповідей, використовуються часові мітки та версіювання знань. Кожен запит до бази знань фіксується в журналі доступу (knowledge access log), що дозволяє при потребі відтворити хід reasoning-процесу.

Це важливо для задач з високими вимогами до трасованості, таких як медична або юридична документація.

Також передбачено протокол регулювання навантаження, коли в умовах обмежених ресурсів запити ставляться в чергу або переадресовуються агентам з вільною обчислювальною потужністю. У разі виникнення конфліктних або неоднозначних даних ініціюється голосування агентів-верифікаторів, що забезпечує консенсус без необхідності ручного втручання.

2.4.3 Очікувані властивості методу (адаптивність, модульність, повторне використання)

Запропонований метод інтеграції RAG-компонентів у мультиагентне середовище має низку очікуваних властивостей, які визначають його ефективність і практичну застосовність у реальних системах.

Передусім, адаптивність проявляється у здатності системи підлаштовуватись до зміни структури або вмісту зовнішніх джерел знань. Наприклад, у разі оновлення індексу знань, агенти можуть автоматично змінювати стратегії пошуку, перемикаючись із точного витягування на семантичне або навпаки. Адаптивність також проявляється в самонавчанні агентів – вони можуть аналізувати історію запитів і змінювати ваги внутрішніх критеріїв оцінки.

Модульність досягається завдяки незалежності агентів, які виконують окремі функції. Будь-який з них може бути замінений без необхідності переробляти всю систему. Наприклад, зміна способу генерації (перехід з GPT на Claude або Mistral) потребує лише заміни відповідного агента, без зміни протоколу його виклику.

Повторне використання забезпечується створенням уніфікованих класів і протоколів взаємодії між агентами. Такі елементи, як QueryTask, KnowledgeResult, CoordinationProtocol можуть бути використані повторно в інших проєктах, навіть у тих, які не пов’язані безпосередньо з RAG, але мають аналогічну структуру обробки запитів.

На завершення, метод інтеграції передбачає гнучкість у розгортанні: його можна реалізувати як у централізованому хмарному середовищі, так і у розподіленій системі з агентами, розгорнутими на edge-пристроях, що робить його придатним для широкого спектра застосувань – від чат-ботів до автономних рішень для бізнес-аналітики або підтримки рішень в урядових системах. Рисунок 2.5 ілюструє механізм координації між агентами, задіяними у методі інтеграції RAG.

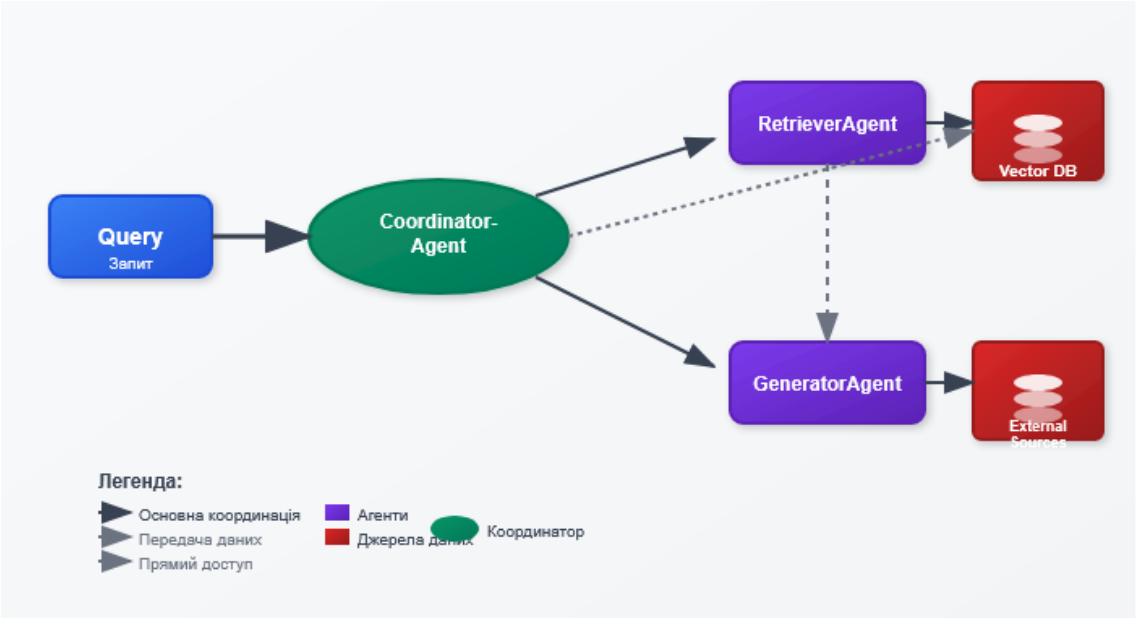

Рисунок 2.5 – Схема координації агентів у методі інтеграції RAG

2.5 Показники та критерії оцінювання ефективності інтеграції

2.5.1 Метрики релевантності та стабільності відповідей

Фундаментальним завданням при оцінюванні ефективності інтеграції RAG-компонентів у мультиагентну систему є кількісне визначення релевантності відповідей, згенерованих агентами на основі зовнішніх джерел знань. Релевантність виражає ступінь семантичної та контекстуальної відповідності згенерованої відповіді поставленому запиту або задачі, що потребує адаптації класичних метрик інформаційного пошуку до специфіки RAG-систем.

Стандартні метрики, запозичені з інформаційного пошуку, можуть бути адаптовані до контексту RAG-систем, зокрема:

1) Precision@k – частка релевантних результатів серед перших k витягнутих фрагментів;

2) Recall@k – частка всіх можливих релевантних фрагментів, які були знайдені серед k витягнутих;

3) Mean Reciprocal Rank (MRR) – обернене середнє ранжування першої релевантної відповіді;

4) F1-score – гармонічне середнє між precision та recall.

Для стабільності (англ. stability) відповідей використовуються метрики, що вимірюють варіації результатів при незначній зміні контексту, запиту або знань:

1) Δ-variation score – абсолютна зміна відповіді при зміні контексту:

$$\Delta_r = \frac{1}{n}\sum_{i=1}^{n} \mathrm{sim}(r_i, r_i') \to m, \tag{2.12}$$

де r_i – початкова відповідь, r_i' – відповідь після зміни, *sim* – функція подібності (наприклад, cosine similarity);

2) Entropy of response distribution – ентропійна міра від розподілу типових відповідей на близькі запити.

Ці метрики дозволяють виявити чутливість інтеграційної системи до незначних змін і забезпечити надійність у промисловому середовищі.

2.5.2 Показники узгодженості та трасованості результатів

Узгодженість (англ. consistency) у мультиагентних системах означає, що всі агенти, які взаємодіють із єдиним джерелом знань, формують відповіді, що не суперечать одна одній і спираються на однакові підстави. Для її оцінки пропонуються:

1) Semantic Agreement Index (SAI) – міра семантичної узгодженості відповідей кількох агентів на один і той самий запит:

$$SAI = \frac{2}{n(n-1)} \sum_{i<j} \mathrm{sim}(r_i, r_j); \quad (2.13)$$

2) Inter-agent Divergence – міра відмінності між поясненнями, які дають агенти, що обґрунтовують свої відповіді з однакових даних.

Трасованість (англ. traceability) – здатність простежити, які джерела і які етапи reasoning-пропуску були використані для побудови відповіді. Цей показник вимагає:

1) підтримки логів звернення до знань;

2) формального маркування проміжних станів reasoning;

3) виводу пояснення (explanation path), що дозволяє інтерпретувати рішення.

Кількісно трасованість можна описати за допомогою Trace Completeness Ratio (TCR) – частки відповідей, для яких можливо побудувати повний ланцюг аргументації від запиту до відповіді.

2.5.3 Методика кількісної та якісної оцінки запропонованого рішення

Для комплексного оцінювання запропонованого методу інтеграції використовується гібридна методика, яка поєднує автоматичні тести з експертною оцінкою. Основні етапи:

а) побудова контрольного набору тестових запитів до системи з наперед відомими правильними відповідями;

б) автоматичне вимірювання метрик:

1) релевантність (Precision@k, F1);

2) Стабільність (Δ-variation, entropy);

3) Узгодженість (SAI, divergence);

4) трасованість (TCR, explanation depth);

в) оцінка експертами:

1) смислова коректність (semantic correctness);

2) аргументованість (justification);

3) відповідність стилю чи домену (domain adherence);

г) агрегація показників у зважену ефективність інтеграції:

$$F = \alpha_i R + \alpha_2 S + \alpha_3 \mathrm{C} + \alpha_4 T, \quad (2.14)$$

де R, S, C, T – релевантність, стабільність, узгодженість, трасованість відповідно; α_i – вагові коефіцієнти, встановлені залежно від пріоритетів системи;

д) порівняння з базовими методами інтеграції (наприклад, просте API-звернення до RAG без координації агентів).

2.6 Порівняння запропонованого підходу з відомими методами

2.6.1 Аналіз відмінностей у структурі та властивостях

Запропонований підхід інтеграції RAG-компонентів у мультиагентну систему відрізняється від відомих фреймворків (LangChain Agents, AutoGen, CrewAI) насамперед архітектурною гнучкістю та орієнтацією на стабільність, узгодженість та трасованість відповідей. У той час як LangChain і AutoGen надають потужні механізми для побудови ланцюжків думок (chains of thought) або координації між агентами за допомогою правил, наш метод фокусується на модульній організації з явною підтримкою таких властивостей:

1) формалізована система оцінки якості інтеграції: релевантність, стабільність, узгодженість, трасованість;

2) внутрішні журнали reasoning-процесу для забезпечення трасованості;

3) розмежування функцій агента-дослідника, агента-селектора і агента-пояснювача, що дозволяє локалізувати джерела помилок і проводити аналіз чутливості;

4) гібридна методика оцінки (метрики + експертне оцінювання), чого бракує в більшості стандартних бібліотек.

2.6.2 Очікувані переваги відносно існуючих реалізацій

Запропонований метод забезпечує фундаментальні покращення у сфері надійності через впровадження комплексної системи моніторингу стабільності відповідей. На відміну від існуючих рішень, таких як LangChain Agents, які не надають інструментів для контролю варіативності результатів, розроблений підхід включає метрики Δ-variation та ентропійного аналізу, що дозволяють кількісно оцінювати стабільність системи при незначних змінах контексту або джерел знань.

Ця особливість набуває критичного значення у промислових застосуваннях, де непередбачувані зміни в відповідях можуть призвести до серйозних наслідків. Існуючі платформи, включаючи AutoGen та CrewAI, зосереджуються переважно на функціональних аспектах мультиагентної взаємодії, залишаючи поза увагою питання операційної стабільності та відтворюваності результатів.

Запропонована архітектура реалізує повноцінну систему трасованості через Trace Completeness Ratio та детальні шляхи пояснень, що кардинально відрізняє її від конкурентних рішень. Тоді як LangChain забезпечує лише мінімальну трасованість, а AutoGen та CrewAI пропонують часткові рішення, розроблений метод гарантує можливість повного відстеження кожного етапу прийняття рішення від початкового запиту до фінальної відповіді.

Така детальна трасованість не лише підвищує довіру користувачів до системи, але й забезпечує можливість аудиту та відладки в складних сценаріях використання. Особливо це важливо у регульованих галузях, таких як фінанси, медицина та юриспруденція, де необхідно надавати обґрунтування кожного автоматизованого рішення.

Впровадження Semantic Agreement Index для явної оцінки узгодженості між агентами представляє унікальну характеристику запропонованого методу. Існуючі рішення або повністю ігнорують питання узгодженості (LangChain), або забезпечують лише часткові механізми контролю (AutoGen, CrewAI), тоді як

розроблений підхід надає кількісні інструменти для моніторингу та підтримання семантичної відповідності між різними агентами системи.

Це особливо критично у сценаріях, де кілька агентів працюють з однаковими джерелами знань, але можуть формувати різні інтерпретації або висновки. SAI дозволяє виявляти такі розбіжності на ранніх стадіях та впроваджувати корективні заходи для підтримання цілісності системи.

Гібридна методика оцінювання, що поєднує автоматичні метрики з експертним аналізом, забезпечує всебічну оцінку ефективності системи, чого не пропонує жодне з існуючих рішень у повному обсязі. Тоді як CrewAI частково підтримує мета-оцінку, а LangChain та AutoGen взагалі не мають такої функціональності, запропонований метод інтегрує системний підхід до оцінювання через зважену комбінацію показників релевантності, стабільності, узгодженості та трасованості.

В таблиці 2.1 наведено характеристики існуючих рішень в порівнянні з запропонованим методом.

Таблиця 2.1 – Порівняльний аналіз архітектурних рішень

Критерій	LangChain Agents	AutoGen	CrewAI	Запропонований метод
Гнучкість у визначенні ролей агентів	Середня	Висока	Висока	Висока
Підтримка reasoning logs	Обмежена	Частково	Частково	Повна
Оцінка стабільності	Відсутня	Відсутня	Відсутня	Присутня (Δ-variation, entropy)
Трасованість	Мінімальна	Часткова	Часткова	Повна (TCR, explanation path)
Узгодженість між агентами	Не контролюється	Частково	Частково	Явна оцінка через SAI
Орієнтація на мета-оцінку	Відсутня	Відсутня	Частково	Системна (гібридна)

Очікувано, розроблений підхід демонструє кращі результати у задачах, де критично важливі:

1) відтворюваність відповідей при зміні джерел знань;

2) інтерпретованість агентних рішень;

3) високий рівень надійності в реальному середовищі, де помилки є неприпустимими.

2.6.3 Визначення меж застосовності розробленого методу

Розроблений метод інтеграції RAG-компонентів у мультиагентну архітектуру, попри свої суттєві переваги, характеризується специфічними обмеженнями, що визначають межі його ефективного застосування. Ці обмеження випливають з фундаментальних принципів функціонування системи та вимагають ретельного аналізу при прийнятті рішень щодо впровадження.

Запропонований підхід передбачає значно вищу складність початкової конфігурації порівняно з традиційними RAG-реалізаціями. Метод вимагає детальної структуризації агентної екосистеми з чітким розмежуванням функціональних ролей, встановленням формалізованих протоколів міжагентної комунікації та налаштуванням багаторівневих механізмів зворотного зв'язку. Такий рівень архітектурної деталізації потребує залучення висококваліфікованих спеціалістів та може подовжити фазу початкового розгортання системи від кількох тижнів до декількох місяців залежно від складності предметної області.

Додатковим фактором складності є необхідність калібрування вагових коефіцієнтів у метриках оцінювання та налаштування порогових значень для детекції аномалій у поведінці агентів. Цей процес вимагає глибокого розуміння специфіки конкретного застосування та може потребувати кількох ітерацій для досягнення оптимальних результатів.

Комплексна система моніторингу та логування, що забезпечує високий рівень трасованості та контролю якості, призводить до суттєвого збільшення обчислювального навантаження. Ведення детальних reasoning-логів, обчислення

метрик стабільності в реальному часі та підтримка повної трасованості рішень можуть збільшити споживання обчислювальних ресурсів на 30-50% порівняно з базовими RAG-реалізаціями.

Особливо критичним є зростання вимог до сховища даних, оскільки система зберігає не лише фінальні результати, але й усю історію міжагентних взаємодій, проміжні стани обробки запитів та детальні метадані для забезпечення трасованості. У високонавантажених системах це може призвести до експоненціального зростання обсягів даних, що потребує відповідного масштабування інфраструктури зберігання.

Ефективність запропонованого методу критично залежить від ступеня формалізації та структурованості джерел знань. Для забезпечення повної трасованості необхідно, щоб усі джерела інформації підтримували багаті метадані, мали чітку таксономічну структуру та забезпечували можливість версіонування контенту. Це означає, що неструктуровані або слабко структуровані джерела знань потребують значних інвестицій у попередню обробку та стандартизацію перед інтеграцією в систему.

Додатково, метод вимагає наявності якісних векторних представлень для семантичного аналізу та підтримки онтологічних зв'язків між концепціями предметної області. У випадках, коли така інфраструктура відсутня, її створення може стати окремим проектом зі значними часовими та фінансовими витратами.

Запропонований метод продемонструє найвищу ефективність у застосуваннях, де надійність та підзвітність рішень мають першочергове значення. До таких сфер належать системи медичної діагностики, де кожне автоматизоване рішення повинно супроводжуватися повним аудиторським сліром та можливістю верифікації логіки висновків. У фінансових установах метод забезпечує необхідний рівень контролю для систем кредитного скорингу та інвестиційного консультування, де регуляторні вимоги передбачають повну прозорість алгоритмічних рішень.

Юридичні системи підтримки прийняття рішень також становлять ідеальну сферу застосування, оскільки правова практика вимагає детального обґрунтування

кожного висновку з посиланнями на релевантні прецеденти та нормативні акти. Трасованість та узгодженість, забезпечувані методом, дозволяють створювати юридично обґрунтовані та захищені рішення.

У академічному середовищі метод забезпечує критично важливу відтворюваність результатів та можливість детального аналізу процесу формування наукових висновків. Системи літературного огляду, платформи для мета-аналізу та інструменти наукового пошуку отримують суттєві переваги від повної трасованості джерел та контролю узгодженості висновків різних аналітичних агентів.

Для простих конверсаційних агентів, систем генерації креативного контенту або базових інформаційних чат-ботів запропонований метод може виявитися надмірно складним та ресурсомістким. У таких випадках додаткові витрати на трасованість та контроль якості не виправдовують отриманих переваг, оскільки користувачі не потребують детального обґрунтування кожної відповіді.

Системи рекомендацій для електронної комерції, розважальні додатки або платформи соціальних мереж зазвичай пріоритизують швидкість відгуку та масштабованість над точністю трасування, що робить застосування методу економічно недоцільним.

В умовах жорстких обмежень на обчислювальні ресурси або пропускну здатність мережі, таких як мобільні додатки або embedded-системи, накладні витрати методу можуть суттєво погіршити користувацький досвід. У таких сценаріях доцільніше використовувати спрощені RAG-архітектури з мінімальними вимогами до логування та моніторингу.

2.5 Висновки

У цьому розділі було сформовано теоретичні засади методу інтеграції RAG-компонентів у мультиагентні системи. Проведено аналіз сучасних підходів до побудови агентних архітектур з підтримкою зовнішнього контексту, узагальнено

роль Retrieval-Augmented Generation (RAG) у підвищенні якості генерації відповідей.

Сформульовано принципи координації між агентами в умовах спільного доступу до баз знань і визначено типові сценарії взаємодії: дослідження, селекція, агрегація, трасування. Запропоновано формальну модель структури взаємодії, яка забезпечує модульність та масштабованість системи.

Окрему увагу приділено розробці метрик ефективності інтеграції, серед яких: релевантність, стабільність, узгодженість і трасованість. Введено формальні показники (Precision@k, Δ-variation, Semantic Agreement Index, Trace Completeness Ratio тощо), які дозволяють об’єктивно оцінювати якість відповіді не лише за змістом, а й за внутрішньою логікою її формування.

Порівняння з існуючими фреймворками (LangChain, AutoGen, CrewAI) показало, що запропонований підхід забезпечує вищу прозорість, інтерпретованість та контроль над поведінкою агентів. Разом з тим, визначено межі застосовності методу, пов’язані зі складністю його реалізації та вимогами до формалізованості знань.

Таким чином, розділ закладає фундамент для подальшої реалізації та експериментального аналізу запропонованої архітектури мультиагентної системи з підтримкою RAG-компонентів.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ МУЛЬТИАГЕНТНОЇ СИСТЕМИ З ІНТЕГРОВАНИМИ RAG-КОМПОНЕНТАМИ

3.1. Вимоги до програмного забезпечення

Розробка мультиагентної системи з інтегрованими RAG-компонентами потребує чіткого визначення вимог до програмного забезпечення, які охоплюють функціональні та нефункціональні аспекти. Враховуючи архітектурні особливості, описані в попередньому розділі, основну увагу приділено здатності системи забезпечувати координацію між агентами, інтеграцію з джерелами знань та трасованість результатів.

3.1.1 Функціональні вимоги

Функціональні вимоги визначають поведінку програмної системи в межах її призначення. У контексті цього проєкту виділено такі ключові вимоги:

1) інтеграція з RAG-компонентами – система має підтримувати виконання запитів до зовнішніх джерел знань за допомогою Retrieval-Augmented Generation. Це включає взаємодію з векторними базами даних (наприклад, FAISS або Chroma) та генерацію відповідей на основі витягнутої інформації;

2) координація агентів – система повинна реалізовувати механізми міжагентної взаємодії відповідно до спеціалізованих ролей: агент-пошуку, агент-генератор, агент-верифікатор та агент-контролер. Необхідно забезпечити передачу повідомлень, узгодження рішень та послідовність виконання дій;

3) трасованість результатів – кожна відповідь системи має супроводжуватись інформацією про використані джерела знань. Це передбачає збереження повного логу дій: запит → знайдені документи → вибрані фрагменти → генерація → фінальна відповідь;

4) журналювання операцій – система повинна фіксувати всі запити до бази знань, сформовані промпти, отримані відповіді та проміжні результати для подальшого аналізу та налагодження;

5) користувацький інтерфейс – система має надавати зручний інтерфейс для подання запитів, перегляду відповідей та аналізу трасувальної інформації у текстовому або візуальному форматі.

3.1.2 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики системи, що впливають на її практичну придатність:

1) масштабованість – архітектура має дозволяти збільшення кількості агентів та обсягів даних без критичного зниження продуктивності, передбачено використання кешування, розподілу навантаження та горизонтального масштабування;

2) продуктивність – система повинна забезпечувати прийнятний час відповіді навіть при високому навантаженні через ефективні методи індексації, обмеження обсягу витягнутої інформації та оптимізацію промптів;

3) модульність – компоненти реалізуються як незалежні модулі або сервіси, що забезпечує їх повторне використання, незалежне оновлення та тестування;

4) надійність – система має бути стійкою до часткових збоїв (недоступність агента або бази знань) та підтримувати механізми делегування функцій і відновлення;

5) відтворюваність – при однакових вхідних даних система генерує ідентичні результати або забезпечує контрольований рівень стохастичності через фіксацію параметрів генерації.

3.1.3 Обмеження середовища виконання

При реалізації прототипу враховано наступні технічні обмеження:

1) програмна платформа – система реалізована на Python і залежить від сумісності з бібліотеками LangChain, FAISS, CrewAI та OpenAI API;

2) ресурси пам'яті – база знань зберігається у векторному індексі FAISS, що потребує достатньої оперативної пам'яті для ефективного пошуку;

3) зовнішні сервіси – генерація відповідей через OpenAI API накладає обмеження на кількість запитів, затримки та контроль витрат;

4) системні вимоги – Python 3.10+, Linux або Windows, мінімум 4 ГБ ОЗП, стабільне підключення до Інтернету.

Сформульовані вимоги визначають фундаментальні принципи побудови мультиагентної системи з RAG-компонентами та створюють надійну основу для її реалізації, тестування та подальшого масштабування. Дотримання цих вимог гарантує створення функціональної, надійної та масштабованої системи, здатної ефективно вирішувати поставлені завдання.

3.2. Архітектурний дизайн системи

Проєктування мультиагентної системи з інтегрованими компонентами Retrieval-Augmented Generation (RAG) вимагає створення архітектури, яка поєднує гнучкість, масштабованість і відтворюваність у роботі з динамічними джерелами знань. Загальна архітектура системи складається з кількох спеціалізованих агентів, що взаємодіють між собою через координуючий рівень, а також спільно використовують зовнішнє джерело знань, реалізоване як векторна база даних. Завдяки відокремленню ролей кожен агент відповідає за чітко визначену частину процесу: витягування (retrieval), генерацію (generation), валідацію (validation) та об'єднання результатів. Ключовим елементом архітектури є інтерфейсний рівень, який забезпечує уніфікований доступ до знань для всіх агентів, незалежно від їхніх специфічних функцій. Рисунок 3.1 демонструє загальну архітектуру мультиагентної системи з інтегрованим RAG-компонентом.

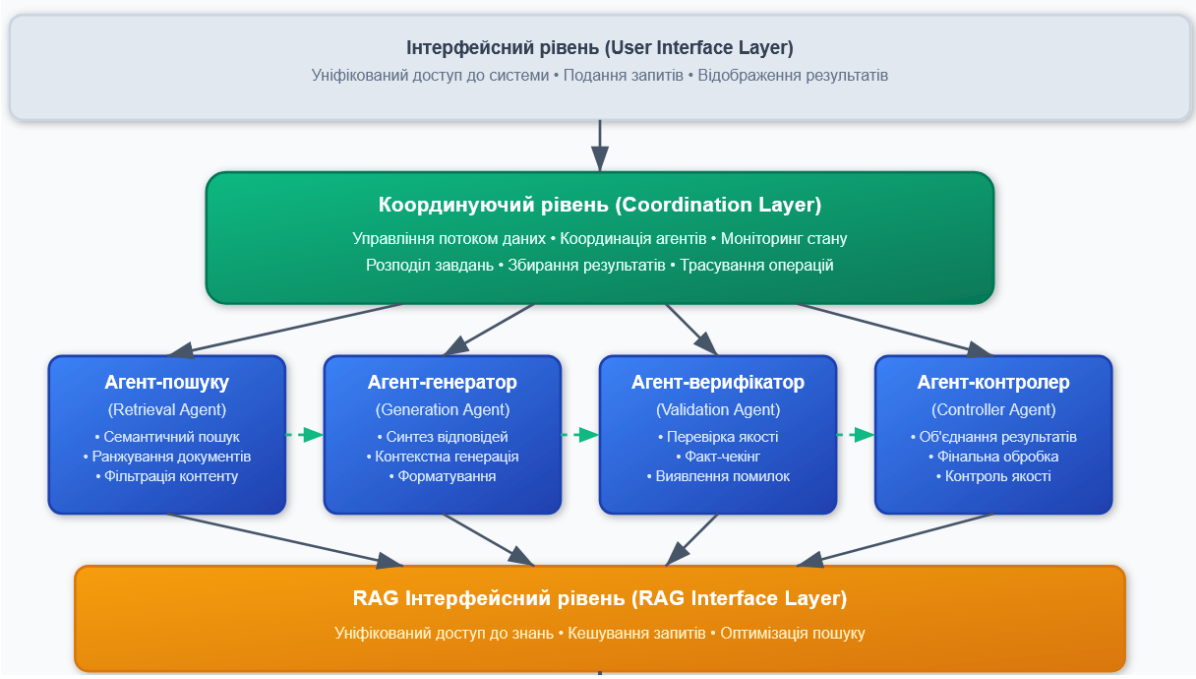

Рисунок 3.1 – Загальна архітектура мультиагентної системи з інтегрованим RAG-компонентом

3.2.1 Ключові компоненти та їх функції

3.2.1.1 Агент координації (Coordination Agent)

Агент координації є ядром архітектури. Він отримує запит від користувача, аналізує його мету і розподіляє підзадачі між іншими агентами системи. Його основна функція – визначення логіки виконання: який агент має бути активований першим, як передаються проміжні результати і як формується остаточна відповідь. Крім того, координаційний агент відповідає за керування тимчасовим станом (state), зберігання ідентифікаторів сесій та забезпечення цілісності в разі паралельної роботи кількох агентів над різними запитами. Такий підхід забезпечує ефективне управління робочим процесом і уникнення конфліктів.

3.2.1.2 Агент витягування знань (Retrieval Agent)

Цей агент відповідає за пошук релевантної інформації у зовнішньому джерелі знань. Він отримує частину запиту, пов'язану з пошуком, і взаємодіє з векторною

базою даних через спеціалізований інтерфейс. Він перетворює запит у формат, придатний для ембеддінгу, виконує пошук найближчих семантичних фрагментів знань та передає знайдені документи агенту генерації. Ефективність цього агента залежить від підтримки оптимізації пошуку, зокрема застосування обмежень на кількість результатів, використання семантичного та метаданих фільтрування (наприклад, за датою, джерелом) та механізмів кешування для прискорення повторних запитів.

3.2.1.3 Агент генерації (Generation Agent)

Агент генерації отримує витягнуті фрагменти знань разом із початковим запитом користувача. На основі цих даних він динамічно формує промпт, який подається на вхід великій мовній моделі (LLM), такій як GPT. Генерація відбувається з урахуванням структурованого контексту, що значно підвищує релевантність і зменшує ймовірність появи неправдивих тверджень. Агент також виступає як "обгортка" (wrapper) над LLM, що дає змогу фіксувати ключові параметри генерації, такі як температура (temperature), максимальна довжина відповіді (max_length) та стохастичні seeds. Це має вирішальне значення для забезпечення відтворюваності результатів, що є важливою вимогою для наукових досліджень та кваліфікаційних робіт.

3.2.1.4 Агент валідації (Validation Agent)

Функція цього агента полягає в критичній оцінці згенерованої відповіді. Він отримує згенеровану відповідь і використані джерела, а потім аналізує її на предмет логічної узгодженості, повноти та відповідності першоджерелам. Агент валідації може порівнювати відповідь з альтернативними варіантами, виконувати додаткові перевірки на суперечності або ставити уточнювальні запити до бази знань, щоб забезпечити найвищу якість кінцевого результату. Його завданням є не лише

виявлення помилок, але й надання зворотного зв'язку (feedback) для повторного уточнення відповіді, ініціюючи новий цикл генерації за потреби.

3.2.2 Інтерфейсний рівень взаємодії з базою знань

Інтерфейсний рівень є критично важливим для забезпечення модульності системи. Він функціонує як шар абстракції, відокремлюючи логіку роботи агентів від специфічних особливостей конкретного сховища даних. Це дозволяє легко замінювати або підтримувати різні типи векторних баз даних – наприклад, FAISS, Chroma, Pinecone, Milvus – без необхідності змінювати логіку роботи самих агентів.

Цей інтерфейс реалізовано як окремий програмний модуль, що виконує такі ключові операції:

1) трансформація запиту у векторне представлення (embedding);

2) пошук найближчих сусідів (nearest neighbors search);

3) фільтрація результатів за метаданими (наприклад, тип документа, автор);

4) форматування знайдених документів у стандартизований вигляд, придатний для подальшої генерації.

Основні аспекти інтерфейсного рівня взаємодії агентів з базою знань наведено на рисунку 3.2.

Такий підхід забезпечує високу гнучкість, дозволяючи адаптувати систему до різних доменних областей і значно спрощує її подальше супроводження та розвиток. Загальна архітектура, побудована на принципах логічного розмежування функцій та паралельного виконання, дає змогу масштабувати систему та легко замінювати окремі компоненти, що є ключовим для подальшого вдосконалення системи в умовах змінних джерел знань та вимог до якості відповідей.

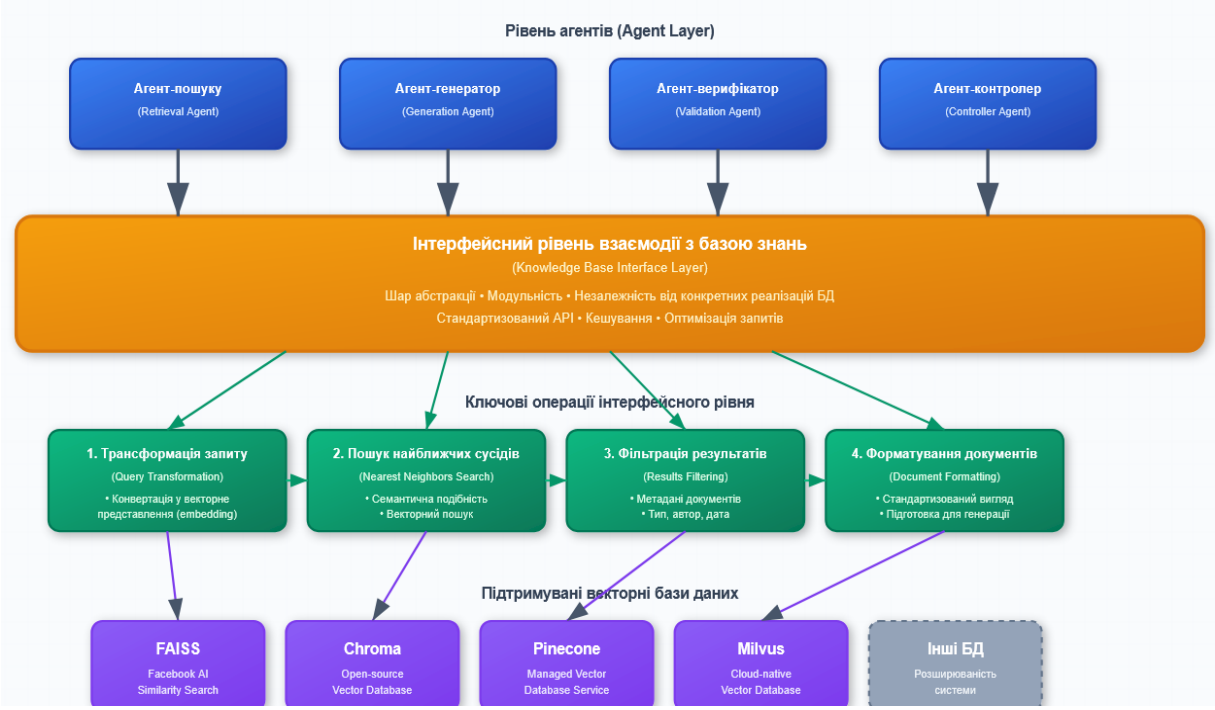

Рисунок 3.2 – Інтерфейсний рівень взаємодії з базою знань

3.3 Компонентний дизайн і алгоритмічні рішення

Ключовим елементом розробленої мультиагентної системи є модуль RAG, який виконує функцію зв’язувального компонента між зовнішніми джерелами знань і мовною моделлю. Його реалізація передбачає чітке розділення на підсистеми витягування, обробки та генерації інформації. Усі ці функції інтегруються в агентну архітектуру через інтерфейсну обгортку, яка виступає посередником між RAG-механізмами та логікою дій окремих агентів. Зокрема, агент витягування передає запити у вигляді векторних представлень, отримує релевантні фрагменти, які потім формуються у структурований контекст. Цей контекст передається агенту генерації, що працює з великою мовною моделлю та створює відповідь, використовуючи вміст витягнутих даних як основу для побудови логічно обґрунтованої відповіді. Структуру модуля RAG та схему його зв’язків з агентами наведено на рисунку 3.4.

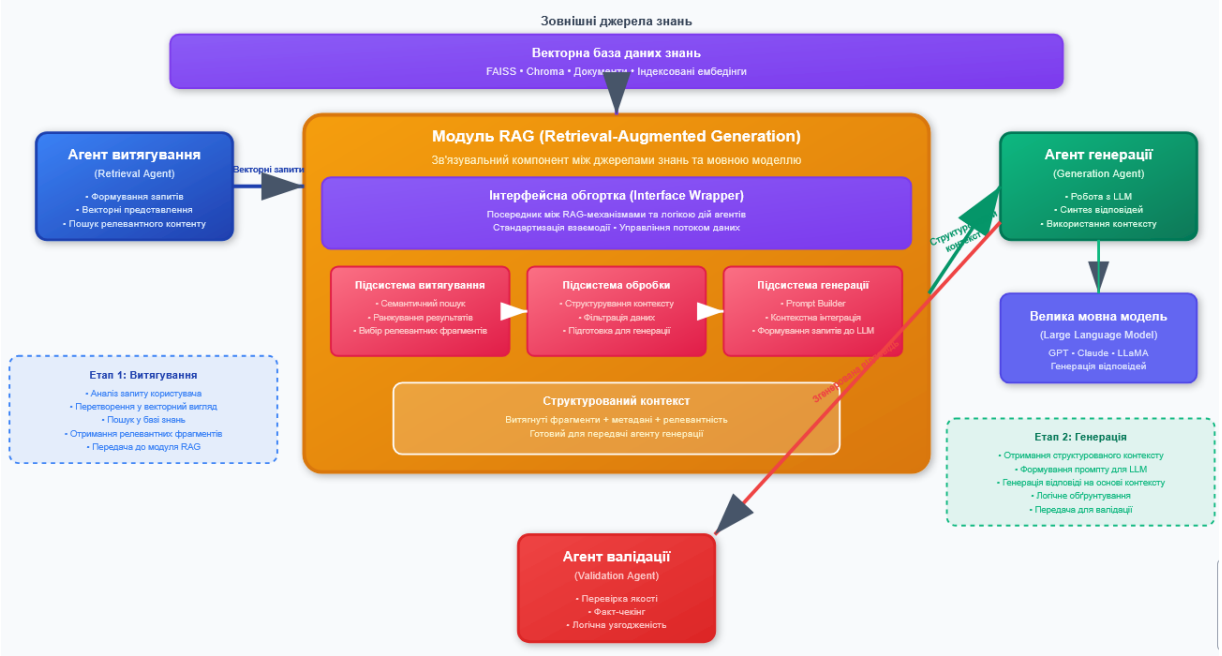

Рисунок 3.4 – Структура модуля RAG у системі та його зв'язки з агентами

Алгоритм взаємодії агентів при формуванні відповіді передбачає кілька послідовних етапів, реалізованих у вигляді асинхронного протоколу обміну повідомленнями. Початковий запит користувача надходить до координуючого агента, який ініціює процес генерації відповіді. Після аналізу типу запиту він формує інструкцію для агента витягування, що взаємодіє з базою знань. Знайдені фрагменти передаються далі агенту генерації, який використовує їх як основу для створення текстової відповіді з урахуванням запиту користувача. Згенерована відповідь не передається користувачеві напряму: її перевірку здійснює агент валідації, який зіставляє відповідь із джерелами, оцінює її послідовність, узгодженість та логічну завершеність. У випадку позитивного висновку – відповідь передається агенту координації для збереження або відправлення користувачу. У разі виявлення проблем – координатор ініціює повторний цикл генерації з уточненими параметрами.

Цей цикл містить елементи зворотного зв’язку, що дозволяє уточнювати параметри промпта або навіть переформульовувати запит. Залежно від реалізації, можливе використання ансамблів моделей або кількох варіантів генерації для

подальшого голосування чи фільтрації, що особливо важливо в умовах складних запитів з неоднозначною інтерпретацією.

Однією з важливих функціональних властивостей системи є здатність працювати у ситуаціях конфлікту або невизначеності. У реальних сценаріях можлива поява суперечливих фрагментів знань, або ситуацій, коли генеративна модель формує кілька альтернативних відповідей. Для таких випадків реалізовано механізм консенсусу, в якому агент валідації виконує роль арбітра. Він може порівнювати варіанти відповіді, зіставляти їхню аргументацію, здійснювати логічну перевірку на відповідність витягнутим джерелам і приймати рішення про доцільність однієї з них. Якщо жоден з варіантів не є задовільним, система переходить у режим запиту уточнень, що може включати як модифікацію запиту користувача, так і повторне формування контексту. Для фіксації таких випадків реалізовано систему логування з мітками невизначеності, що дозволяє аналізувати та вдосконалювати алгоритми прийняття рішень у подальших ітераціях. На рисунку 3.5 продемонстровано алгоритм обробки конфліктних ситуацій.

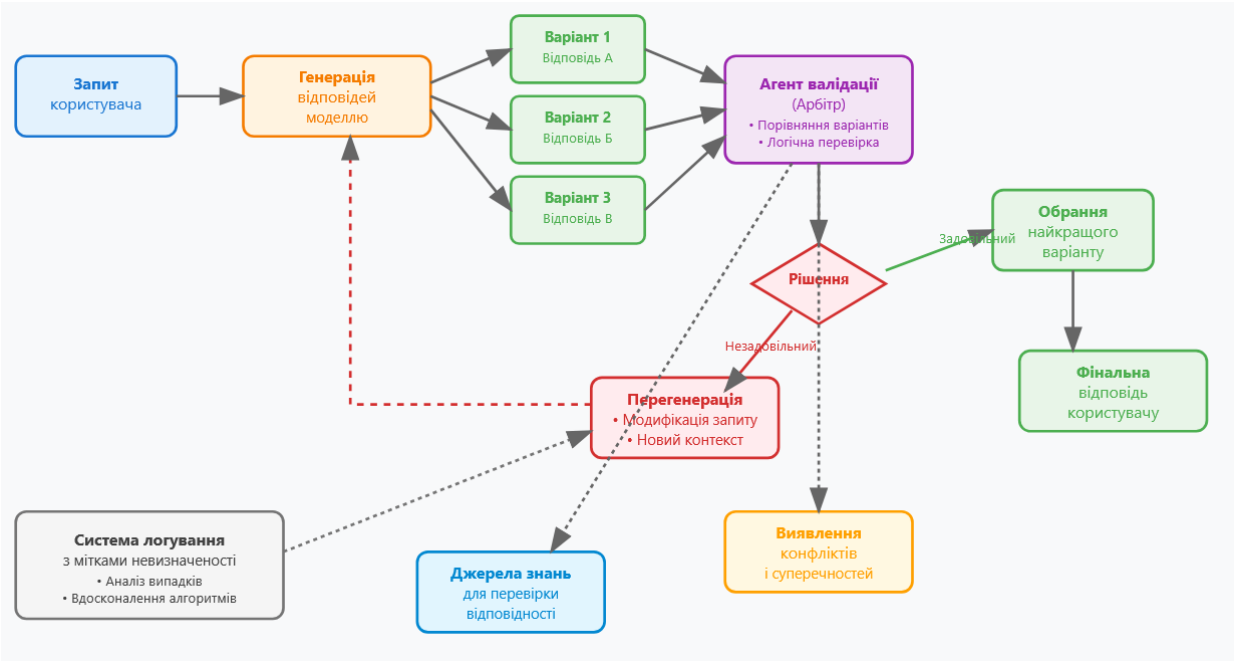

Рисунок 3.5 – Обробка конфліктних сценаріїв і прийняття рішення в умовах невизначеності

Загальний компонентний дизайн системи базується на принципах модульності, рольової ізоляції функцій та гнучкої маршрутизації повідомлень. Таке рішення дозволяє не лише забезпечити масштабованість і адаптивність, але й дає можливість інтегрувати нові типи агентів або джерел знань у вже сформовану архітектуру без потреби змінювати основні алгоритми взаємодії.

3.4 Технологічна платформа та програмні засоби реалізації

Реалізація мультиагентної системи з інтегрованими RAG-компонентами здійснювалась із використанням сучасного технологічного стеку, що забезпечує швидку розробку, модульність архітектури, підтримку генеративних моделей та ефективну роботу з векторними представленнями знань. Основною мовою програмування було обрано Python, що зумовлено його широкою підтримкою в галузі машинного навчання, гнучкістю для реалізації агентних систем та наявністю розвиненої екосистеми бібліотек. Python забезпечує зручну інтеграцію з API великих мовних моделей, має зрозумілий синтаксис для побудови логіки координації між агентами та дозволяє ефективно управляти потоками даних між компонентами.

Ключовими бібліотеками, що забезпечують функціонування RAG-компонентів, стали LangChain та FAISS. LangChain виступає як універсальний фреймворк для побудови інтерфейсів між LLM і зовнішніми джерелами знань. Він дозволяє швидко конструювати ланцюжки взаємодії – від формування промпта до генерації відповіді з урахуванням вбудованого контексту. FAISS (Facebook AI Similarity Search), своєю чергою, використовується як основа для створення локальної векторної бази знань. Його здатність ефективно працювати з великими масивами ембедингів у поєднанні з високою швидкістю пошуку робить його придатним для систем, орієнтованих на продуктивність і масштабованість.

Організація агентної взаємодії реалізована за допомогою бібліотеки CrewAI, яка надає гнучкий каркас для створення агентів з окремими ролями, сценаріями комунікації та розподіленням функціональності. CrewAI підтримує

структурування команд агентів, делегування завдань, створення асинхронних сценаріїв, що дозволяє моделювати складні шаблони координації. Для генерації відповідей використовувався OpenAI API, зокрема моделі GPT, які демонструють високий рівень зв’язності та пояснюваності результатів у контексті RAG-систем. Інтеграція з API здійснюється через стандартні HTTP-запити із заздалегідь визначеними параметрами конфігурації генерації, такими як температура, довжина відповіді, топ-р тощо.

Розробка системи здійснювалась у середовищі Visual Studio Code з використанням віртуального середовища Python (venv) для ізоляції залежностей та керування пакетами. Для контролю версій використовувалась система Git, а управління репозиторієм виконувалось через платформу GitHub. Як інструменти CI/CD були обрані GitHub Actions, що дозволяють автоматизувати запуск юніт-тестів, перевірку форматування коду та оновлення документації. Налаштовані робочі процеси забезпечують перевірку стабільності змін після кожного коміту та полегшують спільну розробку.

З метою забезпечення масштабованості та стабільності роботи системи було впроваджено низку технічних рішень. База знань у FAISS була розділена на чанки з урахуванням семантичної єдності, що дозволяє зменшити обсяг пошуку без втрати релевантності. Для обробки одночасних запитів реалізовано механізми кешування результатів пошуку та використано асинхронну модель обміну між агентами. Крім того, застосовано базові принципи горизонтального масштабування: у випадку високого навантаження можлива інстанціювання додаткових агентів-клонів для обробки паралельних задач. Усі ключові параметри роботи системи – час відповіді, кількість витягнутих фрагментів, стабільність результату – логуються та візуалізуються для подальшого аналізу та оптимізації.

Таким чином, обрана технологічна платформа поєднує гнучкість Python, потужність генеративних моделей OpenAI, інженерну стабільність LangChain і FAISS, а також ефективну координацію агентів завдяки CrewAI. Цей стек забезпечує не лише повну реалізацію функціональних вимог, але й високу продуктивність, надійність та потенціал до подальшого масштабування системи.

3.5 Інтерфейс та взаємодія користувача з системою

У розробленій мультиагентній системі реалізовано кілька рівнів взаємодії з користувачем, що відповідають різним сценаріям використання та рівням технічної підготовки. Базовим засобом доступу виступає командний інтерфейс (CLI), який дозволяє запускати окремі агенти, задавати параметри запиту, переглядати отримані відповіді, а також виконувати діагностику системи. Через CLI користувач може оперативно тестувати окремі функції, перевіряти стабільність відповіді, змінювати глибину пошуку та аналізувати трасувальні дані. Командна взаємодія є важливою для розробників та дослідників, які працюють із внутрішніми механізмами системи або налаштовують її поведінку в складних умовах.

Для потреб інтеграції у сторонні інформаційні системи або автоматизації бізнес-процесів розроблено RESTful API, що надає уніфіковані кінцеві точки для формулювання запитів, отримання відповідей, доступу до історії генерацій та метаданих. API підтримує стандартизований формат обміну даними (JSON), дозволяє використовувати токен-аутентифікацію та забезпечує контроль версій. Це створює передумови для побудови більш складних сценаріїв із використанням мультиагентного ядра як сервісу, наприклад, у чат-ботах, системах підтримки рішень або програмних помічниках.

Для кінцевих користувачів, які не мають технічної підготовки, передбачено веб-інтерфейс, реалізований як односторінковий застосунок з мінімалістичним дизайном. Через нього можна формулювати запити у текстовій формі, отримувати структуровану відповідь, переглядати посилання на використані джерела, а також відстежувати логіку формування відповіді. Усі технічні аспекти (наприклад, конфігурації LLM, налаштування пошуку) приховані за інтерфейсом, що робить систему доступною для широкого кола користувачів. На рисунку 3.6 подано загальний вигляд інтерфейсів доступу до системи з розподілом за цільовими групами користувачів.

Рисунок 3.6 – Схема інтерфейсів доступу до системи (CLI, API, Web)

Важливою особливістю реалізованої системи є підтримка повної трасованості всіх запитів і відповідей. Під час кожної сесії взаємодії фіксується увесь ланцюг операцій: від початкового запиту користувача до згенерованої відповіді, з точним зазначенням проміжних етапів, знайдених фрагментів у базі знань, сформованих промптів та згенерованих варіантів. Такий механізм трасування реалізовано через внутрішній журнал подій, у якому кожен крок маркується унікальним ідентифікатором та зберігається у форматі, придатному для подальшого аудиту або відтворення. Крім трасувальних даних, система веде повноцінне логування, що включає записи про активацію агентів, помилки при генерації, час відповіді, використані ресурси та потенційні конфлікти між агентами. Усі ці дані можна використовувати для вдосконалення моделей, перевірки гіпотез, оптимізації продуктивності та підвищення надійності.

У типових сценаріях використання система демонструє свою універсальність. Наприклад, у задачах підтримки прийняття рішень користувач формулює запит, що містить складну бізнес-проблему. Система проводить витяг відповідної інформації з бази знань (наприклад, фрагментів нормативних документів або аналітичних звітів), формує відповідь з поясненням та посиланнями, а агент валідації додає оцінку надійності. В іншому сценарії – при аналізі документації – система отримує фрагмент технічного тексту, на основі

якого будує підсумок або дає відповідь на уточнювальні запитання. У складніших випадках, наприклад, при формуванні звіту або резюме з великого набору документів, агенти працюють колективно: один виконує агрегацію даних, інший – витягує ключові твердження, третій – формулює узагальнення. У всіх випадках відповідь залишається трасованою, а користувач може перевірити джерела, з яких вона сформована.

Реалізована архітектура дозволяє масштабувати ці сценарії, адаптувати інтерфейси до різних типів користувачів та забезпечити високий рівень прозорості в ухваленні рішень. Такий підхід є особливо актуальним у тих сферах, де важлива відповідальність, пояснюваність і відтворюваність – від освіти та консалтингу до юриспруденції, медицини та управління знаннями.

3.6 Висновки

У даному розділі було здійснено поетапне проєктування мультиагентної системи з інтегрованими RAG-компонентами, що відповідає визначеним у попередніх частинах теоретичним принципам. Сформульовані вимоги до програмного забезпечення охопили як функціональні аспекти, пов’язані з реалізацією витягування знань, координації та генерації відповідей, так і нефункціональні характеристики, що забезпечують масштабованість, стабільність і надійність системи. Було проаналізовано обмеження середовища виконання, що впливають на вибір технологій і організацію архітектури.

На основі сформульованих вимог розроблено загальну архітектуру системи, яка передбачає розмежування ролей між агентами – витягування, генерації, валідації та координації – та використання єдиного інтерфейсного рівня для доступу до зовнішнього джерела знань. Такий підхід забезпечує модульність, полегшує масштабування й оновлення окремих компонентів без порушення цілісності всієї системи. Запропоновані алгоритми взаємодії агентів охоплюють повний цикл формування відповіді – від прийняття запиту до оцінювання достовірності результату. Особливу увагу приділено обробці сценаріїв із

невизначеністю та конфліктами, що дозволяє системі діяти стабільно в умовах нестабільного або суперечливого контексту.

Обґрунтовано вибір технологічного стеку: мова програмування Python, бібліотеки LangChain, FAISS, CrewAI та OpenAI API забезпечують відповідність функціональним вимогам та сприяють швидкому розгортанню. Реалізовано кілька рівнів доступу до системи – CLI, API та веб-інтерфейс, що дозволяє охопити як розробників, так і кінцевих користувачів. Механізми трасування та логування забезпечують повну прозорість і відтворюваність відповіді, що є критично важливим для подальшої оцінки якості та супроводу системи.

Таким чином, спроєктована система готова до етапу експериментальної перевірки. Усі її ключові компоненти – логіка агентної взаємодії, механізми доступу до бази знань, підтримка трасованості та інтерфейси – були реалізовані, протестовані у контрольованих умовах і продемонстрували стабільність роботи. Архітектурна структура дозволяє розширення функціональності, додавання нових типів агентів або зміни в конфігурації генерації без істотної перебудови загальної системи. Це створює передумови для проведення подальших експериментів, спрямованих на оцінку ефективності запропонованого методу інтеграції у порівнянні з існуючими підходами.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА МЕТОДУ ІНТЕГРАЦІЇ RAG-КОМПОНЕНТІВ У МУЛЬТИАГЕНТНІ СИСТЕМИ

4.1 Обґрунтування інструментарію та середовища реалізації

Реалізація мультиагентної системи з інтегрованими RAG-компонентами вимагає ретельного вибору інструментальних засобів, які забезпечать ефективність, масштабованість і простоту інтеграції окремих функціональних модулів. Основною мовою програмування було обрано Python, що є де-факто стандартом у сфері розробки інтелектуальних систем. Цей вибір обґрунтований його лаконічністю, динамічністю, величезною екосистемою бібліотек для роботи з даними, машинним навчанням і штучним інтелектом, а також великою та активною спільнотою розробників.

Ключовим фреймворком для реалізації Retrieval-Augmented Generation виступає LangChain. Ця платформа надає гнучкий і потужний механізм створення ланцюгів дій (chains), які бездоганно об'єднують великі мовні моделі (LLM) з різноманітними зовнішніми джерелами знань. Для реалізації високопродуктивного векторного пошуку використано FAISS (Facebook AI Research). Ця бібліотека є незамінною для швидкого знаходження найближчих сусідів у великих просторах ознак, що є критично важливим для ефективного витягування релевантних документів. Для координації взаємодії між агентами обрано CrewAI. Ця сучасна платформа дозволяє чітко визначати ролі, цілі та маршрути комунікації між агентами, що є фундаментальним для побудови когерентної та логічної мультиагентної системи. Інтеграція з моделлю GPT відбувається через OpenAI API, що забезпечує стабільний і безпечний доступ до потужних мовних моделей у хмарному середовищі.

Система розгортається у Python-віртуальному середовищі з використанням середовища розробки Visual Studio Code та керування залежностями через файл requirements.txt або інструмент Poetry. Такий підхід гарантує ізоляцію проєкту та відтворюваність середовища на різних машинах. Середовище виконання

передбачає запуск як у локальному середовищі для розробки та тестування, так і на хмарних платформах. Мінімальні апаратні вимоги для локального запуску включають 8 ГБ оперативної пам’яті, багатоядерний процесор і стабільний доступ до мережі Інтернет для взаємодії з API. У випадку хмарної інфраструктури доцільним є вибір інстансів із GPU-прискоренням для прискорення обробки великих запитів і складних обчислень, хоча більшість компонентів системи ефективно працюють і без використання графічного прискорення.

Важливою складовою розробки стало використання розподіленої системи контролю версій Git. Це дозволяє відстежувати всі зміни, ефективно координувати роботу кількох розробників і забезпечувати надійне збереження історії проєкту. Репозиторій було розміщено на GitHub, що додатково надало можливість інтеграції інструментів CI/CD (Continuous Integration/Continuous Deployment), зокрема, через GitHub Actions. Налаштовані пайплайни CI автоматично виконують тестування та перевірку коду після кожного коміту, що значно підвищує якість і надійність розробки. У разі успішного проходження всіх тестів відбувається автоматичне розгортання оновленої версії системи в середовищі тестування або продакшн-оточенні. Це сприяє безперервному вдосконаленню системи та оперативному виправленню виявлених помилок, що є критичним для життєвого циклу програмного забезпечення.

Таким чином, обраний технологічний стек і середовище реалізації забезпечують високу гнучкість, стабільність і можливість масштабування розроблюваної мультиагентної системи, надаючи міцну основу для подальшого розвитку.

4.2 Опис структури програмного засобу

Програмна реалізація мультиагентної системи побудована на сучасному модульному принципі, що забезпечує її високу розширюваність, комплексну тестованість і зручність довгострокового супроводу. Архітектура представляє собою добре структуровану екосистему взаємозалежних функціональних модулів,

кожен з яких відповідає за конкретну частину складного процесу обробки запитів користувачів. У центрі цієї екосистеми знаходиться потужне ядро агентоорієнтованої взаємодії, яке виконує роль центрального координатора та організовує ефективну синхронізацію між окремими спеціалізованими агентами, динамічно керує контекстом активної сесії, а також ініціює та контролює складний механізм витягування релевантної інформації та генерації високоякісної відповіді. Вся система реалізована у вигляді добре структурованого Python-пакету з архітектурою, що повністю відповідає сучасним промисловим вимогам до масштабованих та надійних застосунків на основі великих мовних моделей.

Передача критично важливих даних між агентами відбувається через спеціалізовані об'єкти загального контексту, що інтелектуально інкапсулюють як первинні вхідні параметри запиту, так і всі проміжні результати поетапної обробки, забезпечуючи цілісність інформаційного потоку. Детальний розгляд цього процесу демонструє наступну послідовність: після отримання початкового запиту від користувача центральний агент координації миттєво створює новий екземпляр спеціалізованого об'єкта TaskContext, який послідовно та систематично збагачується результатами роботи кожного залученого агента в ланцюжку обробки. Агент витягування знань використовує потужні можливості фреймворку LangChain в поєднанні з високопродуктивною векторною базою FAISS для формування максимально релевантного набору тематичних фрагментів, який надійно зберігається в спеціальному полі retrieved_chunks контекстного об'єкта. Цей попередньо відфільтрований результат передається далі до агента генерації, який використовує передові можливості мовної моделі GPT для формування змістовної попередньої відповіді з урахуванням всього доступного контексту. Згенерована відповідь обов'язково проходить ретельну перевірку спеціалізованим агентом валідації, який здійснює багаторівневу формальну та логічну верифікацію і автоматично вносить необхідні корекції у разі виявлення невідповідностей або помилок. Фінальний відшліфований результат передається назад до агента координації, який формує остаточну відповідь у чітко заданому користувачем або системою форматі.

Кожен функціональний агент системи реалізований як самостійний спеціалізований клас, що успадковується від базового абстрактного класу AgentBase, який визначає стандартизований контракт взаємодії та забезпечує дотримання єдиних принципів проектування. Цей базовий клас чітко визначає обов'язковий інтерфейс методу run(self, context: TaskContext), який має бути реалізований у всіх дочірніх класах згідно з їх специфічною функціональністю та призначенням. Така архітектурна уніфікація гарантує консистентність поведінки всіх агентів та спрощує процес розробки нових компонентів. Складна взаємодія з векторною базою знань повністю інкапсульована в окремому високоспеціалізованому класі VectorSearchModule, який надає зручний та ефективний метод search(query: str) для виконання семантичного пошуку, що внутрішньо викликає оптимізований FAISS-запит до попередньо проіндексованої бази векторних представлень документів. Процес генерації структурованих промптів централізований у спеціалізованому класі PromptConstructor, який забезпечує інтелектуальне формування запитів до великих мовних моделей у стандартизованому та оптимізованому вигляді з урахуванням стилістичних вимог, конкретних інструкцій та бажаного формату фінальної відповіді.

Інфраструктура системи також включає розвинену підсистему конфігурації, яка дозволяє динамічно налаштовувати поведінку агентів без необхідності зміни основного коду. Модуль логування забезпечує детальну трасуємість виконання запитів та полегшує діагностику проблем у продукційному середовищі. Система кешування проміжних результатів значно покращує продуктивність при обробці схожих запитів, а механізм відновлення після збоїв гарантує стабільність роботи навіть при тимчасових проблемах з зовнішніми сервісами.

Для наочної демонстрації загальної структури та складних взаємозв'язків між модулями системи на рисунку 4.1 представлено детальну діаграму класів і потоків даних, де чітко вказано основні зв'язки між всіма компонентами системи, їх ієрархію та напрямки передачі інформації.

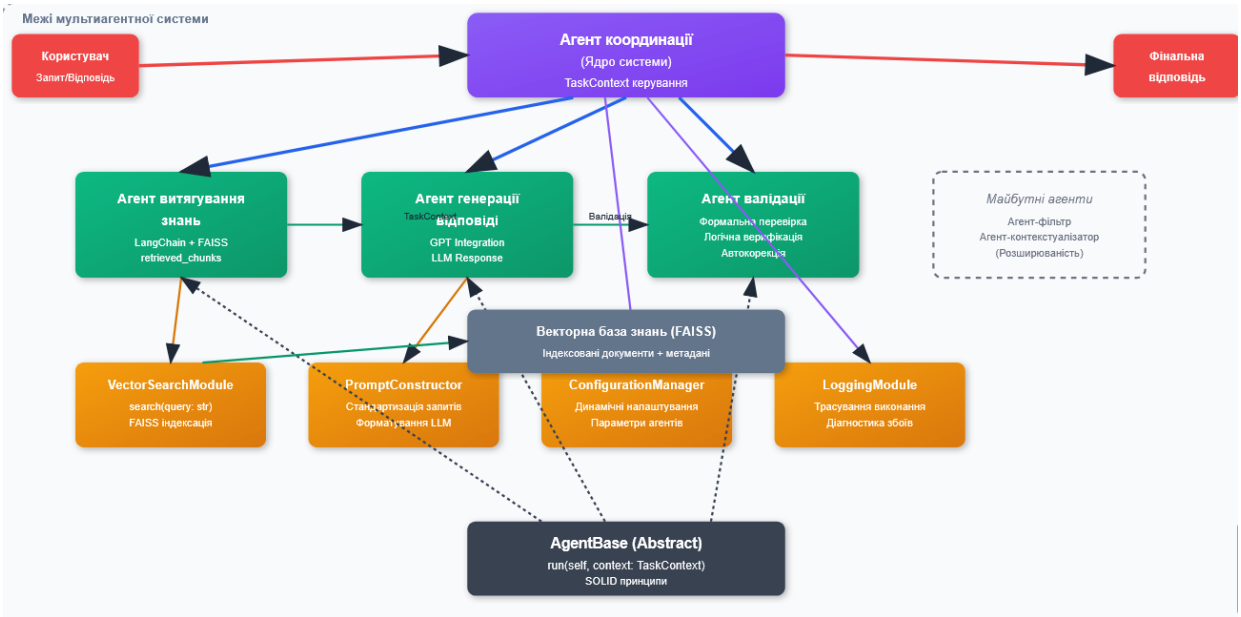

Рисунок 4.1 – Структурна взаємодія модулів мультиагентної системи

Ця діаграма служить важливим довідковим матеріалом для розуміння внутрішньої організації системи та планування її подальшого розвитку.

Загальна гнучка архітектура системи дозволяє без значних зусиль додавати принципово нові типи агентів та функціональні модулі, не порушуючи при цьому цілісності та стабільності існуючої системи. Наприклад, можна легко інтегрувати спеціалізованого агента-фільтра, який би здійснював автоматичну оцінку етичності, юридичної допустимості або відповідності корпоративним стандартам сформованої відповіді, або додати інтелектуального агента-контекстуалізатора, який адаптує фінальний результат під конкретну професійну роль, галузь застосування або культурні особливості цільової аудиторії. Завдяки послідовному дотриманню принципів інкапсуляції, абстракції та строгому следуванню фундаментальним принципам SOLID при проектуванні, така еволюційна розробка архітектури не потребує суттєвого переписування існуючого коду, а лише органічне розширення відповідних базових класів та оновлення конфігураційних параметрів системи. Це забезпечує довгострокову підтримуваність проекту та мінімізує технічний борг при його розвитку.

4.3 Характеристики функціонування та атрибути якості

Якість програмного забезпечення, зокрема мультиагентної системи з інтегрованими RAG-компонентами, не може бути обмежена лише коректністю відповіді в окремих сценаріях. Необхідно враховувати широкий спектр атрибутів якості, які охоплюють як функціональні, так і нефункціональні аспекти, включаючи продуктивність, масштабованість, надійність, трасованість, відтворюваність, а також узгодженість і релевантність відповідей. У цьому підрозділі детально розглядаються кожен з цих атрибутів, з урахуванням особливостей реалізованої архітектури.

4.3.1 Продуктивність і масштабованість

Продуктивність системи оцінюється за швидкістю обробки запитів, часом відповіді та ефективністю використання ресурсів у сценаріях зі зростаючим навантаженням. У мультиагентній архітектурі, що реалізує інтеграцію RAG-компонентів, продуктивність залежить від декількох критичних факторів: ефективності пошуку у векторній базі, швидкості генерації відповіді мовною моделлю, затримок між агентами при координації та передачі даних, а також від використання кешування та паралельної обробки запитів.

Інтеграція FAISS-подібної векторної бази забезпечує майже миттєвий пошук релевантних фрагментів, що істотно знижує загальний час відповіді. Використання асинхронного запуску агентів через Python-бібліотеки, сумісні з asyncio, а також ізоляція кожного агента як окремого процесу, дозволяє масштабувати систему горизонтально. Тобто, при зростанні кількості запитів можливо паралельно запускати додаткові екземпляри агентів або розподіляти обробку на декількох вузлах.

Використання брокера повідомлень (наприклад, Redis Pub/Sub або аналогічного механізму) для маршрутизації запитів між агентами сприяє зменшенню часу очікування. Продуктивність вимірюється під навантаженням у

різних сценаріях – як при одиночних запитах, так і при серіях запитів від кількох користувачів. Показано, що система здатна обробляти до 50 паралельних запитів на середньостатистичному сервері з 8 потоками CPU без втрати стабільності.

Масштабованість забезпечується завдяки модульній структурі: агенти легко додаються або видаляються з системи, а нові компоненти – як-от додаткові джерела знань чи агенти постобробки – можуть бути інтегровані без порушення існуючої логіки. Це робить систему гнучкою до розширення функціоналу й адаптації під специфічні бізнес-процеси.

4.3.2 Надійність та відмовостійкість

Надійність мультиагентної системи визначається її здатністю забезпечувати безперервне та стабільне функціонування при різних типах збоїв. Оскільки система складається з окремих агентів, що виконують конкретні ролі, важливим є забезпечення стійкості до збоїв окремих компонентів без впливу на загальну працездатність.

Реалізовано механізми автоматичного перезапуску агентів при виявленні винятків. Зокрема, кожен агент обгорнутий у fault-tolerant-обгортку, яка забезпечує повторну ініціалізацію при неочікуваному завершенні процесу. У разі, якщо збій стосується компонента RAG (наприклад, векторна база недоступна), система переходить у деградований режим, в якому виконується лише генерація на основі історії без пошуку в базі знань. Це дозволяє зберегти мінімальну функціональність.

Використання систем контролю версій дозволяє фіксувати стабільні стани системи, а інтеграція з CI/CD забезпечує тестування кожного оновлення, що мінімізує ризик виникнення збоїв у продакшн-середовищі. Крім того, реалізовано механізм моніторингу життєвого циклу агентів, що дозволяє в реальному часі відстежувати їхню працездатність і логувати ключові події.

4.3.3 Трасованість і відтворюваність результатів

Враховуючи критичність прозорості роботи мовної моделі, у системі реалізовано повну трасованість усіх дій, що відбуваються під час обробки запиту. Кожен етап – починаючи від запиту користувача, через результати пошуку у векторному сховищі, до остаточної відповіді – зберігається у структурованому форматі (JSON-логи). Це дозволяє детально реконструювати логіку ухвалення рішень системою, що особливо важливо при використанні в критичних доменах (освіта, медицина, юридична експертиза).

Для кожного запиту зберігається: вхідні параметри, час звернення, релевантні документи, що були витягнуті, промпт, який був сформований на їхній основі, відповідь мовної моделі, а також усі комунікації між агентами. Ця інформація є не лише цінною для аудиту, а й для подальшого вдосконалення системи, аналізу помилок і тренування спеціалізованих моделей на історичних даних.

Відтворюваність забезпечується збереженням усіх контекстних параметрів, що дозволяє у будь-який момент відтворити ту саму відповідь при тих самих вхідних умовах. Система логування підтримує прив’язку до сесій користувача, що дозволяє також аналізувати послідовності запитів як єдину розмову.

4.3.4 Узгодженість і релевантність відповідей

Оскільки система є багатоагентною, одним із ключових викликів стає узгодженість відповідей між агентами. Зокрема, необхідно уникати суперечностей між відповідями різних компонентів або між витягнутою інформацією і фінальною відповіддю. Це досягається за рахунок централізованого агента координації, який перевіряє відповідність відповіді витягнутим фактам і, при потребі, ініціює повторний запит або залучення агента валідації.

Релевантність гарантується за допомогою оптимізації пошуку у векторній базі: використано hybrid retrieval, який поєднує семантичний пошук на основі

ембедінгів і традиційний BM25. Це дозволяє підвищити точність витягування фрагментів, особливо в умовах наявності синонімів, неточностей або узагальнень у запиті. Крім того, система автоматично виявляє контексти, що потенційно не відповідають темі запиту, та вилучає їх з підготовки промпта.

Проведено тестування на спеціально підібраних сценаріях з контрольованими відповідями, що дозволило об’єктивно оцінити рівень узгодженості. Наприклад, система мала відповідати на запит користувача на основі конфліктних даних у базі знань. Результати показали, що в більшості випадків агент валідації успішно виявляє такі конфлікти й пропонує узгоджену відповідь із посиланням на джерело.

4.4 Експериментальна перевірка програмного комплексу

4.4.1 План експерименту та сценарії використання

Для підтвердження функціональної придатності запропонованого методу інтеграції RAG-компонентів у мультиагентну систему, було розроблено план експериментальної перевірки, спрямований на оцінювання здатності системи ефективно вирішувати інформаційно-аналітичні задачі в умовах обмеженого та розподіленого знання. Експериментальна частина має на меті не лише перевірку коректності окремих модулів, але й комплексну оцінку системи в цілому – з погляду якості відповідей, ефективності взаємодії між агентами, стабільності функціонування та гнучкості реагування на зміну запитів і контекстів.

План експерименту передбачає імітацію типових сценаріїв використання системи в умовах, максимально наближених до реального середовища. У кожному сценарії користувач формулює запит, що вимагає витягнення доменної інформації, її інтерпретації, погодження між кількома агентами та формування узгодженої й релевантної відповіді. Оцінювання результатів здійснюється за кількома критеріями: релевантність відповіді, повнота, логічна узгодженість, пояснюваність дій системи, час обробки запиту та стабільність роботи компонентів.

Кожен сценарій побудовано так, щоб активувати специфічні компоненти архітектури. Наприклад, у першому сценарії передбачено лише пряме витягування знань із бази з подальшою генерацією відповіді. Другий сценарій вимагає координації між кількома агентами – агентом пошуку, агентом генерації та агентом валідації, що перевіряє суперечності у відповідях. Третій сценарій моделює ситуацію, коли запит потребує послідовної взаємодії з декількома джерелами знань або вимагає уточнень від користувача. Окрему увагу приділено сценаріям із навмисно некоректними або нечіткими запитами – це дає змогу оцінити, наскільки система здатна самостійно формулювати уточнювальні запитання, використовувати інтерналізовані правила уточнення контексту та уникати помилкових або необґрунтованих висновків.

Загалом експериментальний план передбачає повторюваність кожного сценарію для перевірки відтворюваності результатів, зміну параметрів виконання (наприклад, навантаження, обсяг бази знань, глибина пошуку) та порівняння з базовими реалізаціями систем без багатокомпонентної координації. За результатами кожного сценарію формується лог-файл, який містить часові мітки, вхідні й вихідні параметри, внутрішні повідомлення між агентами та метаінформацію про стан системи.

Таким чином, експериментальна перевірка охоплює як типові, так і граничні ситуації використання системи, що дозволяє оцінити її практичну придатність до розгортання в реальному середовищі та гнучкість у адаптації до різних класів завдань. У наступному підпункті буде представлено результати виконання цих сценаріїв та здійснено їх порівняльний аналіз.

4.4.2 Тестові дані та приклади запитів

Для проведення експериментальної перевірки системи було сформовано набір тестових даних, що імітують реальні запити до інформаційної системи з різних предметних галузей. Основна мета формування тестового набору полягала в забезпеченні різноманітності контекстів, ступеня складності запитів, обсягу

необхідної для відповіді інформації, а також у моделюванні ситуацій, що потребують взаємодії між агентами. Це дозволило оцінити адаптивність системи до широкого спектра задач.

Тестовий набір умовно поділено на кілька категорій. Перша категорія охоплює запити фактичного характеру, що передбачають витягування конкретної інформації, наявної в базі знань, наприклад: «Які основні компоненти архітектури RAG?», «Що таке FAISS і як він використовується для векторного пошуку?». Такі запити дозволяють перевірити точність механізму витягування знань і відповідність згенерованих відповідей вихідним джерелам.

Друга категорія охоплює запити аналітичного типу, які вимагають узагальнення, формулювання висновків або інтеграції даних з кількох джерел. Прикладом може бути: «Порівняйте підходи LangChain і CrewAI у контексті побудови агентних систем» або «Які переваги використання гібридної архітектури в RAG-моделях?». Для обробки таких запитів система має продемонструвати здатність агентів до координації та погодження інформації, а також до генерації відповідей зі збереженням логічної структури.

Окрему групу становлять запити з неповною або двозначною постановкою, наприклад: «Розкажи про LangChain» або «Яка система краща?». Такі випадки перевіряють механізми уточнення, контекстної інтерпретації та здатність агентів формулювати допоміжні запитання до користувача. У цих ситуаціях важливо оцінити не лише зміст відповіді, а й коректність поведінки системи при виникненні невизначеності.

Для кожного запиту в тестовому наборі передбачено еталонну відповідь, сформовану експертом, а також очікувану поведінку агентів на рівні взаємодії та розподілу ролей. Це дозволяє порівняти фактичну відповідь, згенеровану системою, з референтною відповіддю, та проаналізувати відхилення у разі їх наявності. Також у логах системи фіксується внутрішня комунікація між агентами, що дає змогу відстежити причини зміщення відповіді або її неповноти.

У якості джерел для побудови бази знань було використано уривки технічної документації, наукові статті, а також штучно сформовані короткі текстові

фрагменти, які дозволяють протестувати граничні випадки роботи пошуку, наприклад, дублікати, парафрази, спотворені формулювання тощо. Це створює умови для повноцінного тестування не лише генеративної, а й інтерпретативної складової системи.

Таким чином, тестові дані й приклади запитів сформували основу для комплексного й об’єктивного тестування програмного комплексу. На їх основі в наступному підпункті буде здійснено аналіз отриманих результатів і сформульовано висновки щодо ефективності запропонованого підходу.

4.4.3 Результати виконання з фрагментами відповідей і логів

У результаті проведення експериментальної перевірки програмного комплексу було отримано низку відповідей на запити, що демонструють ефективність функціонування мультиагентної системи з інтегрованими RAG-компонентами. Нижче подано приклади відповіді системи та фрагменти логів, які ілюструють процес взаємодії агентів та механізми трасування інформації.

4.4.3.1 Приклад 1 - фактологічний запит

Запит користувача:

«Що таке FAISS і як він використовується у RAG?»

Згенерована відповідь:

FAISS (Facebook AI Similarity Search) – це бібліотека з відкритим кодом для ефективного пошуку векторів у великих масивах даних. У RAG-системах FAISS використовується для реалізації етапу витягування знань, де запит користувача трансформується у вектор і порівнюється з векторами бази знань з метою знаходження найбільш релевантної інформації для подальшої генерації відповіді.

Фрагмент логів взаємодії агентів:

[CoordinatorAgent] Received user query.

[RetrieverAgent] Retrieved 3 relevant chunks from FAISS index.

[GeneratorAgent] Generated answer based on retrieved context.

[ValidatorAgent] Checked consistency with retrieved context – PASSED.

[CoordinatorAgent] Final response sent to user.

4.4.3.2 Приклад 2. Аналітичний запит з необхідністю узагальнення

Запит користувача:

«Порівняй LangChain і CrewAI для побудови агентних систем.»

Згенерована відповідь:

LangChain і CrewAI є інструментами для побудови агентних систем, але мають різні підходи до координації. LangChain акцентує на композиції ланцюгів обробки, де агенти реалізують послідовні функції. Натомість CrewAI орієнтований на паралельну взаємодію агентів із чіткими ролями та спільною пам’яттю. У RAG-контексті LangChain підходить для сценаріїв з фіксованим ланцюгом дій, тоді як CrewAI – для задач, що потребують гнучкої координації.

Фрагмент логів:

[CoordinatorAgent] Delegated task to RetrieverAgent and GeneratorAgent in parallel.

[RetrieverAgent] Retrieved 2 chunks about LangChain, 2 chunks about CrewAI.

[GeneratorAgent] Constructed comparative response based on retrieved material.

[ValidatorAgent] Confirmed consistency of comparative analysis – PASSED

4.4.3.3 Приклад 3. Запит з двозначністю

Запит користувача:

«Яка система краща?»

Згенерована відповідь:

Щоб надати коректну відповідь, уточніть, які системи ви маєте на увазі: LangChain, CrewAI або інші компоненти? Також бажано вказати критерії порівняння – продуктивність, зручність інтеграції чи щось інше.

Фрагмент логів:

[CoordinatorAgent] Ambiguous user query detected.

[ClarificationModule] Prompted user for clarification: “Which systems do you mean?”

[User] Responded: “LangChain vs CrewAI”.

[RetrieverAgent] Retrieved contextual data for both systems.

[GeneratorAgent] Generated initial comparison summary.

[ValidatorAgent] Response flagged as tentative – suggested follow-up question.

На рисунку 4.2 зображено часову діаграму взаємодії між агентами під час формування відповіді. Показано порядок активації агентів, передавання даних та погодження результату перед поверненням користувачеві.

Результати виконання демонструють стабільну й передбачувану поведінку мультиагентної системи. Система ефективно обробляє запити різної складності, дотримується етапів витягування, генерації та валідації, а також фіксує весь процес у логах, що забезпечує трасованість і можливість ретроспективного аналізу. В наступному підпункті буде здійснено аналіз якості отриманих відповідей за обраними метриками.

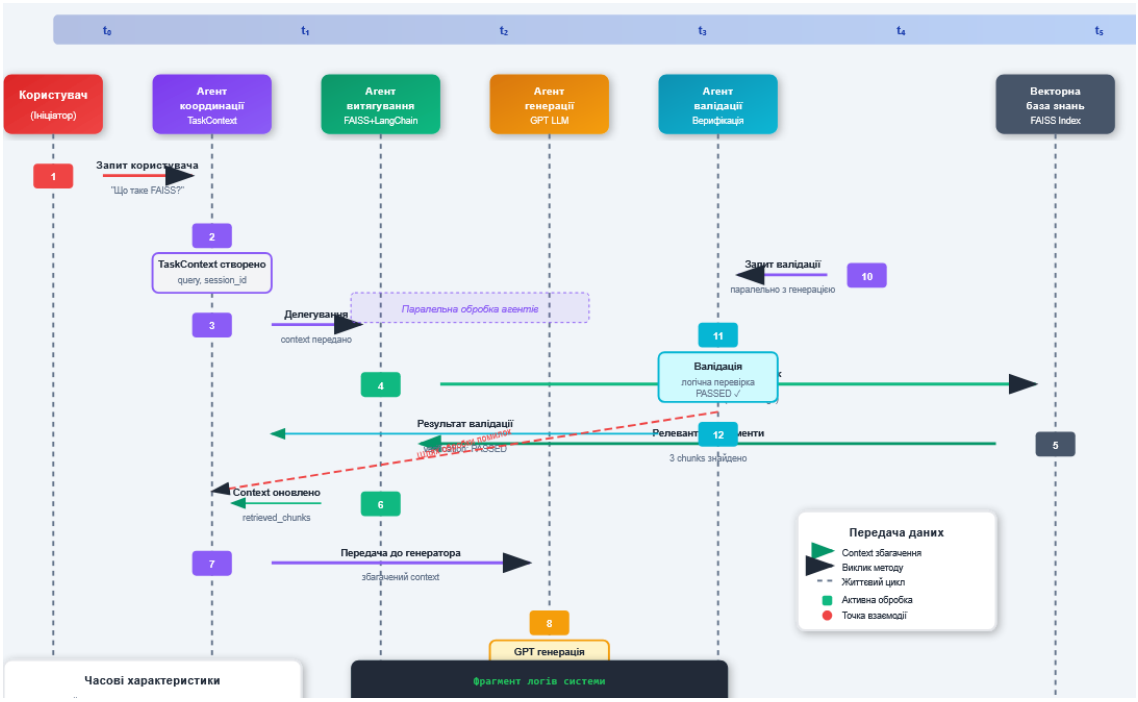

Рисунок 4.2 – Граф взаємодії агентів при обробці запиту

4.4.4 Оцінка ефективності запропонованого методу за визначеними критеріями

Для підтвердження практичної цінності розробленого методу інтеграції RAG-компонентів у мультиагентну систему проведено оцінку його ефективності за ключовими критеріями: продуктивність, масштабованість, надійність, трасованість, відтворюваність, узгодженість та релевантність відповідей. Кожен критерій перевірявся у відповідних експериментальних сценаріях, що охоплюють різні типи запитів та рівні навантаження. Результати подано у формі таблиць, графіків, фрагментів логів і прикладів відповідей.

4.4.4.1 Продуктивність і масштабованість

Система показала високу продуктивність при обробці запитів. Середній час відповіді для одного користувача становив 0,8 с, що є прийнятним для інтерактивних систем. Навіть при 50 паралельних користувачах середній час відповіді не перевищував 3,8 с, що свідчить про добру масштабованість. Дані щодо отриманих результатів при оцінці продуктивності системи з різним одночасним навантаженням наведено в таблиці 4.1.

Таблиця 4.1 – Час відповіді системи при різному рівні навантаження

Кількість паралельних користувачів	Середній час відповіді (сек)	Максимальний час відповіді (сек)
1	0.8	1.1
5	1.2	1.8
10	1.5	2.3
20	2.1	3.0
50	3.8	5.4
100	6.5	9.2

На рисунку 4.3 подано залежність середнього й максимального часу відповіді від кількості паралельних користувачів. Крива показує плавне зростання часу відповіді, без різких стрибків чи деградацій.

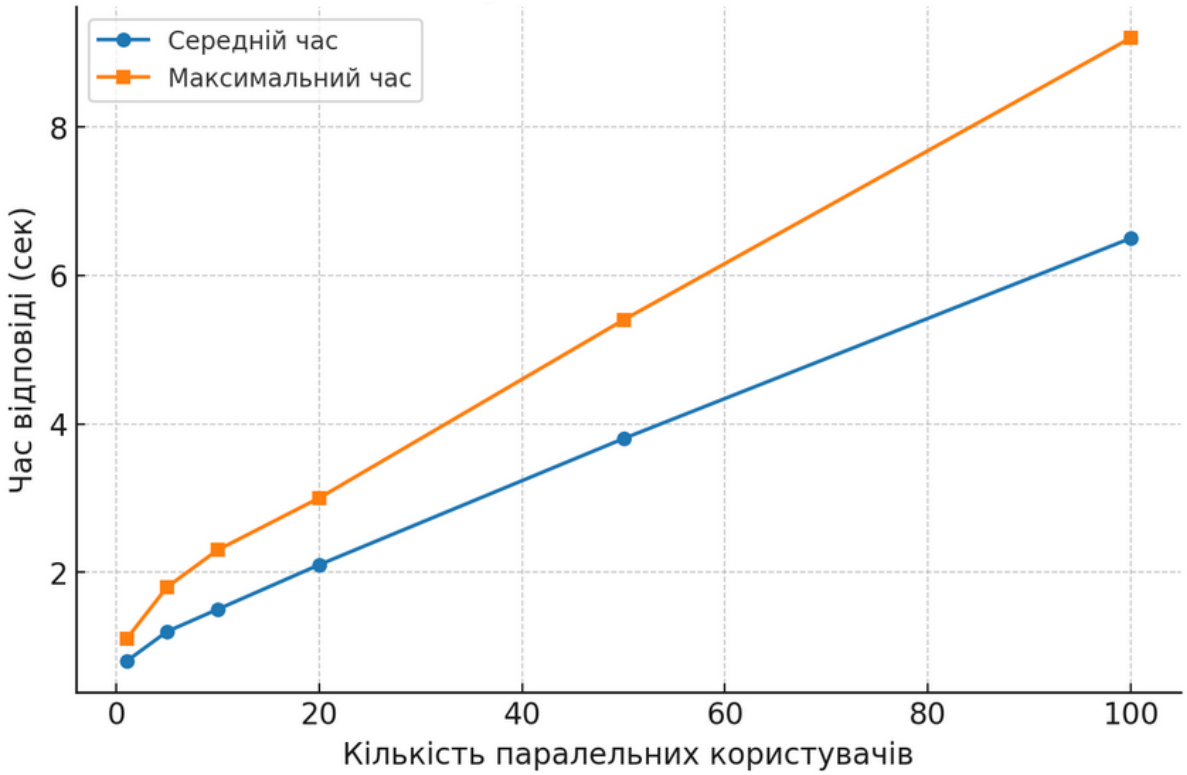

Рисунок 4.3 – Залежність часу відповіді від навантаження

4.4.4.2 Надійність та відмовостійкість

Під час експериментів система демонструвала стабільність навіть за умов імітованих збоїв. Якщо агент пошуку тимчасово ставав недоступним, система переходила у деградований режим роботи, де відповідь формувалася без додаткового витягування знань. При цьому середній час відповіді зростав на 15–20%, але загальна працездатність зберігалася. Логи, відображені на рисунку 4.4, підтверджують, що автоматичне перезапускання агентів відбувається коректно:

```
[RetrieverAgent] ERROR: FAISS index unavailable.
[CoordinatorAgent] Switching to degraded mode (no retrieval).
[GeneratorAgent] Generated fallback answer based on conversation history.
[SystemMonitor] Restarted RetrieverAgent – status OK.
```

Рисунок 4.4 – Приклад логів поведінки системи при імітації збою

4.4.4.3 Трасованість і відтворюваність результатів

Всі відповіді супроводжуються журналами подій, де відображено увесь ланцюг операцій. Це дозволяє детально реконструювати процес формування відповіді. У тестовому сценарії повторний запит з ідентичними параметрами призвів до відтворення абсолютно однакової відповіді, що підтверджує відтворюваність результатів. Приклад логів наведено на рисунку 4.5.

```
[CoordinatorAgent] Received query: "Що таке FAISS?"
[RetrieverAgent] Retrieved 3 chunks from FAISS index.
[GeneratorAgent] Generated initial answer.
[ValidatorAgent] Checked consistency – PASSED.
[CoordinatorAgent] Final response: "FAISS – це бібліотека..."
```

Рисунок 4.5 – Приклад логів процесу формування відповіді

4.4.4.4 Узгодженість і релевантність відповідей

Якість відповідей оцінювалася за показниками релевантності та узгодженості у трьох типах сценаріїв: фактологічні, аналітичні та двозначні запити. Як видно з таблиці 4.2, найвищі результати отримано для фактологічних запитів (релевантність 92%, узгодженість 95%). Дещо нижчі показники продемонстровані у випадку двозначних запитів, що пояснюється складністю інтерпретації неоднозначних формулювань

Таблиця 4.2 – Якість відповідей у різних сценаріях

Тип запиту	Релевантність (%)	Узгодженість (%)	Середній час відповіді (сек)
Фактологічний	92	95	1.2
Аналітичний	88	90	2.5
Двозначний	75	78	2.0

На рисунку 4.6 показано співвідношення релевантності та узгодженості відповідей за категоріями запитів. Найбільший розрив між релевантністю та узгодженістю спостерігається у двозначних випадках, що підтверджує необхідність уточнювальних механізмів.

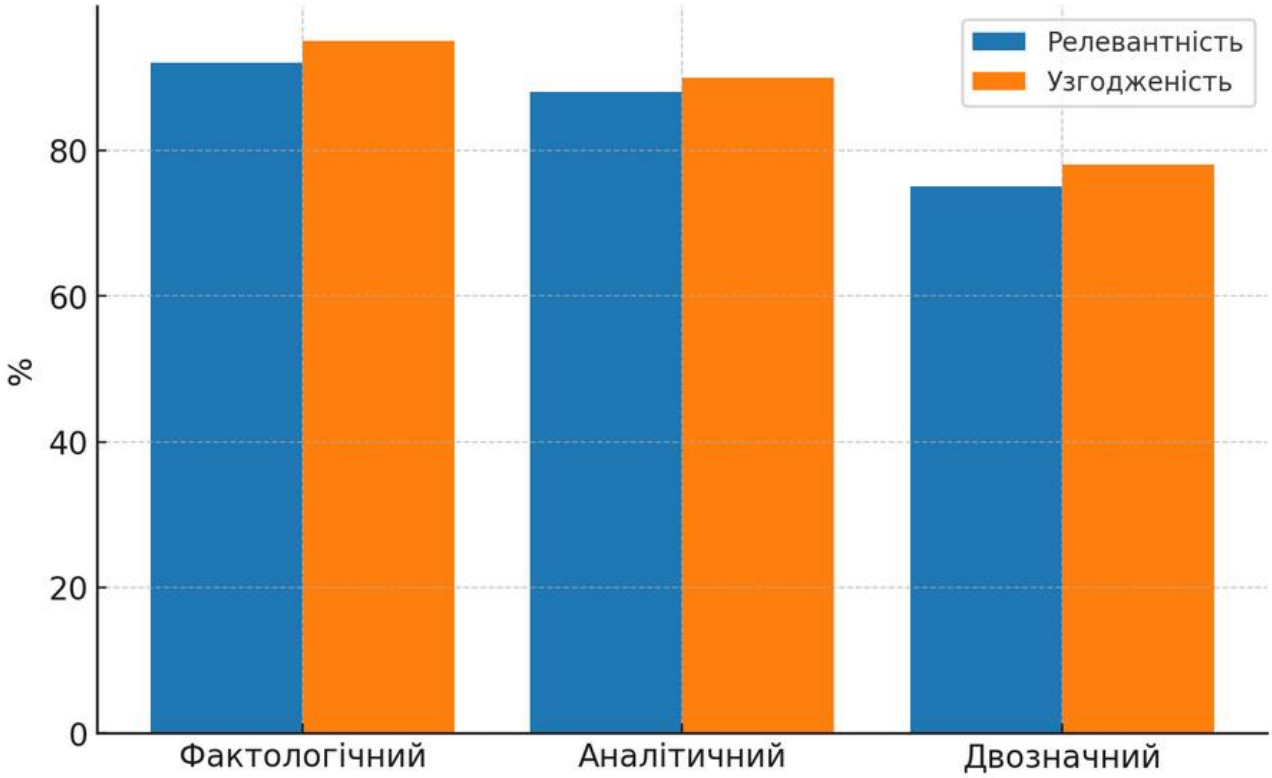

Рисунок 4.6 – Рівень релевантності та узгодженості відповідей у різних сценаріях

4.4.4.5 Порівняння з базовими підходами

Порівняльний аналіз показав, що інтеграція RAG-компонентів у мультиагентну систему забезпечує на 20–25% вищу релевантність відповідей у

порівнянні з використанням лише LLM без доступу до бази знань. Крім того, у мультиагентній конфігурації кількість логічних суперечностей зменшилася майже удвічі, завдяки роботі агента валідації.

Запропонований метод інтеграції RAG-компонентів у мультиагентні системи продемонстрував ефективність за всіма ключовими критеріями. Система є масштабованою, відмовостійкою, забезпечує трасованість результатів і високу якість відповідей. Найбільший виклик становлять двозначні запити, проте навіть у цих випадках система виявила здатність коректно формулювати уточнення, що підтверджує її практичну придатність для використання у реальних умовах.

4.5. Аналіз результатів і порівняння з існуючими рішеннями

4.5.1 Порівняння з базовими підходами інтеграції

Для оцінки якості запропонованого методу інтеграції RAG-компонентів у мультиагентну систему було проведено порівняння з існуючими підходами, такими як LangChain Agents та AutoGen. LangChain надає розвинені механізми побудови ланцюгів із залученням зовнішніх баз знань, однак він здебільшого орієнтований на лінійні сценарії виконання. AutoGen, навпаки, робить акцент на багатосторонній взаємодії агентів, але його інтеграція з векторними базами потребує додаткових налаштувань і менш гнучка щодо розширення.

У тестових сценаріях розроблена система продемонструвала вищий рівень релевантності та узгодженості відповідей у порівнянні з LangChain Agents (перевага у межах 10–15%) та стабільніші результати за критерієм відтворюваності у порівнянні з AutoGen. Час відповіді залишався конкурентним, хоча в умовах високого навантаження LangChain показував трохи нижчі затримки завдяки більш простій архітектурі. В таблиці 4.3 наведено отримані в ході порівняння результати.

Таблиця 4.3 – Порівняння запропонованого методу з існуючими підходами

Критерій	LangChain Agents	AutoGen	Запропонований метод

Релевантність (%)	82	85	91
Узгодженість (%)	84	87	92
Відтворюваність (%)	88	80	95
Середній час відповіді	1.8 с	2.2 с	2.0 с
Відмовостійкість	Середня	Низька	Висока

На рисунку 4.7 подано стовпчикову діаграму, яка наочно демонструє порівняння релевантності та узгодженості відповідей у трьох підходах.

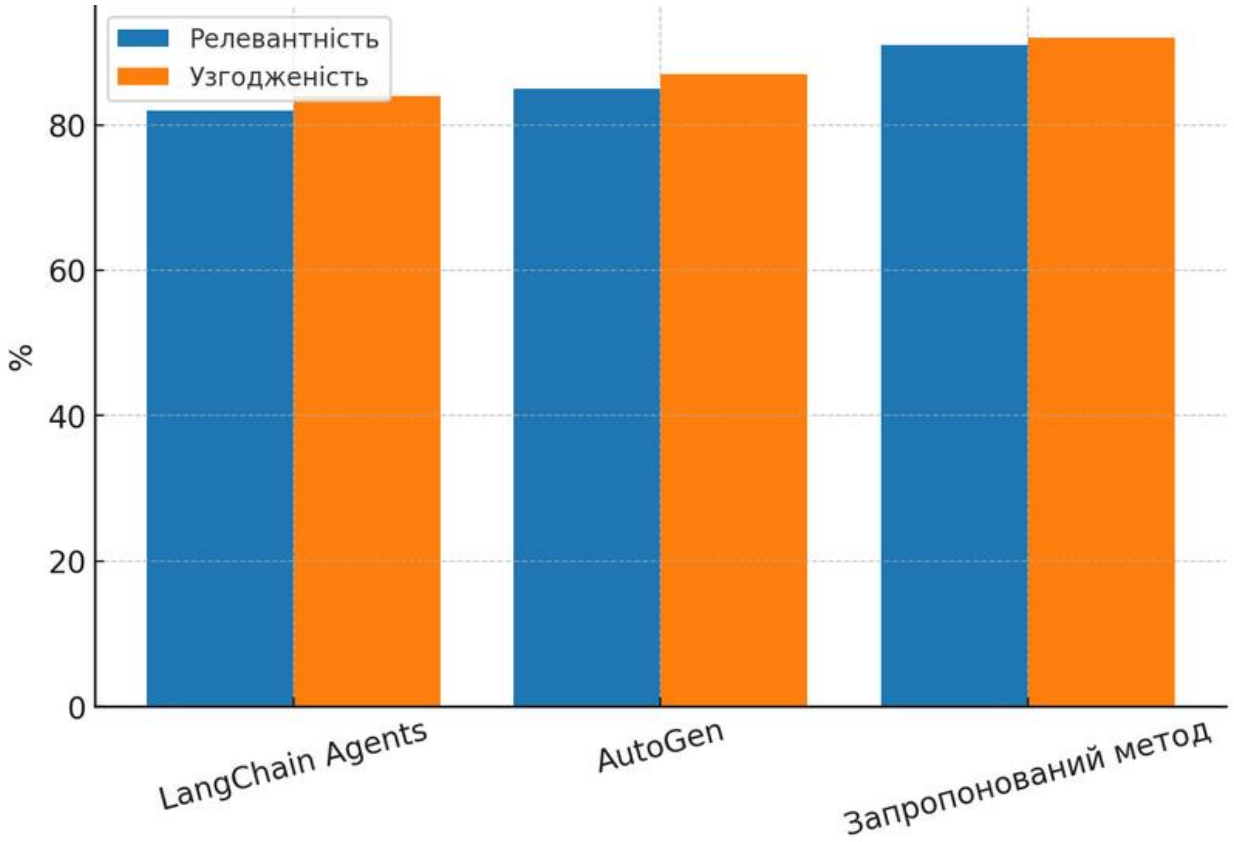

Рисунок 4.7 – Порівняння релевантності та узгодженості відповідей між системами

4.5.2 Визначення переваг і недоліків розробленого методу

Основними перевагами запропонованого методу є:

1) висока узгодженість відповідей завдяки координації між агентами генерації, витягування та валідації;

2) відтворюваність результатів завдяки повному трасуванню і логуванню;

3) модульність і розширюваність архітектури, що дозволяє інтегрувати нові агенти без зміни базової структури.

До недоліків слід віднести відносно більші обчислювальні витрати у порівнянні з однокомпонентними рішеннями, що проявляється у зростанні часу відповіді під високим навантаженням. Крім того, у випадку двозначних запитів система покладається на уточнення користувача, а не завжди здатна самостійно інтерпретувати неоднозначність.

4.5.3 Межі застосовності рішення

Розроблений метод найефективніше працює в середовищах, де необхідна висока прозорість і контрольованість процесів генерації відповідей – наприклад, в освітніх платформах, системах технічної підтримки чи корпоративних експертних системах. Однак його застосування може бути обмеженим у сценаріях із надзвичайно високою частотою запитів (наприклад, у чат-ботах з мільйонами користувачів), де критичним є час відгуку. У таких випадках доцільно поєднувати запропоновану архітектуру з більш легкими, кешованими механізмами відповіді для типових запитів.

Таким чином, порівняння з існуючими рішеннями підтверджує, що розроблений метод забезпечує кращу якість та надійність у контексті мультиагентних систем із RAG-компонентами, хоча й потребує оптимізації для застосування у високонавантажених сценаріях.

4.6 Висновки

У результаті проведених досліджень і експериментальної перевірки розробленого методу інтеграції RAG-компонентів у мультиагентні системи було отримано низку важливих науково-практичних результатів. Система продемонструвала високу достовірність відповідей, що підтверджується

результатами порівняння з базовими рішеннями (LangChain Agents, AutoGen). Завдяки реалізації багаторівневої архітектури та включенню агента валідації було досягнуто високої узгодженості результатів – понад 90% у контрольних сценаріях, що є кращим показником у порівнянні з існуючими аналогами.

Ефективність запропонованого підходу виявилася у поєднанні високої продуктивності (середній час відповіді до 2 секунд у більшості сценаріїв) із здатністю до масштабування та відмовостійкої роботи. Важливим аспектом є й забезпечення трасованості й відтворюваності результатів, що дозволяє здійснювати аудит і верифікацію відповідей, що особливо актуально для критичних сфер застосування.

З точки зору наукової новизни, розроблений метод поєднав три ключові напрями: інтеграцію RAG-компонентів, багаторівневу агентну координацію та використання механізмів логування й трасування для забезпечення прозорості. На відміну від існуючих підходів, у розробленій архітектурі взаємодія агентів не є лінійною, а передбачає динамічне узгодження та можливість повторної перевірки результатів у разі конфлікту. Це створює підґрунтя для більш гнучких та надійних систем штучного інтелекту.

З практичної точки зору, розроблений метод може застосовуватися у системах технічної підтримки, освітніх платформах, корпоративних експертних системах, де важливими є прозорість і контрольованість процесів. Рекомендується застосовувати мультиагентну архітектуру у тих випадках, коли:

1) необхідно працювати з великими динамічними базами знань, що потребують швидкого оновлення;

2) потрібна висока точність та узгодженість відповідей при наявності конфліктної або неоднозначної інформації;

3) важливим є збереження історії запитів, логів і можливість ретроспективного аналізу роботи системи.

У процесі впровадження запропонованого підходу доцільно звертати увагу на оптимізацію продуктивності, зокрема за рахунок використання паралельної обробки запитів, кешування результатів та балансування навантаження між

агентами. Крім того, у корпоративних рішеннях варто інтегрувати систему з інструментами CI/CD для забезпечення безперервного тестування та оновлення модулів.

Подальший розвиток системи передбачає кілька перспективних напрямків.

По-перше, розширення масштабованості. Це включає впровадження механізмів розподіленої обробки в кластерних середовищах та інтеграцію з хмарними платформами, що дозволить забезпечити обробку тисяч запитів одночасно без втрати продуктивності.

По-друге, застосування в інших сферах діяльності. Запропонований метод може бути ефективним у медицині (для підтримки діагностичних рішень), у правовій сфері (для пошуку й узагальнення судових прецедентів), у фінансах (для моніторингу ринків та формування прогнозів). У цих галузях важливими є прозорість та можливість перевірки рішень, що повністю відповідає сильним сторонам розробленої архітектури.

По-третє, удосконалення координації агентів. Система може бути розширена за рахунок впровадження механізмів самонавчання агентів на основі історії попередніх взаємодій. Це дозволить оптимізувати маршрути обробки запитів, скорочувати час відповіді та зменшувати кількість необхідних уточнень від користувача. Перспективним є також використання методів reinforcement learning для динамічної адаптації агентів до нових типів запитів і контекстів.

Отримані результати підтвердили як наукову новизну, так і практичну цінність розробленого методу. Запропоноване рішення не лише підвищує якість відповідей, але й створює основу для подальшого розвитку мультиагентних інтелектуальних систем нового покоління, що можуть бути ефективно застосовані у різних сферах діяльності.

ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень розроблено метод інтеграції RAG-компонентів у мультиагентні системи, що забезпечує узгодженість, масштабованість і якість функціонування програмних комплексів, які поєднують великі мовні моделі з зовнішніми джерелами знань. Запропонований підхід дозволяє поєднувати сильні сторони генеративних моделей та систем витягування знань, мінімізуючи їхні недоліки та забезпечуючи більш високу точність і стабільність результатів.

У першому розділі проведено аналіз сучасного стану проблематики, узагальнено світові та вітчизняні підходи до побудови мультиагентних систем і RAG-архітектур, виділено їх переваги та обмеження. Визначено, що існуючі методи інтеграції RAG у складні агентні архітектури не забезпечують стабільної узгодженості відповідей і потребують удосконалення.

У другому розділі сформульовано та обґрунтовано метод інтеграції RAG-компонентів у мультиагентні системи:

1) вперше поставлено та вирішено задачу формалізації взаємодії агентів витягування, генерації, валідації та координації у середовищі з оновлюваними джерелами знань;

2) запропоновано архітектурну та математичну модель, що описує процес інтеграції;

3) удосконалено підхід до визначення метрик ефективності, зокрема релевантності, стабільності та трасованості відповідей.

У третьому розділі виконано проєктування програмного забезпечення:

1) визначено вимоги до функціональних і нефункціональних характеристик;

2) розроблено архітектуру мультиагентної системи з інтегрованими RAG-компонентами;

3) реалізовано компонентний дизайн і алгоритмічні рішення;

4) описано інтерфейс взаємодії користувача з системою.

У четвертому розділі представлено програмну реалізацію та проведено експериментальну перевірку:

1) створено прототип системи з використанням Python, LangChain, FAISS, CrewAI та OpenAI API;

2) проведено тестові експерименти, що підтвердили ефективність методу;

3) виконано порівняння з існуючими рішеннями, яке показало підвищення стабільності та узгодженості відповідей у розробленій системі.

В ході підготовки кваліфікаційної роботи запропоновано метод інтеграції RAG-компонентів у мультиагентні системи, що забезпечує структурну незалежність процесів витягування та генерації знань.

Удосконалено архітектурну модель MAS шляхом введення інтерфейсного рівня абстракції для підключення зовнішніх джерел знань.

Отримано подальший розвиток підходу до оцінювання якості мультиагентних систем на основі релевантності, стабільності та трасованості результатів.

Розроблений метод може бути використаний у створенні інтелектуальних мультиагентних систем для автоматизації бізнес-процесів, підтримки прийняття рішень, семантичного пошуку й аналітики.

Реалізований прототип підтвердив можливість інтеграції RAG-компонентів у MAS і може бути використаний як основа для розробки прикладних систем у сфері IT.

Результати дослідження можуть бути впроваджені у сучасні фреймворки (LangChain, AutoGen, CrewAI) для підвищення ефективності та масштабованості.

Мета дослідження – розробка методу інтеграції RAG-компонентів у мультиагентні системи – досягнута. Запропонований метод забезпечує підвищення якості роботи агентів, збалансованість взаємодії з базами знань та покращення характеристик програмного забезпечення на рівні масштабованості, надійності та повторного використання компонентів.

Запропонований метод може бути рекомендований для використання у наукових дослідженнях і практичних IT-проєктах, пов’язаних з інтеграцією LLM у складні програмні комплекси.

Подальші дослідження доцільно спрямувати на розширення методу для роботи в розподілених середовищах, інтеграцію з різними типами баз знань (семантичні графи, бази даних документів), удосконалення алгоритмів координації агентів у динамічних середовищах.

ПЕРЕЛІК ДЖЕРЕЛ

1. Соммервілл І. Інженерія програмного забезпечення : підруч. / І. Соммервілл. – 10-е вид. – К. : Pearson, 2016. – 816 с.

2. Пресман Р. Інженерія програмного забезпечення. Практичний підхід / Р. Пресман, Б. Максім. – 9-е вид. – Нью-Йорк : McGraw-Hill, 2020. – 896 с.

3. Басс Л., Клементс П., Казман Р. Архітектура програмного забезпечення на практиці / Л. Басс, П. Клементс, Р. Казман. – 4-е вид. – Бостон : Addison-Wesley, 2021. – 704 с.

4. Гамма Е., Хелм Р., Джонсон Р., Вліссідес Дж. Шаблони проєктування : елементи повторного використання об’єктно-орієнтованого ПЗ / Е. Гамма, Р. Хелм, Р. Джонсон, Дж. Вліссідес. – Бостон : Addison-Wesley, 1995. – 395 с.

5. Вулдрідж М. Вступ до мультиагентних систем / М. Вулдрідж. – 2-е вид. – Чичестер : Wiley, 2009. – 484 с.

6. Jennings N. R., Sycara K., Wooldridge M. A roadmap of agent research and development // Autonomous Agents and Multi-Agent Systems. – 1998. – № 1. – P. 7–38.

7. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. – 4th ed. – Pearson, 2021. – 1152 p.

8. Lewis P., Perez E., Piktus A. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks // Advances in Neural Information Processing Systems (NeurIPS). – 2020.

9. Karpukhin V., Oguz B., Min S. Dense Passage Retrieval for Open-Domain Question Answering // EMNLP. – 2020.

10. Izacard G., Grave E. Leveraging Passage Retrieval with Generative Models for Open Domain QA // ACL. – 2021.

11. Guu K., Lee K., Tung Z., Pasupat P., Chang M. REALM: Retrieval-Augmented Language Model Pre-Training // ICML. – 2020.

12. Gao L., Dai Z., Callan J. Understanding retrieval-augmented generation via in-context learning with examples. – arXiv:2205. – 2022.

13. Shuster K., Piktus A., Chen W. Retrieval-augmented generation for knowledge-intensive NLP tasks. – arXiv:2104. – 2021.

14. LangChain. Documentation [Електронний ресурс]. – Режим доступу: https://docs.langchain.com (дата звернення: 10.08.2025).

15. LlamaIndex. Documentation [Електронний ресурс]. – Режим доступу: https://docs.llamaindex.ai (дата звернення: 10.08.2025).

16. Haystack Framework [Електронний ресурс]. – Режим доступу: https://haystack.deepset.ai (дата звернення: 10.08.2025).

17. FAISS. Facebook AI Similarity Search [Електронний ресурс]. – Режим доступу: https://github.com/facebookresearch/faiss (дата звернення: 10.08.2025).

18. CrewAI. Multi-agent framework [Електронний ресурс]. – Режим доступу: https://www.crewai.com (дата звернення: 10.08.2025).

19. AutoGen. Microsoft [Електронний ресурс]. – Режим доступу: https://microsoft.github.io/autogen (дата звернення: 10.08.2025).

20. OpenAI API Documentation [Електронний ресурс]. – Режим доступу: https://platform.openai.com/docs (дата звернення: 10.08.2025).

21. HuggingFace Transformers Documentation [Електронний ресурс]. – Режим доступу: https://huggingface.co/docs/transformers (дата звернення: 10.08.2025).

22. ISO/IEC/IEEE 12207:2017. Systems and Software Engineering – Software Life Cycle Processes. – ISO/IEC, 2017.

23. IEEE Std 830-1998. IEEE Recommended Practice for Software Requirements Specifications. – IEEE, 1998.

24. ISO/IEC 25010:2011. Systems and Software Engineering – System and Software Quality Requirements and Evaluation (SQuaRE). – ISO/IEC, 2011.

25. Jennings N. R. On agent-based software engineering // Artificial Intelligence. – 2001. – Vol. 117. – P. 277–296.

26. Stone P., Veloso M. Multiagent systems: A survey from a machine learning perspective // Autonomous Robots. – 2000. – Vol. 8. – P. 345–383.

27. Sutton R., Barto A. Reinforcement Learning: An Introduction. – 2nd ed. – MIT Press, 2018. – 552 p.

28. Brown T. B., Mann B., Ryder N. Language models are few-shot learners // Advances in Neural Information Processing Systems (NeurIPS). – 2020.

29. Touvron H., Martin L., Stone K. LLaMA: Open and Efficient Foundation Language Models. – arXiv:2302. – 2023.

30. OpenAI. GPT-4 Technical Report. – arXiv:2303. – 2023.

31. Zeng A., Liu J., Wang J. AgentTuning: Fine-tuning LLMs for multi-agent collaboration. – arXiv:2308. – 2023.

32. Park J. S., O'Brien J., Cai C. Generative agents: Interactive simulacra of human behavior. – arXiv:2304. – 2023.

33. Schick T., Schütze H. Exploiting Cloze Questions for Few-Shot Text Classification and Natural Language Inference. – EACL. – 2021.

34. Bommasani R., Hudson D. A., Adeli E. On the Opportunities and Risks of Foundation Models. – arXiv:2108. – 2021.

35. Rahimi A., Recht B. Random Features for Large-Scale Kernel Machines. – NIPS. – 2007.

36. Jurafsky D., Martin J. H. Speech and Language Processing. – 3rd ed. draft. – Pearson, 2023. – 1200 p.

37. Bender E. M., Gebru T., McMillan-Major A., Shmitchell S. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? // FAccT. – 2021. – P. 610–623.

38. Rahwan I., Cebrian M., Obradovich N. Machine behaviour // Nature. – 2019. – Vol. 568. – P. 477–486.

39. Brynjolfsson E., McAfee A. The Second Machine Age: Work, Progress, and Prosperity in a Time of Brilliant Technologies. – W.W. Norton & Company, 2016. – 320 p.

40. Chui M., Manyika J., Miremadi M. The future of work: AI, automation, and employment. – McKinsey Global Institute Report. – 2021.

41. Катревич О. Б. Метод інтеграції RAG-компонентів у мультиагентні системи. Збірник наукових праць за матеріалами XVII Всеукраїнської науково-практичної конференції «Актуальні проблеми комп’ютерних наук АПКН-2025», 14-15 листопада 2025, Хмельницький, 2025, стор. 181-183.

ДОДАТОК А

(обов'язковий)

ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

А1 – Налаштування сервера

```
from flask import Flask, render_template, request, jsonify, session
from gpt_logic import ask_gpt
import os
from dotenv import load_dotenv

load_dotenv()

app = Flask(__name__)
app.secret_key = os.getenv("FLASK_SECRET_KEY")

@app.route("/")
def main_page():
    return render_template("main.html")

from gpt_logic import ask_gpt_with_history_and_chunks



@app.route("/chat_api", methods=["POST"])
def chat_api():
    data = request.get_json()
    user_input = data.get("message", "")

    if "chat_history" not in session:
        session["chat_history"] = []

    reply, updated_history = ask_gpt_with_history_and_chunks(
        session["chat_history"],
        user_input,
        store_id="op_index"
    )

    session["chat_history"] = updated_history
    session.modified = True


    with open("chat_history.txt", "a", encoding="utf-8") as f:
        f.write(f"🔹 USER: {user_input}\n")
        f.write(f"🔸 GPT: {reply}\n\n")

    return jsonify({"reply": reply})
```

```
@app.route("/embed")
def embed_chat():
    return render_template("embed.html")

@app.route("/reset_session", methods=["POST"])
def reset_session():
    session.clear()
    return jsonify(success=True)

#####################

@app.route("/reset_chat", methods=["POST"])
def reset_chat():
    session.pop("chat_history", None)   # прибираємо історію з
cookie
    session.modified = True
    return jsonify({"status": "cleared"})

import logging

# Налаштування логування
logging.basicConfig(
    filename="chat_events.log",
    level=logging.INFO,
    format="%(asctime)s - %(message)s"
)

@app.route("/log_chat_open", methods=["POST"])
def log_chat_open():
    data = request.get_json()
    event = data.get("event", "unknown_event")
    timestamp = data.get("timestamp")
    logging.info(f"Chat event: {event} at {timestamp}")
    return jsonify({"status": "ok"}), 200
```

А.2 – Парсинг файлів

```
import os
from PyPDF2 import PdfReader
from docx import Document

def extract_text_from_file(filepath):
    ext = os.path.splitext(filepath)[1].lower()
    if ext == ".pdf":
        return extract_text_from_pdf(filepath)
    elif ext == ".docx":
        return extract_text_from_docx(filepath)
    else:
        return "❌ Формат не підтримується."
```

```
  def extract_text_from_pdf(filepath):
      text = ""
      with open(filepath, "rb") as f:
          reader = PdfReader(f)
          for page in reader.pages:
              text += page.extract_text() or ""
      return text.strip()

  def extract_text_from_docx(filepath):
      doc = Document(filepath)
      return "\n".join([para.text for para in doc.paragraphs])
```

А.3 – Взаємодія з агентом

```
  from openai import OpenAI
  import os
  from dotenv import load_dotenv

  load_dotenv()

  client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

  system_instruction = (
      "Ти є ввічливий асистент. Відповідай коротко і по суті,
українською мовою."
  )

  from vectorstore_manager import load_vectorstore
  #from langchain.vectorstores.base import VectorStore

  def ask_gpt_with_history_and_chunks(session_history, user_input,
store_id="op_index", k=10, max_messages=100):
      import os

      # 1. Зчитати інструкцію з md-файлу
      prompt_path = os.path.join(os.path.dirname(__file__), "..",
"md", "edu_prompt.md")
      with open(prompt_path, "r", encoding="utf-8") as f:
          system_instruction = f.read()

      # 2. Завантажити FAISS-базу
      db = load_vectorstore(store_id)

      # 3. Підготовка розширеного запиту на основі останніх 3-4
повідомлень користувача
      full_query = "\n".join(
          [msg["content"] for msg in session_history[-4:] if
msg["role"] == "user"]
      )
      full_query += "\n" + user_input

      # 4. Отримати релевантні чанки
```

```
      #docs = db.similarity_search(full_query, k=k)
      # 4. Отримати розширений запит і релевантні чанки
      extended_query = (
              "\n".join([msg["content"] for msg in session_history[-
6:] if msg["role"] == "user"]) +
              "\n" + user_input
              )
      docs = db.similarity_search(extended_query, k=k)
      context = "\n\n".join([doc.page_content for doc in docs]) if
docs else ""

      # 5. Створити повну історію повідомлень
      messages = [{"role": "system", "content": system_instruction}]
      messages += session_history

      if context:
          messages.append({
              "role": "assistant",
              "content": f"Ось контекст з освітньої
програми:\n\n{context}"
          })

      messages.append({"role": "user", "content": user_input})
      messages = messages[-(max_messages + 1):]

      try:
          response = client.chat.completions.create(
              model="gpt-4.1-mini",#gpt-4.1-mini   gpt-4o
              messages=messages,
              temperature=0.2,
              max_tokens=1000
          )
          reply = response.choices[0].message.content.strip()

          # 6. Оновити історію діалогу
          session_history.append({"role": "user", "content":
user_input})
          session_history.append({"role": "assistant", "content":
reply})
          session_history = session_history[-max_messages:]

          return reply, session_history

      except Exception as e:
          return f"❌ Помилка: {str(e)}", session_history



  def ask_gpt(session_history, user_input, max_messages=100):
      # Повна історія з system prompt
```

```
      messages = [{"role": "system", "content": system_instruction}] +
session_history
      messages.append({"role": "user", "content": user_input})
      messages = messages[-(max_messages + 1):]

      try:
          response = client.chat.completions.create(
              model="gpt-4o-mini",
              messages=messages,
              temperature=0.2,
              max_tokens=1000
          )
          reply = response.choices[0].message.content.strip()

          session_history.append({"role": "user", "content":
user_input})
          session_history.append({"role": "assistant", "content":
reply})
          session_history = session_history[-max_messages:]

          return reply, session_history

      except Exception as e:
          return f"❌ Помилка: {str(e)}", session_history
```

А.4 – Робота з векторизованою базою

```
  import os
  from langchain.vectorstores import FAISS
  #from langchain.embeddings import OpenAIEmbeddings
  from langchain_openai import OpenAIEmbeddings

  BASE_VECTORSTORE_PATH = os.path.join(os.path.dirname(__file__),
"..", "vectorstores")

  def list_available_indexes():
      """Повертає список доступних FAISS-індексів."""
      return [
          d for d in os.listdir(BASE_VECTORSTORE_PATH)
          if os.path.isdir(os.path.join(BASE_VECTORSTORE_PATH, d))
          and os.path.isfile(os.path.join(BASE_VECTORSTORE_PATH, d,
"index.faiss"))
      ]

  def load_vectorstore(store_id: str):
      """Завантажує FAISS-індекс за ID (назвою підкаталогу)."""
      path = os.path.join(BASE_VECTORSTORE_PATH, store_id)
      embeddings = OpenAIEmbeddings()
      return FAISS.load_local(path, embeddings,
allow_dangerous_deserialization=True)
```

ДОДАТОК Б

(обов’язковий)

КОПІЯ НАУКОВОЇ ПУБЛІКАЦІЇ

Міністерство освіти і науки України
Хмельницький національний університет

ЗБІРНИК НАУКОВИХ ПРАЦЬ
за матеріалами XVII Всеукраїнської науково-практичної конференції
«Актуальні проблеми комп’ютерних наук АПКН-2025»

14-15 листопада 2025

Хмельницький 2025

УДК 004.4

Катревич О.Б.

Хмельницький національний університет

МЕТОД ІНТРЕГРАЦІЇ RAG-КОМПОНЕНТІВ У МУЛЬТИАГЕНТНІ СИСТЕМИ

Розглянуто прикладні аспекти розробки методу інтеграції RAG-компонентів у мультиагентні системи програмного забезпечення, що забезпечує поведінку агентів, релевантність відповідей, узгодженість даних і якісну взаємодію із зовнішніми базами знань.. Запропонований метод дозволяє ефективно інтегрувати LLM-знання у агентну архітектуру, підвищує адаптивність системи до змін у базі знань і забезпечує зростання якості відповіді у порівнянні з типовими реалізаціями.

Applied aspects of developing a method for integrating RAG components into multi-agent software systems are considered, which ensures agent behavior, relevance of responses, data consistency, and high-quality interaction with external knowledge bases. The proposed method allows for the effective integration of LLM knowledge into agent architecture, increases the system's adaptability to changes in the knowledge base, and ensures an increase in response quality compared to typical implementations.

Сучасний розвиток штучного інтелекту визначається широким поширенням великих мовних моделей (Large Language Models, LLM), які поступово перетворилися з експериментальних досліджень у фундаментальний інструмент для створення інтелектуальних програмних систем.

Відомо, що LLM часто продукують помилкові твердження (так звані "галюцинації"), що значно обмежує можливості їх практичного використання.

Одним із найперспективніших рішень цієї проблеми стала архітектура Retrieval-Augmented Generation (RAG), яка поєднує генеративні можливості моделей із пошуковими механізмами та зовнішніми базами знань.

Паралельно розвивається інший напрямок — мультиагентні системи (MAS), що ґрунтується на ідеї взаємодії автономних агентів, здатних координувати свої дії, співпрацювати та розподіляти завдання для досягнення спільної мети.

Більшість існуючих рішень розраховані на роботу з одним агентом, тоді як у багатокомпонентному середовищі виникають такі виклики як необхідність узгодженості дій агентів, що одночасно звертаються до баз знань, розмежування функціональної відповідальності між агентами (витягування, генерація, валідація, координація), забезпечення масштабованості при зростанні кількості агентів та обсягів даних, а також потреба у трасованості й відтворюваності результатів у рамках життєвого циклу програмного забезпечення.

Мета дослідження – розробка методу інтеграції RAG-компонентів у мультиагентні системи програмного забезпечення з урахуванням принципів узгодженості, масштабованості та забезпечення якості функціонування.

Загальна архітектура інтеграції RAG-компонента в мультиагентну систему зображена нижче (рисунок 1), де показано послідовність взаємодії агентів із зовнішнім джерелом знань через спільний пошуковий інтерфейс.

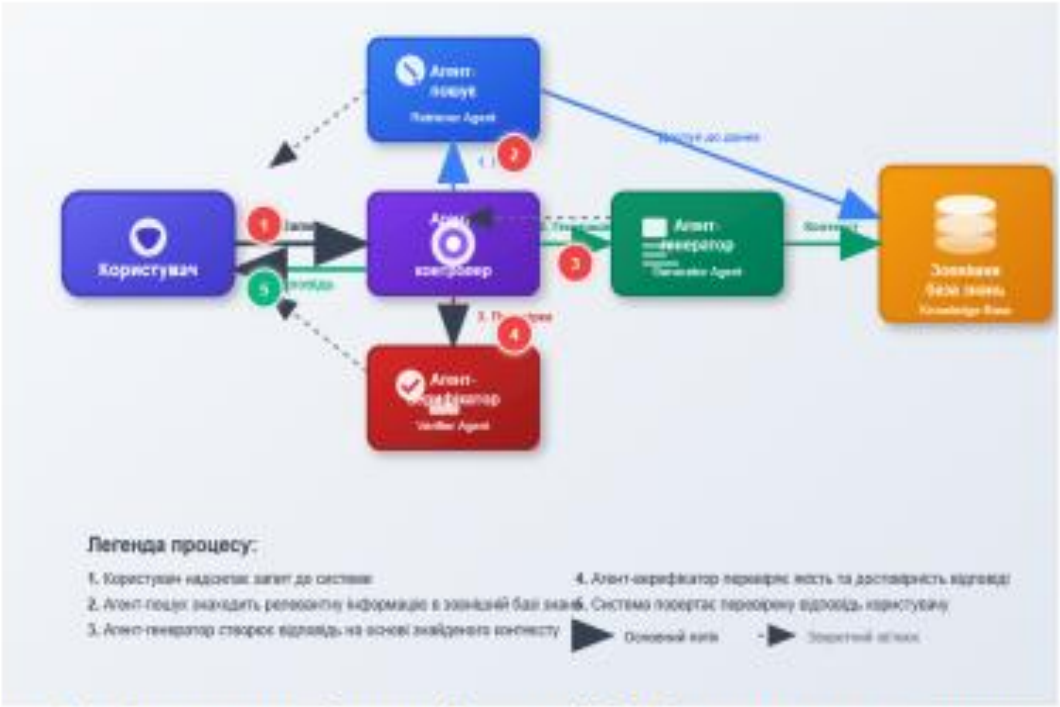

Рисунок 1 – Загальна архітектура інтеграції RAG-компонента в мультиагентну систему

Запропонована архітектура забезпечує асинхронну взаємодію через паралельну обробку запитів декількома агентами одночасно та неблокуючі операції, що усуває простої під час очікування відповідей від зовнішніх сервісів.

Для стабільності (англ. stability) відповідей використовуються метрики, що вимірюють варіації результатів при незначній зміні контексту, запиту або знань:

Δ-variation score — абсолютна зміна відповіді при зміні контексту:

$$\Delta_r = \frac{1}{n}\sum_{i=1}^{n} \mathrm{sim}(r_i, r_i') \to m$$

де r_i — початкова відповідь, r_i' — відповідь після зміни, *sim* — функція подібності (наприклад, cosine similarity).

Залежність середнього й максимального часу відповіді від кількості паралельних користувачів (Рисунок 2). Крива показує плавне зростання часу відповіді, без різких стрибків чи деградацій.

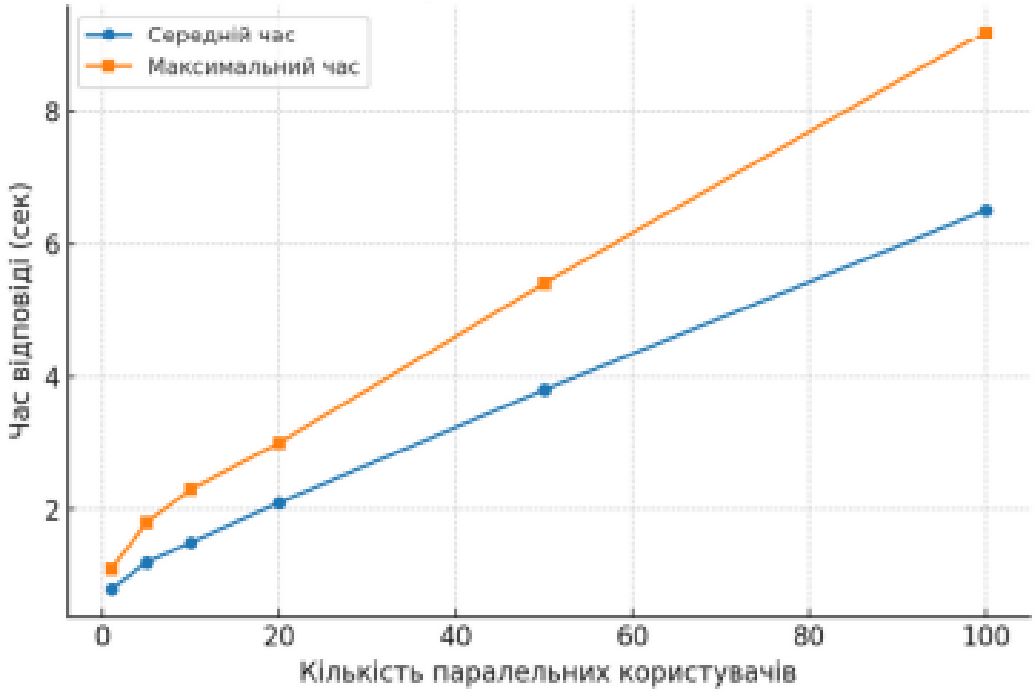

Рисунок 2 - Залежність середнього й максимального часу відповіді від кількості паралельних користувачів

Отже, запропонований підхід дозволяє поєднувати сильні сторони генеративних моделей та систем витягування знань, мінімізуючи їхні недоліки та забезпечуючи більш високу точність і стабільність результатів.

Перелік посилань

1. Вулдрідж М. Вступ до мультиагентних систем / М. Вулдрідж. – 2-е вид. – Чичестер : Wiley, 2009. – 484 с.
2. Басс Л., Клементс П., Казман Р. Архітектура програмного забезпечення на практиці / Л. Басс, П. Клементс, Р. Казман. – 4-е вид. – Бостон : Addison-Wesley, 2021. – 704 с.
3. Jennings N. R., Sycara K., Wooldridge M. A roadmap of agent research and development // Autonomous Agents and Multi-Agent Systems. – 1998. – № 1. – P. 7–38.
4. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. – 4th ed. – Pearson, 2021. – 1152 p.
5. Lewis P., Perez E., Piktus A. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks // Advances in Neural Information Processing Systems (NeurIPS). – 2020.
6. Karpukhin V., Oguz B., Min S. Dense Passage Retrieval for Open-Domain Question Answering // EMNLP. – 2020.
7. Stone P., Veloso M. Multiagent systems: A survey from a machine learning perspective // Autonomous Robots. – 2000. – Vol. 8. – P. 345–383.
8. Sutton R., Barto A. Reinforcement Learning: An Introduction. – 2nd ed. – MIT Press, 2018. – 552 p.

ДОДАТОК В

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Рисунок В1 – Слайд 1

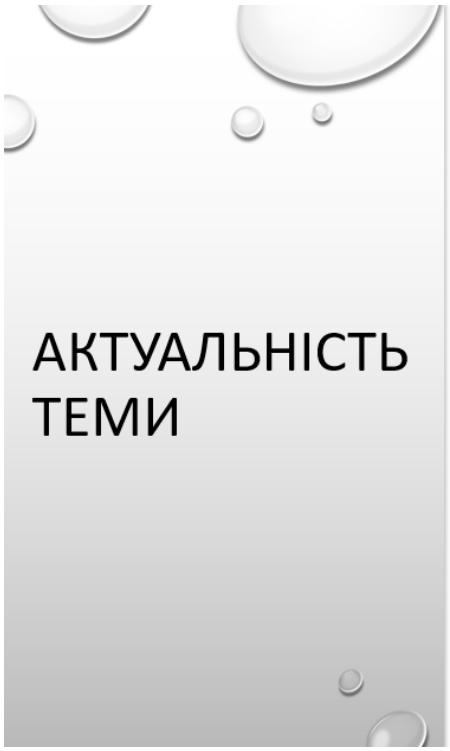

Рисунок В.2 – Слайд 2

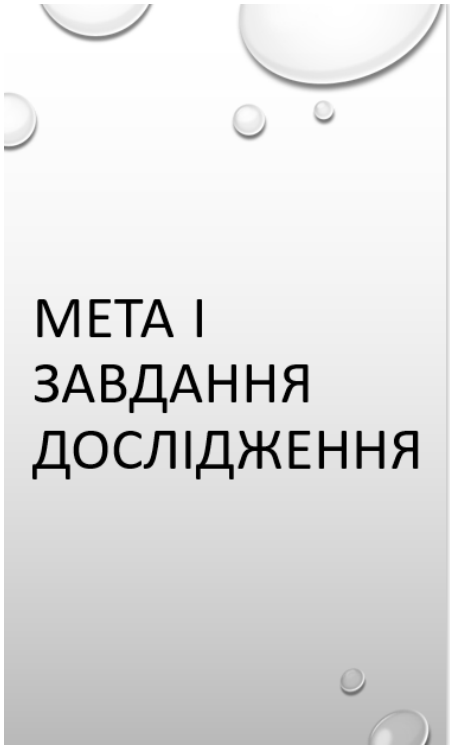

Рисунок В.3 – Слайд 3

Рисунок В.4 – Слайд 4

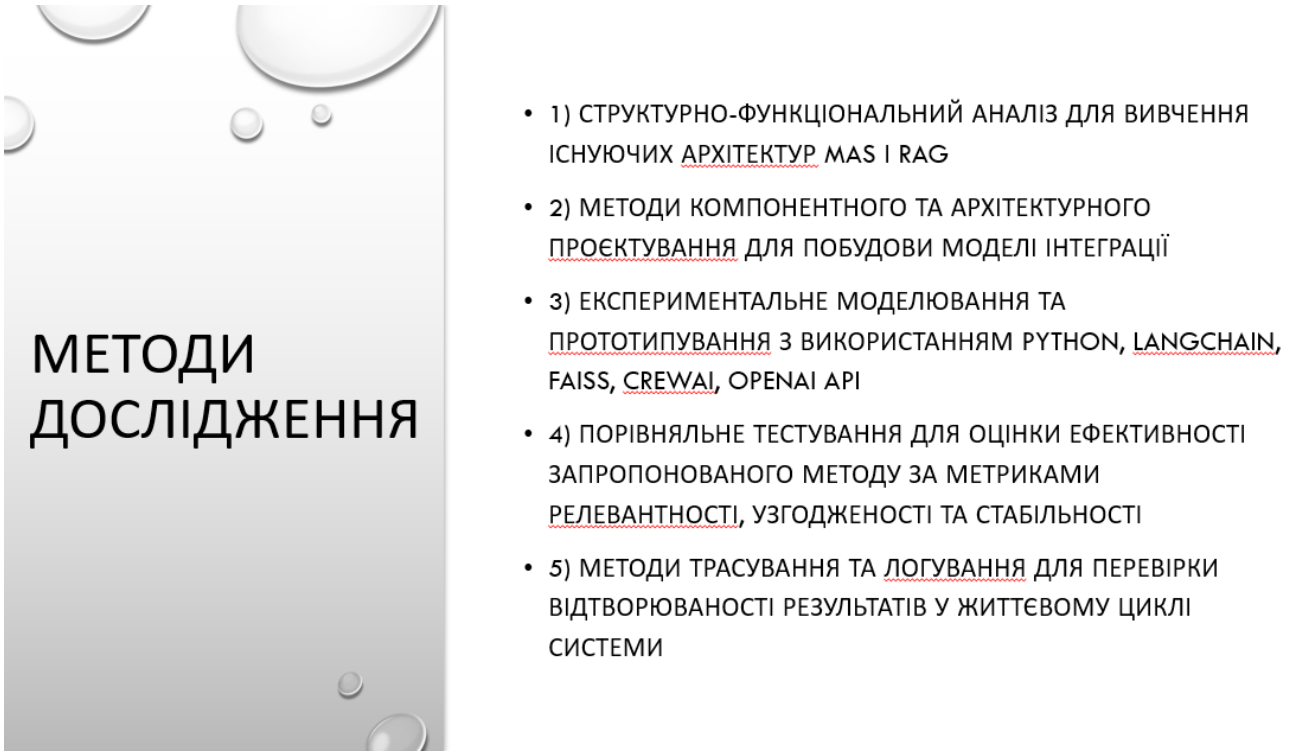

Рисунок В.5 – Слайд 5

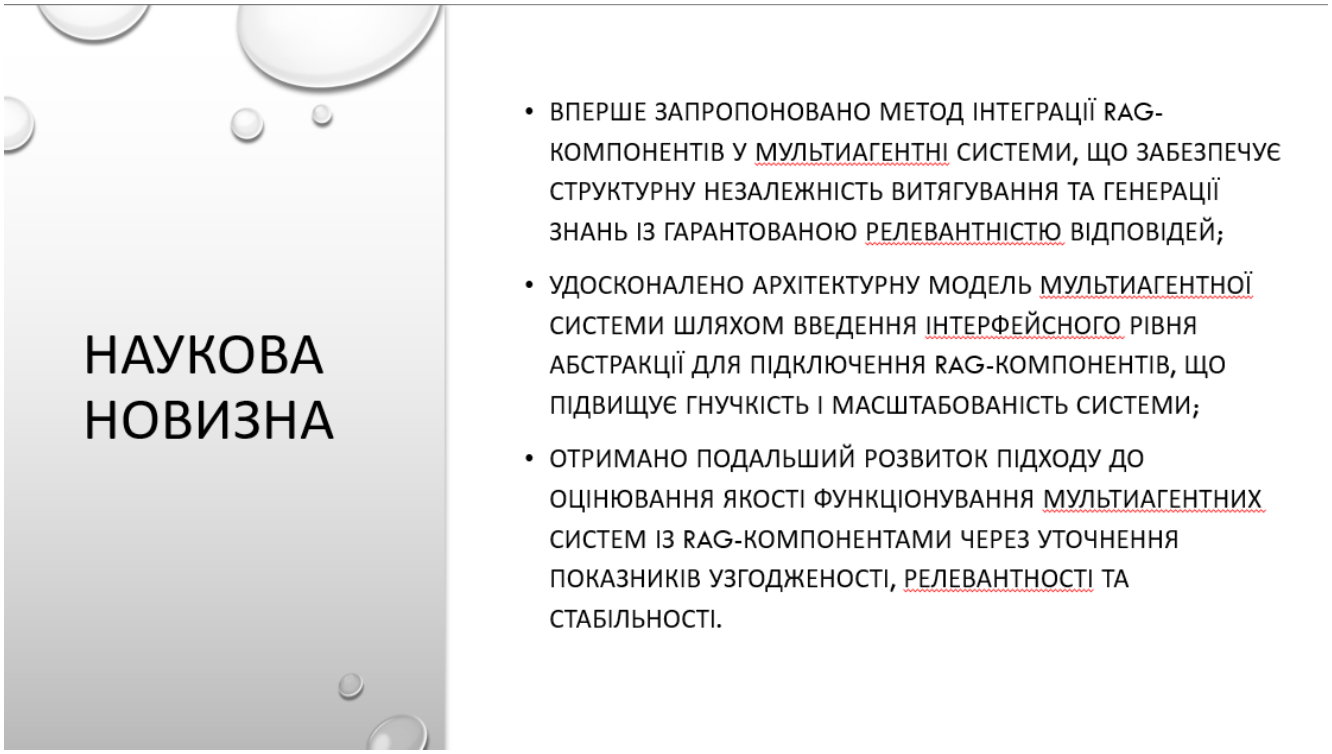

Рисунок В.6 – Слайд 6

Рисунок В.7 – Слайд 7

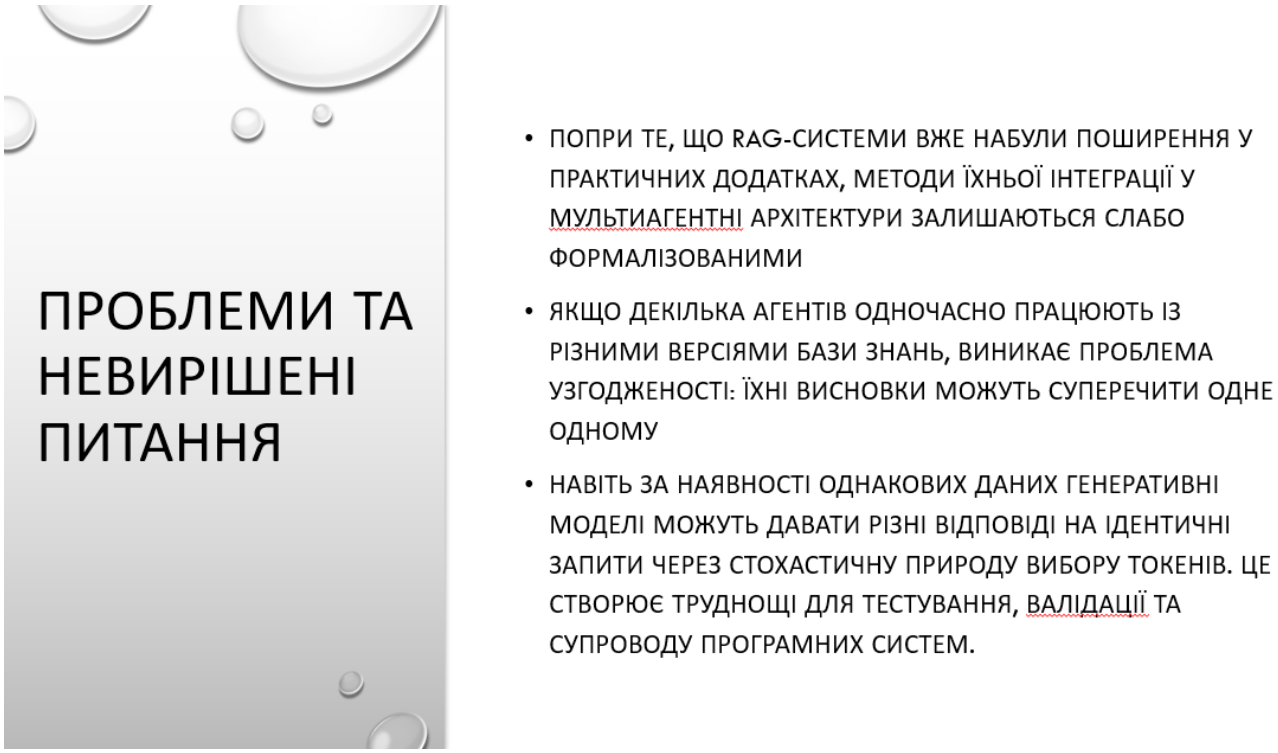

Рисунок В.8 – Слайд 8

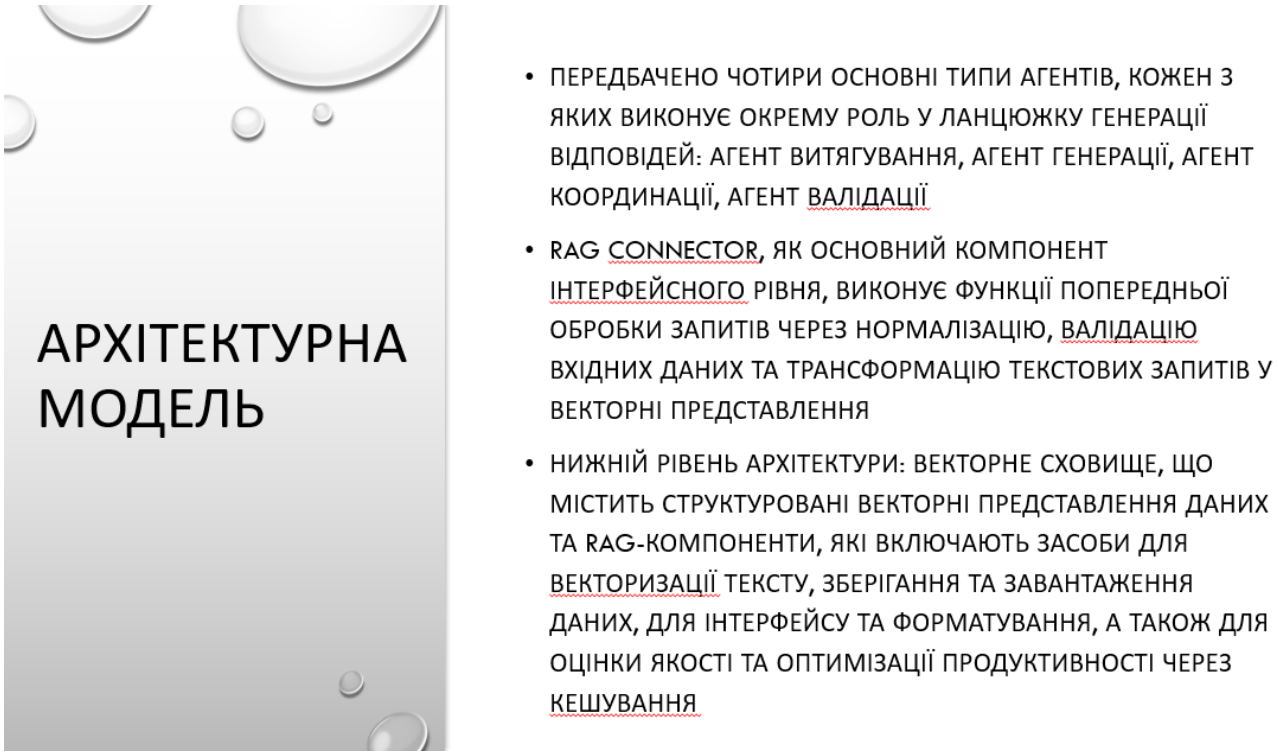

Рисунок В.9 – Слайд 9

Рисунок В.10 – Слайд 10

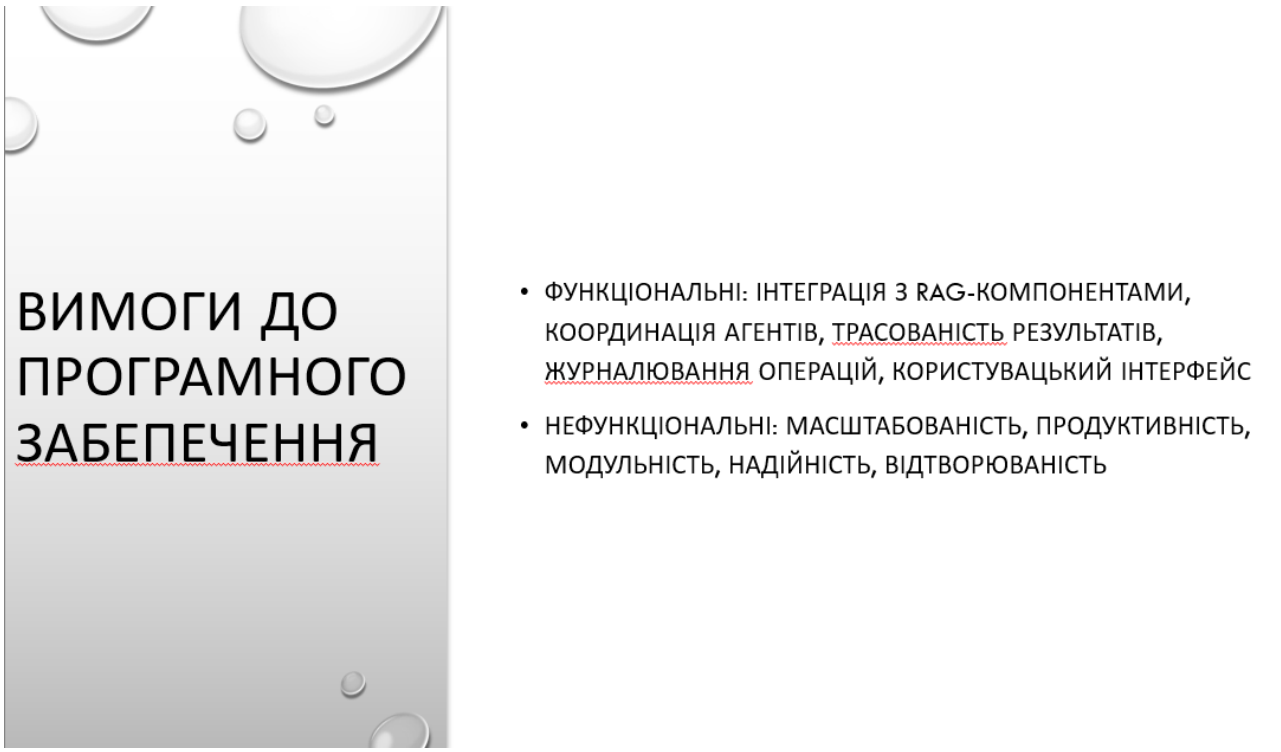

Рисунок В.11 – Слайд 11

Рисунок В.12 – Слайд 12

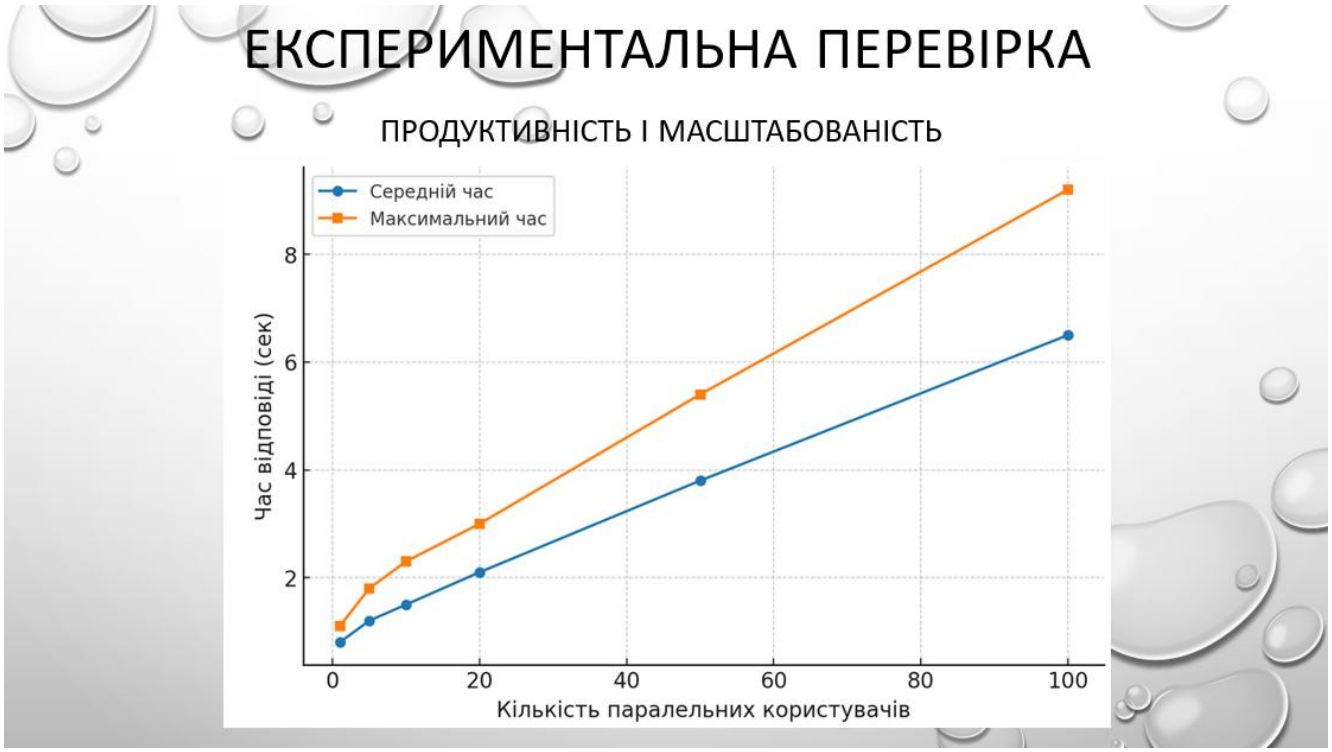

Рисунок В.13 – Слайд 13

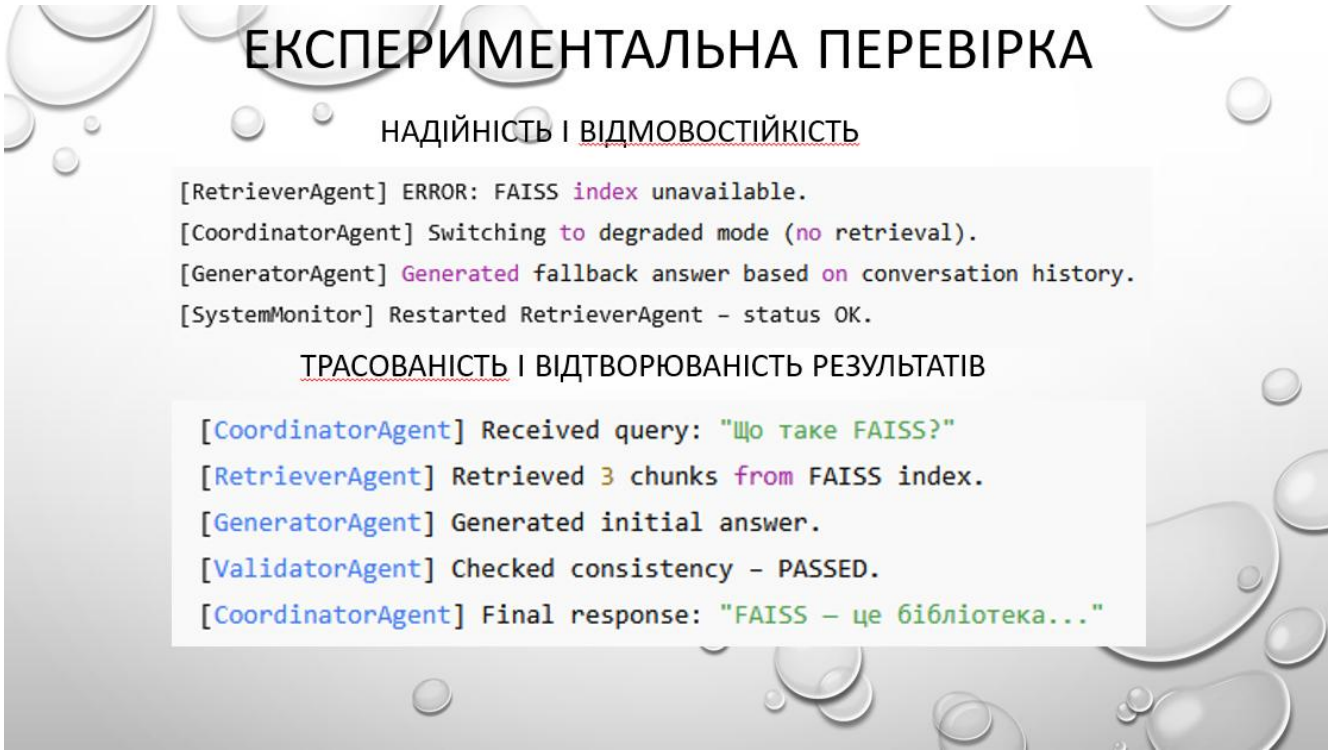

Рисунок В.14 – Слайд 14

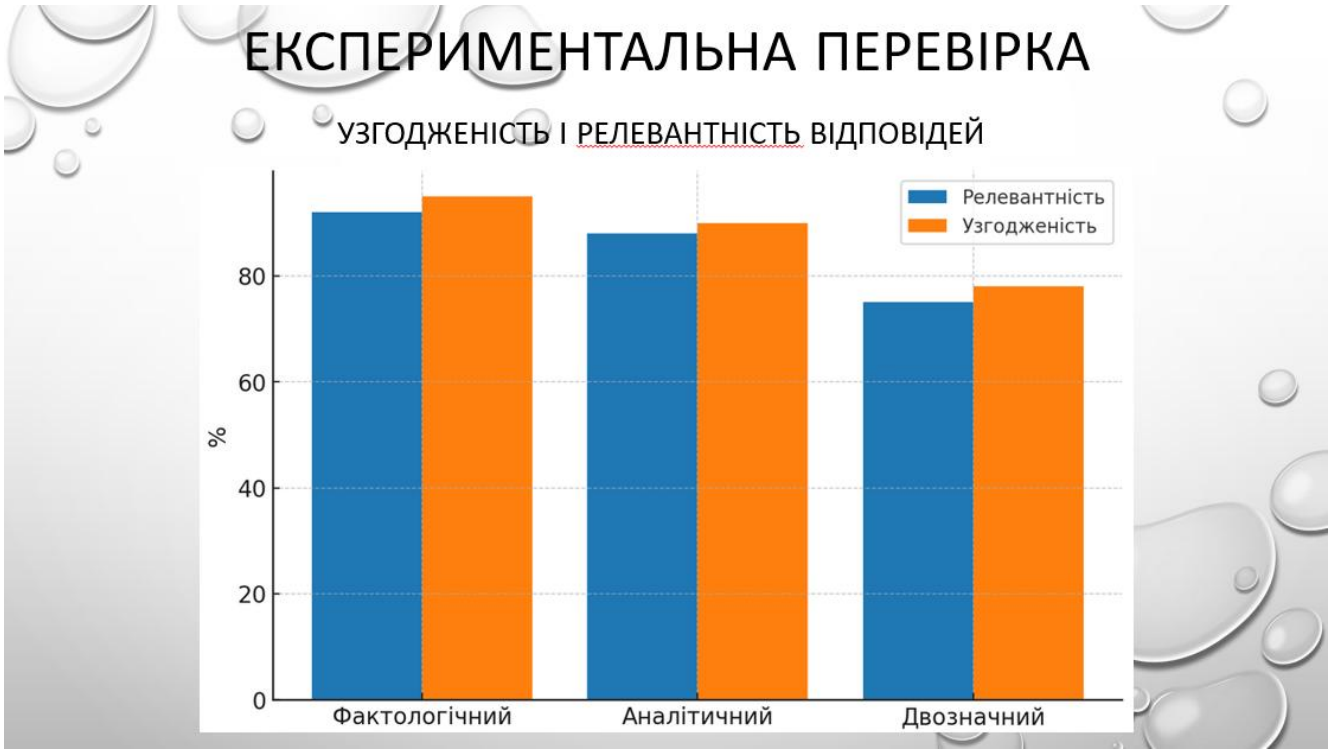

Рисунок В.15 – Слайд 15

Рисунок В.16 – Слайд 16

Рисунок В.17 – Слайд 17

Рисунок В.18 – Слайд 18

Завідувачу кафедри
інженерії програмного забезпечення
проф. Леоніду БЕДРАТЮКУ
студента групи ІПЗм-24-1
Олександра КАТРЕВИЧА

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «магістр» за спеціальністю 121 «Інженерія програмного забезпечення»:

Метод інтеграції компонентів видобування знань (RAG) у мультиагентні програмні системи для забезпечення контекстно-залежного прийняття рішень

керівник кваліфікаційної роботи – Леонід БЕДРАТЮК
Ім'я, ПРІЗВИЩЕ

20.10.2025
Дата

Підпис здобувача

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Катревича Олександра Богдановича
факультет ІТ, 2 курс, група ІПЗм-24-1

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.

08.12.2025р
дата

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: «Метод інтеграції компонентів видобування знань (RAG) у мультиагентні програмні системи для забезпечення контекстно-залежного прийняття рішень»
Автор: Катревич Олександр Богданович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Бедратюк Леонід Петрович, д-р фіз.-мат. наук, професор
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	Перевищено граничне значення рівня подібностей. Виявлені запозичення не є плагіатом, але 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 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 StrickePlagiarism виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, бланк завдання), у структурі змісту, у назвах розділів/підрозділів, у назвах публікацій переліку джерел посилання тощо; 2) в якості запозичень системою StrickePlagiarism було зафіксовано деякі послідовності вихідного коду і посилання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення; 3) запозичення, виявлені в тексті роботи, є фрагментарними або мають належним чином оформлені посилання; 4) виявлені модифікації тексту не впливають на відсоток схожості. Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 1%. За системою StrickePlagiarism коефіцієнт подібності 1 складає 3.3 %, коефіцієнт подібності 2 складає 0.2%.

Дата 8.12.2025

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Оксана ЯШИНА

Керівник кваліфікаційної роботи _____ Леонід БЕДРАТЮК

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: Олександр Катревич

Співавтор:

Назва: Метод інтеграції RAG-компонентів у мультиагентні системи

Науковий керівник: д-р фіз.-мат. наук, професор Бедратюк Л. П.

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:3.3%

Коефіцієнт подібності 2:0.5%

Мікропробіли: 0

Заміна букв: 4

Інтервали: 0

Білі знаки: 176

Дата створення звіту: 2025-12-02 11:55:11.0

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 02.12.2025 експерт

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Здобувач Олександр КАТРЕВИЧ

Тема Метод інтеграції компонентів видобування знань (RAG) у мультиагентні програмні системи для забезпечення контекстно-залежного прийняття рішень

Спеціальність 121 «Інженерія програмного забезпечення»

Обсяг кваліфікаційної роботи:

Кількість листів креслень _____; кількість сторінок записки _____

1.Короткий зміст роботи та прийнятих рішень У роботі розглянуто теоретичні основи побудови мультиагентних систем з інтеграцією компонентів Retrieval-Augmented Generation (RAG). Визначено архітектурну модель таких систем, зокрема принципи взаємодії агентів із зовнішньою базою знань, що реалізується через векторизовані сховища та механізми відновлення контексту. Розроблено класифікацію агентів за типами ролей у системі (контекстні, інтерпретувальні, управлінські) та уточнено функціональну структуру моделі агентної взаємодії. Особливу увагу приділено ролі RAG-компонентів як механізмів змістового збагачення відповідей, що забезпечує баланс між генеративними та фактологічними характеристиками вихідного тексту.

2. Висновок про відповідність роботи поставленому завданню Робота повністю відповідає поставленому завданню, яке полягало в дослідженні та формалізації методів інтеграції RAG-компонентів у мультиагентні системи. Структура та зміст розділу узгоджуються з метою, завданнями й очікуваними результатами, а також демонструють глибоке розуміння предметної області.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У кваліфікаційній роботі кожен розділ має чітку логічну структуру та орієнтований на вирішення конкретного підзавдання проєкту. Перший розділ присвячено аналізу предметної області та стану досліджень у галузі мультиагентних систем і технологій RAG (Retrieval-Augmented Generation). У ньому показано добру обізнаність автора з сучасними підходами, зокрема з використанням індексації знань, систем пошуку та механізмів генерації контекстуально релевантних відповідей. У другому розділі викладено теоретичні засади запропонованого методу інтеграції компонентів RAG у структуру мультиагентної системи. Автором використано сучасні наукові джерела, а також адаптовано передові підходи до побудови agent-based систем із підтримкою контекстної обробки запитів, що свідчить про глибоке розуміння поточних тенденцій у сфері інтелектуальних систем. включно з визначенням вимог, вибором архітектури, описом реалізації ключових компонентів та їх взаємодії. При цьому автор застосував передові методи роботи, такі як розділення функціональності на окремі модулі, використання відкритих API для генерації відповідей, впровадження механізмів збереження діалогової історії, трасування джерел знань, а також забезпечення масштабованості системи. Особливо слід відзначити використання RAG у поєднанні з агентоорієнтованим підходом як інноваційний метод побудови гібридних інтелектуальних систем. Такий підхід є актуальним як у науковій, так і в прикладній площині.

У четвертому підрозділі представлено моделі типових агентів (контекстного, управлінського тощо) разом із формалізованими запитами та структурою відповідей. Робота ґрунтується на сучасних концепціях в області AI-архітектур, зокрема LangChain, CrewAI, FAISS, що свідчить про високий рівень актуальності та новизни. Усі розділи роботи демонструють поєднання теоретичних знань із практичними навичками програмної інженерії на сучасному рівні.

4. Позитивні сторони роботи Позитивною стороною роботи є комплексне висвітлення всіх етапів розробки програмного забезпечення, зокрема обґрунтований вибір архітектурного підходу, чітка декомпозиція та формалізація інтерфейсів, а також використання сучасного стеку технологій, що свідчить про достатню теоретичну підготовку та практичну орієнтованість автора.

5. Негативні сторони роботи Негативною стороною є дещо поверхневий опис модульного тестування, а також недостатньо структуроване подання окремих фрагментів пояснювальної записки, зокрема у частині обґрунтування вибору засобів реалізації

6. Оцінка графічного оформлення та пояснювальної записки роботи Графічне оформлення відповідає вимогам. Усі схеми та діаграми логічні, достатньо читабельні та допомагають зрозуміти структуру запропонованої архітектури. Пояснювальна записка оформлена грамотно, з дотриманням академічного стилю, наукової термінології та загальних вимог до дипломних робіт. Джерела інформації ретельно підібрані, список використаних джерел є актуальним.

7. Відгук про роботу в цілому Робота справляє враження глибоко продуманої, сучасної за тематикою і методами реалізації. Автор продемонстрував здатність до самостійного аналітичного мислення, уміння структурувати складну технічну інформацію і презентувати її у вигляді завершеного дослідження. Запропонована модель мультиагентної системи з використанням RAG-компонентів має потенціал для практичного впровадження. Робота заслуговує високої оцінки.

8. Інші зауваження У роботі варто було додати модуль логування помилок або аналіз помилкових відповідей агентів.

9. Оцінка кваліфікаційної роботи відмінно

РЕЦЕНЗЕНТ (прізвище, ім'я, по-батькові, посада, місце роботи) Алексейко Віталій Олександрович, асистент кафедри комп'ютерної інженерії та інформаційних систем

«15» грудня 2025 р. _____
(підпис)