

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА Пояснювальна записка

рівень вищої освіти другий (магістерський)

**ДОСЛІДЖЕННЯ МЕТОДІВ ПОКРАЩЕННЯ
КОНТЕКСТНОЇ ПАМ'ЯТІ ЧАТ-БОТУ
НА ОСНОВІ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ**
(тема)

Виконав:
здобувач 2 року навчання,
групи ІНФМ-24-1

Богдан Н. І.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Науковий керівник доц. Творошенко І. С.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____
(підпис)

Кобилін О. А.
(прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджментуКафедра ІнформатикиРівень вищої освіти другий (магістерський)Спеціальність 122 Комп'ютерні науки
(код і повна назва)Тип програми освітньо-професійнаОсвітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

« _____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУздобувачеві Богдан Надії Ігорівні
(прізвище, ім'я, по батькові)1. Тема роботи Дослідження методів покращення контекстної пам'яті чат-боту на основі великих мовних моделей

затверджена наказом університету від 14 листопада 2025 року № 1045Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 22 листопада 2025 р.

3. Вихідні дані до роботи методи покращення контекстної пам'яті, літературні джерела щодо застосування методів покращення, програмні засоби для реалізації розглянутих методів покращення пам'яті та застосунків чат-ботів, набір тестових повідомлень та тестовий сценарій для тестування підходів.

4. Перелік питань, що потрібно опрацювати в роботі _____

1. Аналіз сучасних методів покращення контекстної пам'яті чат-боту на основі великих мовних моделей.

2. Аналіз літературних джерел щодо апробації методів покращення контекстної пам'яті чат-боту на основі великих мовних моделей.

3. Формування покрокових алгоритмів для кожного із вибраних методів покращення пам'яті.

4. Візуалізація сформованих покрокових алгоритмів.

5. Розробка програмних застосунків для кожного підходу покращення контекстної пам'яті чат-боту на основі великих мовних моделей.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) актуальність проблеми контекстної пам'яті чат-боту на основі великих мовних моделей, об'єкт та мета дослідження, постановка задачі дослідження, блок-схеми алгоритмів вибраних методів покращення пам'яті, набір тестових даних для перевірки ефективності кожного з підходів, інтерфейс розроблених застосунків для кожного підходу, результати ефективності контекстної пам'яті кожного із розглянутих методів, висновки, перспективи та апробація роботи.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	29.09.2025	
2	Аналіз завдання, підбір літератури	29.09.25-29.09.25	
3	Аналіз літератури з досліджуваної проблеми	29.09.25-01.10.25	
4	Особливості методів покращення контекстної пам'яті чат-боту на основі великих мовних моделей	01.10.25-02.10.25	
5	Дослідження методів покращення контекстної пам'яті чат-боту на основі великих мовних моделей	02.10.25-05.10.25	
6	Програмна реалізація	05.10.25-15.10.25	
7	Обґрунтування отриманих результатів	15.10.25-16.10.25	
8	Оформлення пояснювальної записки	17.10.25-21.10.25	
9	Перевірка на нормоконтроль	19.11.25-21.11.25	
10	Перевірка на плагіат	21.11.25-25.11.25	
11	Рецензування	25.11.25-27.11.25	
12	Підготовка презентації та доповіді	28.11.25-29.11.25	
13	Занесення роботи в електронний архів	29.11.25-30.11.25	
14	Попередній захист кваліфікаційної роботи	01.12.25	

Дата видачі завдання 29 вересня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____
(підпис)

доц. Творошенко І. С.
(посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи: 102 с., 11 табл., 18 рис., 2 дод., 47 джерел.

ВЕБЗАСТОСУНОК, ВЕКТОРИЗАЦІЯ, ЕМБЕДІНГ, КОНТЕКСТ, МОВА ЗАПИТІВ SQL, ПАМ'ЯТЬ, ПРОМПТ, ЧАТ-БОТ, ШІ, AIOGRAM, FASTAPI, LLM, PGVECTOR, POSTGRE SQL, PYTHON, RAG.

Об'єктом дослідження є методи покращення контекстної пам'яті чат-боту на основі великих мовних моделей.

Мета дослідження – порівняти методи покращення контекстної пам'яті чат-боту на основі великих мовних моделей задля вибору та обґрунтування найкращого з них.

Використано методи Метод Sliding Window Context, Summarized (Episodic) Memory та RAG для реалізації чат-боту на основі великих мовних моделей. Проведено аналіз сучасних методів покращення контекстної пам'яті. Сформовано та візуалізовано алгоритми методів блок-схемами.

Наукова новизна роботи полягає у розробленні гібридної методики покращення контекстної пам'яті, враховуючи сильні та слабкі сторони існуючих методів.

Взаємозв'язок з іншими роботами полягає в удосконаленні контекстної пам'яті чат-боту для вирішення складних завдань, що вимагають багатоетапних міркувань або розрахунків.

Рекомендації щодо використання результатів роботи сформовано за результатами тестування та наведено у висновках.

У результаті дослідження розроблено застосунки чат-боту на основі великих мовних моделей з реалізацією методів покращення контекстної пам'яті. У результаті їх тестування отримано статистику точності відповідей відносно усіх підходів.

ABSTRACT

Explanatory note to the qualification work: 102 pages, 11 tables, 18 figures, 2 appendixes, 47 sources.

AIOGRAM, AI, CHATBOT, CONTEXT, EMBEDDING, FASTAPI, LLM, MEMORY, PGVECTOR, POSTGRE SQL, PROMPT, PYTHON, RAG, SQL QUERY LANGUAGE, VECTORIZATION, WEB APPLICATION.

The object of the research is the methods of improving the contextual memory of a chatbot based on large language models.

The aim of the research is to compare methods for improving the contextual memory of a chatbot based on large language models in order to select and justify the most effective one.

The methods used include Sliding Window Context, Summarized (Episodic) Memory and RAG for implementing an LLM-based chatbot. An analysis of modern methods for improving contextual memory and the relevant literature sources has been conducted. Algorithms of the methods have been designed and visualized using block diagrams.

The scientific novelty of the work lies in the development of hybrid methodology for enhancing contextual memory, taking into account the strengths and weaknesses of existing methods.

The connection with other works lies in improving the contextual memory of the chatbot to solve complex tasks that require multi-step reasoning or calculations.

Recommendations for the practical application of the research results are formulated based on the testing outcomes and presented in the conclusions.

As a result of the research, chatbot applications based on large language models were developed with the implementation of contextual memory improvement methods. Their testing produced statistics on the accuracy of responses with respect to all approaches.

ЗМІСТ

Перелік умовних позначень, символів, одиниць, скорочень і термінів	8
Вступ.....	9
1 Аналіз існуючих методів покращення контекстної пам’яті чат-боту на основі великих мовних моделей	11
1.1 Аналіз сучасних методів покращення контекстної пам’яті чат-боту на основі великих мовних моделей та приклади їх практичного використання.....	11
1.1.1 Метод Sliding Window Context	12
1.1.2 Метод Summarized (Episodic) Memory	14
1.1.3 Метод RAG.....	15
1.1.4 Обґрунтування вибору методів для дослідження	17
1.2 Аналіз літературних джерел щодо апробації результатів застосування існуючих методів покращення контекстної пам’яті чат-боту, зокрема на основі великих мовних моделей.....	18
1.3 Постановка задачі дослідження.....	23
2 Особливості вибраних методів покращення контекстної пам’яті чат-боту на основі великих мовних моделей	25
2.1 Метод Sliding Window Context	25
2.2 Метод Summarized (Episodic) Memory.....	26
2.3 Метод RAG	27
2.4 Формування методики для покращення контекстної пам’яті чат-боту на основі великих мовних моделей	28
2.4.1 Огляд переваг та недоліків обраних методів для формування ефективної методики покращення пам’яті чат-боту.....	28
2.4.2 Розроблення гібридної методики	32
2.5 Моделювання структури програмного застосування для покращення контекстної пам’яті чат-боту на основі великих мовних моделей.....	35

2.5.1	Моделювання структури бази даних	35
2.5.2	Моделювання структури програмних застосунків щодо методів покращення контекстної пам'яті	40
3	Дослідження методів покращення контекстної пам'яті чат-боту на основі великих мовних моделей	42
3.1	Вибір інструментальних засобів для реалізації поставлених задач.....	42
3.1.1	Вибір мов програмування та їх фреймворків для програмної реалізації поставлених задач	42
3.1.2	Вибір IDE для програмної реалізації поставлених задач	47
3.1.3	Вибір СУБД для програмної реалізації поставлених задач	48
3.2	Етапи програмної реалізації методів покращення контекстної пам'яті чат-боту на основі великих мовних моделей.....	50
3.2.1	Програмна реалізація Sliding Window Context та Summarized (Episodic) Memory методів	50
3.2.2	Програмна реалізація RAG методу	56
3.2.3	Програмна реалізація гібридного методу.....	58
3.3	Порівняльний аналіз методів покращення контекстної пам'яті чат-боту на основі великих мовних моделей	62
3.3.1	Підготовка тестових даних для тестування методів покращення пам'яті	62
3.3.2	Тестування методів покращення контекстної пам'яті чат-боту	66
3.4	Перспективи подальшої роботи	76
	Висновки	78
	Перелік джерел посилання	80
	Додаток А Фрагмент тестових повідомлень (вихідних даних).....	85
	Додаток Б Відповіді LLM (отриманий результат).....	99

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – база даних

СУБД – система управління базами даних

ШІ – штучний інтелект

API – Application Programming Interface (інтерфейс програмування застосунків)

GPT – Generative Pre-trained Transformer (генеративний попередньо навчений трансформер)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

ID – Identifier (ідентифікатор)

IDE – Integrated Development Environment (інтегроване середовище розробки)

JSON – JavaScript Object Notation (нотація об'єктів JavaScript)

JSONB – JSON Binary (бінарний JSON)

LLM – Large Language Model (велика мовна модель)

LLMs – Large Language Models (великі мовні моделі)

RAG – Retrieval-Augmented Generation (генерація з підкріпленням пошуком даних)

REST API – Representational State Transfer Application Programming Interface (інтерфейс програмування застосунків із передачею репрезентативного стану)

SQL – Structured Query Language (мова структурованих запитів)

UI – User Interface (інтерфейс користувача)

ВСТУП

Контекстна пам'ять – ключовий компонент чат-ботів, котрий забезпечує зв'язність та релевантність відповідей [1]. Проте, більшість сучасних LLMs (великих мовних моделей) мають обмежене контекстне вікно – розмір робочої пам'яті, яку модель може врахувати при генерації відповіді. Наприклад, GPT-3.5 мав вікно близько 4096 токенів, а сучасні моделі – навіть до сотні тисяч токенів на сесію [2].

Однак, навіть при збільшенні контекстного вікна, моделі залишаються обмеженими: ШІ (штучний інтелект) не може розуміти контекст чату з людиною, втрачаючи деталі, котрі були згадані у старих повідомленнях, що значно погіршує якість релевантності відповідей. Це призводить до втрати важливих деталей, повторних запитань, суперечливих відповідей та зниження загальної якості комунікації. У задачах, де від бота очікується послідовне та логічно узгоджене спілкування, наприклад, у довготривалих рольових сценаріях, або під час консультацій із технічних питань, або при супроводі навчального процесу – ці обмеження стають критичними для користувацького досвіду.

Вирішення проблеми обмеженого контекстного вікна LLM-моделей є однією із ключових задач у сфері розроблення чат-ботів. Для цього дослідники та розробники пропонують різні підходи, котрі дозволяють ефективно управляти та розширювати контекстну пам'ять. Один із найпростіших, але дієвих методів є Sliding Window Context. Метод передбачає зберігання лише останніх повідомлень, котрі вміщуються у заданий розмір контексту [3]. Такий підхід дозволяє підтримувати актуальність розмови, але призводить до втрати інформації, яка була надана на початку діалогу.

Більш складні та ефективні підходи включають використання Summarized (Episodic) Memory: старі частини діалогу узагальнюються, а їхні стислі версії додаються до нового запиту. Це допомагає зберегти ключові ідеї розмови, не перевантажуючи контекст.

Найбільш інноваційним та багатообіцяючим методом є RAG (Retrieval-Augmented Generation), який для урахування контексту використовує зовнішню базу даних. Замість того, щоб покладатися виключно на внутрішню пам'ять моделі, RAG-системи шукають релевантну інформацію у базі даних (наприклад, у векторній) і додають її до запиту, що дозволяє чат-боту надавати більш точні та обгрунтовані відповіді, навіть якщо інформація не була частиною поточного діалогу [4, 5].

Завданням кваліфікаційної роботи є порівняльний аналіз, реалізація та оцінка ефективності методів Sliding Window Context, Summarized (Episodic) Memory та RAG щодо покращення контекстної пам'яті чат-боту на основі великих мовних моделей. У рамках дослідження буде розроблено програмні застосунки, де будуть реалізовані зазначені підходи. Це дозволить на практиці дослідити їхні переваги та недоліки, а також визначити, котрий з них є найбільш ефективним у різних сценаріях використання.

Актуальність теми дослідження методів покращення контекстної пам'яті чат-боту на основі великих мовних моделей зумовлена тим, що чат-боти стали невід'ємною частиною цифрового ландшафту та використовуються у різних сферах – від сервісів підтримки клієнтів до персональних помічників та розважальних платформ. Ефективність цих систем безпосередньо залежить від їхньої здатності підтримувати зв'язний та осмислений діалог. Їхня обмежена пам'ять часто призводить до незадовільного користувацького досвіду, що знижує довіру до технологій у цілому [6, 7]. Отже, вирішення цієї проблеми є критично важливим для подальшого розвитку та масштабування систем чат-ботів.

Таким чином, дослідження, котре порівнює різні методи покращення контекстної пам'яті – від базових до передових – є актуальною задачею.

1 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ПОКРАЩЕННЯ КОНТЕКСТНОЇ ПАМ'ЯТІ ЧАТ-БОТУ НА ОСНОВІ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ

1.1 Аналіз сучасних методів покращення контекстної пам'яті чат-боту на основі великих мовних моделей та приклади їх практичного використання

Сучасні методи покращення контекстної пам'яті чат-ботів на основі великих мовних моделей використовують різні підходи для вирішення проблеми обмеженого контекстного вікна. Ці методи дозволяють забезпечити зв'язність, релевантність та послідовність відповідей у довготривалих діалогах. У даному розділі буде розглянуто декілька сучасних методів покращення пам'яті, таких як Sliding Window Context, Summarized (Episodic) Memory та RAG, їх головні переваги та недоліки, а також буде обґрунтовано вибір методів для порівняльного аналізу в рамках кваліфікаційної роботи.

Методики покращення контекстної пам'яті стрімко зростають у кількості та різноманітності, розвиваючись завдяки прогресу в галузі обробки природної мови та машинного навчання. Це зумовлене потребою у появі таких підходів, котрі можуть ефективно управляти потоком інформації, зберігаючи ключові деталі без перевантаження моделі [8].

Методи роботи з контекстною пам'яттю на основі LLMs дозволяють зберігати важливі деталі з діалогу із високою точністю. Однак, головною та актуальною проблемою є потреба у збереженні інформації, котра не вміщується в обмежене контекстне вікно моделі. Для подолання цього обмеження необхідне використання методів, здатних ефективно керувати довготривалою пам'яттю.

У контексті створення діалогових систем, котрі будуть підтримувати зв'язний діалог протягом тривалого часу, важливо використовувати методи, що дозволяють зберігати та знаходити інформацію, яка була надана на ранніх етапах розмови.

Отже, розглянемо основні методи покращення пам'яті.

Sliding Window Context – це метод, котрий використовує підхід «ковзного вікна», де для генерації відповіді враховуються лише найновіші повідомлення діалогу. Метод є ефективним у випадку, коли доступ до попередньої інформації не є критичним. Sliding Window Context підтримує актуальність розмови, навіть якщо для навчання доступні лише поодинокі повідомлення. Даний факт робить його непоганим вибором для чат-ботів з нетривалими діалогами.

Summarized (Episodic) Memory як метод покращення пам'яті чат-боту у довготривалому діалозі полягає у використанні моделі для узагальнення попередніх частин розмови. Вже узагальнені фрагменти діалогу можуть бути адаптовані для використання як додатковий контекст у нових запитах. Це дозволяє значно зменшити потребу в надмірному обсязі контексту, зберігаючи при цьому високу релевантність відповідей [9]. Зазначений спосіб відповідає потребам чат-ботів з підтримкою довготривалих діалогів, де є необхідність в збереженні ключових ідей.

RAG як метод для збагачення контексту LLM працює за принципом інтеграції зовнішніх джерел знань. Його архітектура використовує систему пошуку для вилучення релевантної інформації з бази даних, котра потім подається як додатковий контекст до LLM-моделі [10]. Такий підхід дозволяє чат-боту надавати точні та обґрунтовані відповіді, навіть за наявності мінімальної інформації в поточному діалозі. Під час вирішення задачі створення універсального чат-боту це особливо корисно, оскільки система може ідентифікувати нові теми та надавати відповіді, маючи мінімум даних у поточному діалозі.

1.1.1 Метод Sliding Window Context

Метод Sliding Window Context є одним із базових і найпоширеніших підходів для управління контекстною пам'яттю чат-ботів. Його принципи ґрунтуються на припущенні, що найбільш важлива інформація для генерації наступної відповіді знаходиться у найновіших повідомленнях діалогу.

Суть методу полягає у фіксації максимального розміру контекстного вікна, яке вимірюється в токенах. Коли нове повідомлення додається до діалогу – система перевіряє: чи не перевищено ліміт. Якщо загальна довжина діалогу (в токенах) виходить за межі заданого ліміту, то найстаріші повідомлення, котрі не вміщуються у контекстне вікно, автоматично відкидаються [6]. Таким чином, контекст завжди складається із найновіших фрагментів діалогу, що робить його «ковзним».

Такий підхід є надзвичайно простим у реалізації та не вимагає значних додаткових обчислювальних ресурсів, оскільки не передбачає складних операцій, таких як узагальнення чи пошук у зовнішній базі даних. Його основна перевага полягає у підтримці актуальності розмови, що є критично важливим для короткострокових та середньострокових діалогів. Наприклад, у ситуаціях, коли користувач ставить кілька послідовних запитань, Sliding Window Context дозволяє чат-боту пам'ятати останнє запитання та пов'язані з ним деталі.

Незважаючи на існуючі переваги, метод Sliding Window Context має суттєвий недолік: він призводить до безповоротної втрати довготривалої інформації. Якщо важлива деталь була надана на початку довгої розмови, то вона з часом «випаде» із контекстного вікна, і чат-бот втратить здатність посилатися на неї.

Дана ситуація може призвести до:

- втрати зв'язності діалогу: чат-бот може повторно запитувати інформацію, яку користувач вже надавав;

- нерелевантних відповідей: генеративна модель не зможе врахувати ключові аспекти, котрі були обговорені раніше, що знизить якість відповідей.

Незважаючи на ці обмеження, Sliding Window Context є ефективним і достатнім для багатьох практичних застосувань, де не очікуються тривалі та складні діалоги. Він ідеально підходить для чат-ботів, що використовуються для швидких, сфокусованих запитів, наприклад, в електронній комерції для перевірки статусу замовлення або в технічній підтримці для вирішення типових проблем. У цих сценаріях ключова інформація, зазвичай, міститься в останніх кількох повідомленнях, що робить метод «ковзного вікна» оптимальним вибором.

1.1.2 Метод Summarized (Episodic) Memory

Метод Summarized (Episodic) Memory (узагальнена, епізодична пам'ять) є вдосконаленим підходом до управління контекстом, що дозволяє чат-боту підтримувати зв'язність діалогу протягом тривалого часу, не втрачаючи ключову інформацію. Крім того, на відміну від методу «ковзного вікна», котрий безповоротно видаляє старі повідомлення, цей підхід намагається зберегти їхню сутність. Основний його принцип полягає у використанні LLM для створення стислого узагальнення попередніх частин розмови.

Процес реалізації цього методу, зазвичай, включає такі кроки:

Крок 1. Накопичення діалогу: чат-бот зберігає повну історію розмови до тих пір, поки її довжина не наблизиться до максимального розміру контекстного вікна.

Крок 2. Узагальнення: у випадку, коли ліміт досягнутий, чат-бот надсилає частину найстаріших повідомлень до LLM з інструкцією «Узагальни наступний діалог у стислій формі, зберігаючи ключові ідеї та факти».

Крок 3. Оновлення контексту: отримане узагальнення додається до контексту, замінюючи собою оригінальні, довші повідомлення.

Таким чином, контекстне вікно складається із узагальненої пам'яті та найновіших повідомлень діалогу.

Переваги методу Summarized (Episodic) Memory очевидні. Він дозволяє ефективно зберігати ключову інформацію з усієї розмови. Це значно покращує якість відповідей у довготривалих діалогах, оскільки модель може посилається на факти, згадані значно раніше. Завдяки тому, що узагальнення займає набагато менше токенів, ніж оригінальний діалог, цей метод є більш ефективним з точки зору використання контекстного вікна, ніж Sliding Window Context.

Однак, існують і недоліки. Головним недоліком є обчислювальні витрати, пов'язані з необхідністю додаткових звернень до LLM для узагальнення тексту.

Це може збільшити затримку між запитом і відповіддю, а також вартість використання API. Крім того, процес узагальнення може призвести до втрати дрібних деталей, які не були включені до стислої версії.

Summarized (Episodic) Memory ідеально підходить для розробки чат-ботів у сферах, де важлива послідовність та діалогові асистенти для управління проектами, котрі повинні пам'ятати обговорювані завдання протягом декількох сесій.

1.1.3 Метод RAG

Метод RAG є одним із найпотужніших та найінноваційніших підходів для покращення контекстної пам'яті чат-ботів, що базуються на великих мовних моделях. На відміну від попередніх методів, котрі працюють лише з діалогом, RAG-системи доповнюють внутрішні знання LLM зовнішніми джерелами даних. Цей підхід дозволяє моделі виходити за межі свого початкового контексту і генерувати відповіді, засновані на актуальних та перевірених фактах.

Основний принцип роботи RAG-системи складається з трьох ключових етапів:

Етап 1. Пошук (retrieval): запит користувача аналізується і перетворюється на векторне подання (embedding). Цей вектор використовується для пошуку найбільш релевантних фрагментів інформації у зовнішній базі даних векторного типу, котра містить векторизовані документи, статті або будь-яку іншу текстову інформацію.

Етап 2. Доповнення (augmentation): знайдені релевантні фрагменти інформації додаються до початкового запиту користувача. Це створює «розширений» запит, котрий містить як питання користувача, так і додатковий контекст.

Етап 3. Генерація (generation): розширений запит надсилається до LLM-моделі, яка використовує цю інформацію для формування більш точної, обґрунтованої та достовірної відповіді.

Приклад архітектури RAG-системи зображено на рисунку 1.1 [11].

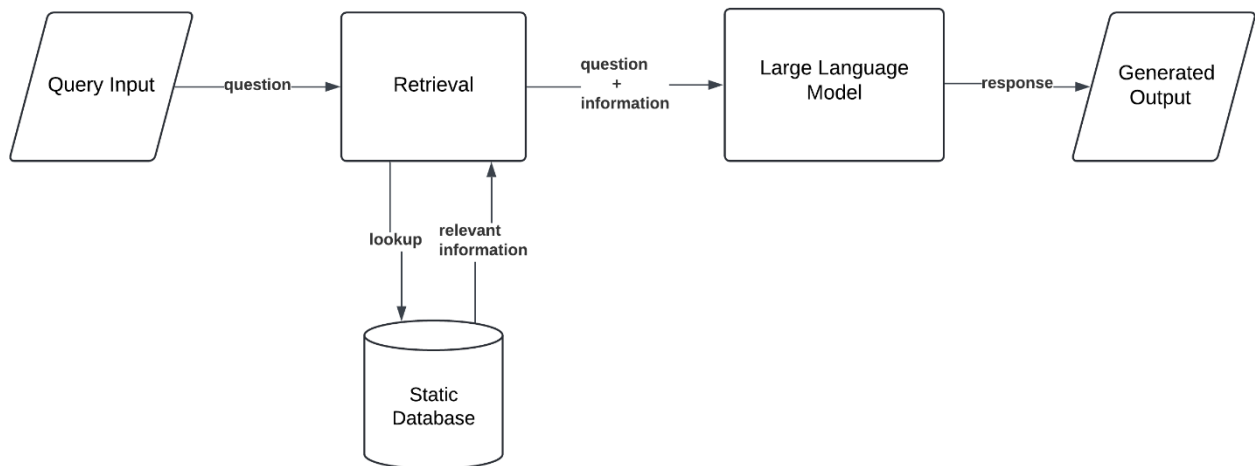


Рисунок 1.1 – Архітектура RAG-системи

Переваги RAG-підходу є значними. Він дозволяє чат-боту надавати високоточні відповіді, оскільки вони ґрунтуються на актуальних даних. Цей метод також допомагає зменшити «галюцинації» (проблему, коли LLM генерує неправдиву інформацію), так як модель має конкретні факти для посилання. А ще RAG-системи можуть працювати з великими обсягами знань, що робить їх ідеальними для корпоративних застосунків, де потрібен доступ до великої кількості документації або баз даних [12].

Щодо недоліків RAG, то вони полягають у його складності. Реалізація вимагає створення та підтримки додаткових компонентів, таких як система векторизації та векторна база даних. Це може призвести до збільшення обчислювальних витрат та затримки під час обробки запиту, оскільки необхідно виконати пошук перед генерацією відповіді [13].

RAG ідеально підходить для розробки внутрішніх корпоративних асистентів, котрі відповідають на запитання співробітників, використовуючи внутрішню базу знань компанії, або для наукових чат-ботів, які можуть надавати відповіді на основі декількох тисяч наукових статей.

Цей підхід є найкращим рішенням для завдань, котрі вимагають високої точності та доступу до динамічних або специфічних даних.

1.1.4 Обґрунтування вибору методів для дослідження

Обрання для дослідження Sliding Window Context, Summarized (Episodic) Memory та RAG методів є обґрунтованим, оскільки дозволяє охопити весь спектр сучасних підходів – від найпростішого до найскладнішого, що забезпечує комплексний та багатоаспектний аналіз.

Метод Sliding Window Context був обраний як базовий підхід. Цей метод є найменш ресурсомістким, але має суттєві обмеження щодо довготривалої пам'яті. Його аналіз дозволить встановити початкову точку відліку для порівняння, оцінивши мінімальний рівень ефективності та виявивши недоліки, які інші методи мають подолати. Дослідження цього підходу дасть розуміння того, як працюють найпростіші реалізації та чому виникла потреба в складніших рішеннях.

Метод Summarized (Episodic) Memory був обраний як проміжний підхід. Він є значно складнішим за Sliding Window Context, але простішим за RAG, оскільки не вимагає зовнішньої бази знань. Цей метод демонструє, як можна покращити пам'ять, використовуючи лише можливості LLM-моделі для узагальнення інформації. Порівняння його з Sliding Window Context дозволить оцінити ефективність узагальнення як стратегію збереження довготривалого контексту.

Метод RAG був обраний як передовий метод. Він представляє найсучасніший підхід, що використовує гібридну архітектуру для поєднання внутрішніх знань LLM із зовнішньою базою даних. Дослідження RAG дозволить оцінити максимальну ефективність, якої можна досягти в сучасних умовах, а також виявити переваги та недоліки, пов'язані з його складністю та обчислювальними витратами.

Таким чином, вибір цих трьох методів забезпечує повноцінну структуру для порівняльного аналізу. Дослідження буде сфокусоване на ключових аспектах: точності та якості відповідей у різних сценаріях використання. Це дозволить надати об'єктивну оцінку кожного підходу та сформулювати рекомендації щодо їх застосування у залежності від вимог до діалогового чат-боту.

1.2 Аналіз літературних джерел щодо апробації результатів застосування існуючих методів покращення контекстної пам'яті чат-боту, зокрема на основі великих мовних моделей

Для формування повного розуміння існуючих підходів до покращення контекстної пам'яті, проведено глибокий аналіз наукових публікацій. Цей підрозділ містить розширений огляд літературних джерел, котрі стосуються апробації результатів застосування існуючих методів покращення контекстної пам'яті чат-боту на основі великих мовних моделей, із акцентом на їхні переваги, недоліки, обмеження та практичну цінність.

У статті [2] автори досліджують, як структура пам'яті впливає на ефективність ШІ у складних завданнях, таких як діалогові системи. Автори вивчають чотири різні типи структур пам'яті: чанки (chunks), трійки знань (knowledge triples), атомні факти (atomic facts) та узагальнення (summaries), а також їхню комбінацію – змішану пам'ять (mixed memory). Ключовим результатом роботи є те, що різні структури пам'яті мають свої переваги і можуть бути адаптовані до конкретних завдань. Наприклад, атомарні факти можуть бути корисними для точних відповідей на запитання, тоді як узагальнення краще підходять для підтримки зв'язності в діалогах. Трійки знань можуть допомогти моделювати складні взаємозв'язки між сутностями. Автори виявили, що змішана пам'ять, котра поєднує різні типи структур, демонструє

чудову стійкість, особливо в умовах «шумного» середовища, де дані можуть бути неповними або містити помилки.

У статті [5], котра є однією із основоположних для методу RAG, автори вивчали методіку, як вона поєднує можливості великих мовних моделей із доступом до зовнішньої бази знань. Основна мета роботи полягала у подоланні ключових обмежень традиційних моделей, котрі зберігають знання лише у своїх параметрах: обмежена здатність до точного маніпулювання фактами, відсутність джерел інформації та складність оновлення знань моделі. Незважаючи на те, що в статті не обговорюються недоліки безпосередньо, характер RAG-системи має певні обмеження. Архітектура RAG є значно складнішою, так як вимагає додаткових компонентів (наприклад, векторну базу даних), що робить реалізацію та підтримку системи більш ресурсномісткою. Також якість відповідей залежить від якості та повноти зовнішньої бази даних.

У статті [6], що є оглядом великих мовних моделей для діалогових систем, автори вивчали підходи до управління контекстом. Серед них метод Sliding Window Context розглядається як один із найпростіших та найпоширеніших методів. У роботі зазначається, що цей метод є достатньо ефективним для короткострокових, сфокусованих розмов, де інформація швидко втрачає свою актуальність. Проте, незважаючи на простоту і легкість реалізації, в роботі висвітлюються достатньо значні недоліки цього підходу. Як вже зазначено, найзначніший та найочевидніший – занадто обмежені можливості моделі у довготривалих та складних діалогах.

На відміну від традиційних методів, котрі зосереджені на послідовному або узагальненому збереженні діалогу, стаття [7] представляє систему пам'яті, засновану на тематичній векторній базі даних. Це дозволяє LLM ефективно управляти довгостроковими спогадами, зберігаючи переваги високої швидкості оброблення. Замість збереження історії діалогу, автори використовують промпт-інжиніринг, щоб спонукати LLM витягувати ключові теми, вподобання та характеристики користувача із розмови. Тематичні дані перетворюються на векторні подання та зберігаються у спеціалізованій базі даних. Коли користувач

робить новий запит, система спочатку шукає релевантні тематичні вектори, а потім додає їх до запиту, дозволяючи моделі формувати високоперсоналізовану та осмислену відповідь. Однак, як і будь-який інноваційний підхід, цей метод має потенційні обмеження. Якість системи безпосередньо залежить від точності промпт-інжинірингу та здатності LLM правильно ідентифікувати та витягувати ключові теми. Якщо модель неправильно класифікує або пропускає важливу інформацію, то це може призвести до неточностей у майбутньому.

Стаття [9], що є оглядом LLMs, доповнених контекстом через використання методу Summarized (Episodic) Memory, є вкрай актуальною для досліджень у сфері розроблення діалогових систем. Метод рекурсивної сумаризації, що є імплементацією Summarized (Episodic) Memory, пропонує ефективне та масштабоване рішення. Незважаючи на те, що рекурсивна сумаризація допомагає зберегти ключові ідеї, існує ризик, що дрібні, але важливі деталі можуть бути втрачені в процесі узагальнення. Якщо в діалозі є специфічна інформація (наприклад, точні дати, імена або числа), котра не була включена в резюме, то модель не зможе її відновити. Також виявлена значна залежність від якості сумаризації: ефективність методу безпосередньо залежить від якості резюме, яке генерує сама LLM. Якщо модель погано справляється із сумаризацією або допускає «галюцинації», це може призвести до накопичення помилок у пам'яті, що з часом спотворюватиме контекст діалогу.

У роботі [10] автори досліджують RAG метод. Експерименти, що були проаналізовані в цьому огляді, підтверджують, що RAG-моделі значно перевершують звичайні LLM у достовірності та обґрунтованості відповідей, особливо в задачах, що вимагають доступу до великих обсягів знань. Це досягається завдяки тому, що RAG надає моделі конкретні факти, що значно знижує ризик «галюцинацій». Перевагами цього підходу є висока точність і здатність до роботи з великими обсягами знань. Крім того, RAG дозволяє оновлювати знання моделі, просто оновлюючи зовнішню базу даних, що є набагато ефективнішим, ніж повне перенавчання LLM. Дана робота також вказує на ряд суттєвих обмежень. RAG-системи є дуже ресурсомісткими та складними

в реалізації. Вони вимагають створення та підтримки додаткових компонентів, таких як векторні бази даних та системи індексування, що значно збільшує вимоги до інфраструктури та обчислювальних ресурсів.

У статті [12] описано дослідження методу RAFT (Retrieval-Augmented Fine-Tuning), котрий є вдосконаленням існуючих підходів до покращення можливостей великих мовних моделей, зокрема, у сфері генеративних діалогових систем. Автори пояснюють, що Retrieval-Augmented Fine-Tuning інтегрує потужність RAG, котра забезпечує доступ до зовнішньої, актуальної інформації, з методом Chain-of-Thought, що дозволяє моделі розбивати складні завдання на послідовні кроки. Це поєднання критично важливе для підвищення логічних міркувань. Додавання Supervised Fine-Tuning дозволяє додатково налаштувати модель на конкретні завдання, забезпечуючи ефективне використання як витягнутих даних, так і розбиття на кроки для досягнення кращих результатів.

У статті [13] авторами подано вдосконалений підхід до RAG, який вони назвали Multiple Partitions RAG. Це дослідження виявляє ключове обмеження традиційного RAG – організацію всіх даних в одній великій базі. Автори стверджують, що такий підхід може призвести до «шуму» та зниження фокусу на найбільш критичних для задачі спогадах. Multiple Partitions RAG пропонує нову парадигму: розподіляти дані на кілька окремих розділів, кожен з яких служить базовою одиницею для RAG. Запропонований підхід є особливо актуальним для завдань, де пам'ять має бути розділена за темами або доменами (наприклад, окремі розділи для історії розмов, технічної документації, персональних даних). Таке сегментування дозволяє створювати більш сфокусовані та ефективні чат-боти, котрі не «плутаються» у великих обсягах інформації.

У статті [14] автори проводять комплексне дослідження та запроваджують новий набір даних для оцінки ефективності різних методів управління пам'яттю у довготривалих діалогах. Дослідники зосередилися на вивченні таких методів, як Sliding Window Context та Summarized (Episodic) Memory. Основна мета

роботи полягала в тому, щоб надати об'єктивну оцінку продуктивності кожного з підходів у реальних умовах, що вимагають збереження контексту протягом довготривалих обмінів повідомленнями. Автори стверджують, що метод Sliding Window Context демонструє гірші результати, коли діалог перевищує розмір контекстного вікна. Крім того, дослідження показало, що метод Summarized (Episodic) Memory значно перевершує «ковзне вікно». LLMs, котрі використовували узагальнену пам'ять, демонстрували кращу зв'язність і точність у відповідях, оскільки вони могли посилалися на ключові ідеї, збережені у вигляді стислих резюме. Проте, результати апробації також виявили певні обмеження узагальненої пам'яті. У тих випадках, коли для відповіді були потрібні дрібні, специфічні деталі, які не потрапили в узагальнення, LLMs з цим підходом демонстрували неточність. Це підтверджує тезу про компроміс між стислістю і деталізацією, що є ключовим недоліком методу Summarized (Episodic) Memory. Цей висновок особливо важливий для завдань, де не можна допускати втрати навіть найдрібніших фактів.

Однак, основний недолік статті [14] полягає в її зосередженості виключно на вивченні методів управління пам'яттю на основі вже існуючих моделей та наборів даних. Хоча автори надають аналіз, їхня робота не включає розроблення рішення, яке, враховуючи результати дослідження методів, реалізує оптимальну методику для покращення пам'яті. Вони не пропонують готового до впровадження алгоритму, котрий дозволив би розробникам інтегрувати таку архітектуру у власні проєкти. Стаття є суто науковою і не включає пропозицію кращої методики та пошук оптимального рішення, покликаною вирішити недоліки вже існуючих методів.

У статті [15] автори проводять глибокий аналіз етичних та практичних викликів, пов'язаних із впровадженням довготривалої пам'яті у діалогових системах на основі великих мовних моделей. На відміну від попередніх джерел, які фокусувалися на ефективності та архітектурі, ця робота зосереджена на потенційних ризиках, що виникають, коли чат-боти зберігають історію розмов. Основна мета дослідження полягала в тому, щоб виявити та класифікувати ці

ризика, а також запропонувати можливі шляхи їх мінімізації [16, 17]. Однією з головних проблем, яку порушують автори, є безпека та конфіденційність даних. Довготривала пам'ять, незалежно від того, чи реалізована вона через узагальнення, RAG чи ієрархічні системи, по суті є постійним сховищем персональних даних користувачів [18].

Таким чином, проаналізовано низку джерел на тему кваліфікаційної роботи, тим самим, підтверджено актуальність даного дослідження.

1.3 Постановка задачі дослідження

Розроблення чат-боту на основі великих мовних моделей, що здатний підтримувати довготривалу контекстну пам'ять, є актуальним і важливим завданням. Прийнято рішення щодо реалізації та порівняння основних методів управління пам'яттю: Sliding Window Context, Summarized (Episodic) Memory та RAG, а також розроблення покращеної методики.

Об'єктом дослідження є методи покращення контекстної пам'яті чат-боту на основі великих мовних моделей.

Мета дослідження – порівняти методи покращення контекстної пам'яті чат-боту на основі великих мовних моделей задля вибору та обґрунтування найкращого з них.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- провести аналіз сучасних методів покращення контекстної пам'яті чат-ботів на основі великих мовних моделей;
- вивчити та проаналізувати літературні джерела щодо апробації методів, що забезпечують довготривалий контекст у діалогових системах;
- сформулювати покрокові алгоритми для кожного із вибраних методів управління пам'яттю та візуалізувати їх у вигляді блок-схем;
- сформулювати покроковий алгоритм для удосконаленого методу управління пам'яттю, котрий врахує переваги та недоліки існуючих методів;

- розробити програмні застосунки, на основі котрих буде проведено тестування ефективності методів покращення пам'яті;
- провести тестування удосконаленої методики та порівняти її ефективність із обраними для дослідження методами за ключовими метриками, зокрема, за точністю та зв'язністю відповідей у довготривалих діалогах;
- на основі порівняльного аналізу та оцінки ефективності зробити висновок щодо використання методів в залежності від вимог до діалогової системи.

2 ОСОБЛИВОСТІ ВИБРАНИХ МЕТОДІВ ПОКРАЩЕННЯ КОНТЕКСТНОЇ ПАМ'ЯТІ ЧАТ-БОТУ НА ОСНОВІ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ

2.1 Метод Sliding Window Context

Розглянемо детально метод Sliding Window Context, його схематичне подання зображено на рисунку 2.1.

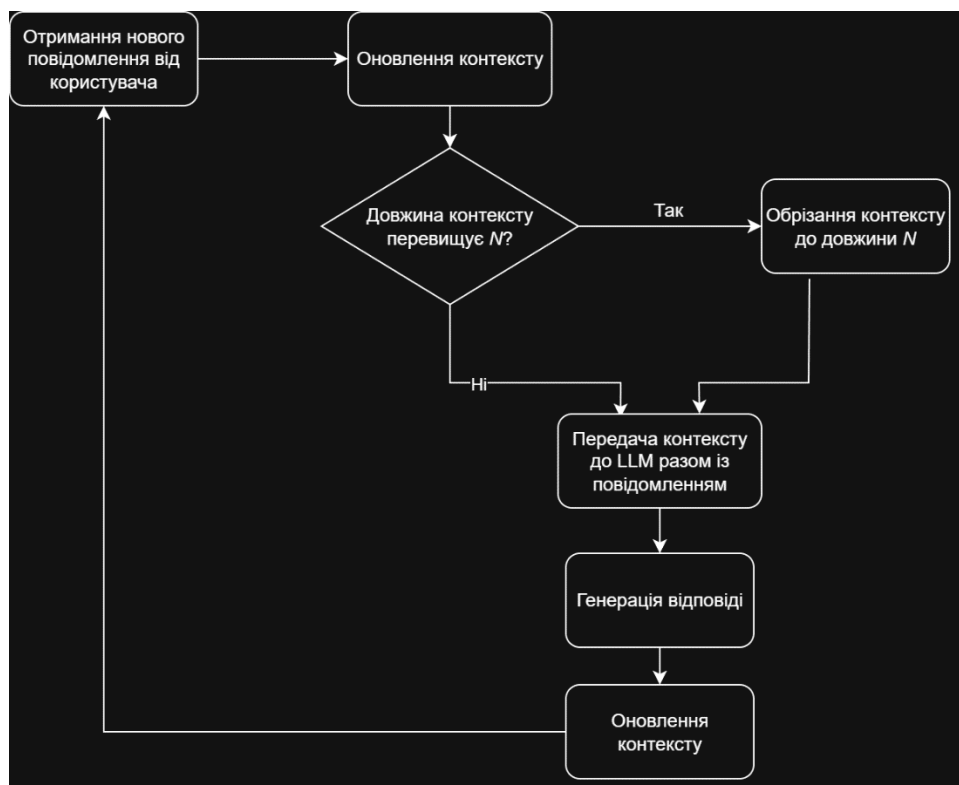


Рисунок 2.1 – Схематичне подання методу Sliding Window Context

Метод Sliding Window Context передбачає виконання таких ключових етапів:

Етап 1. Ініціалізація: на цьому етапі створюється порожній контекстний буфер фіксованого розміру N (наприклад, 4000 токенів).

Етап 2. Отримання повідомлення: система перехоплює новий запит від користувача.

Етап 3. Оновлення контексту: поточний контекстний буфер об'єднується з новим запитом. Якщо загальний розмір перевищує N токенів, то обрізається початок буфера, щоб він вмщував лише останні N токенів.

Етап 4. Генерація відповіді: оновлений контекст разом із запитом передається у велику мовну модель для генерації відповіді.

Етап 5. Збереження: згенерована відповідь додається до контекстного буфера.

Етап 6. Повторення: етапи 2-5 повторюються для кожного запиту.

2.2 Метод Summarized (Episodic) Memory

Розглянемо детально метод Summarized (Episodic) Memory, його схематичне подання зображено на рисунку 2.2.

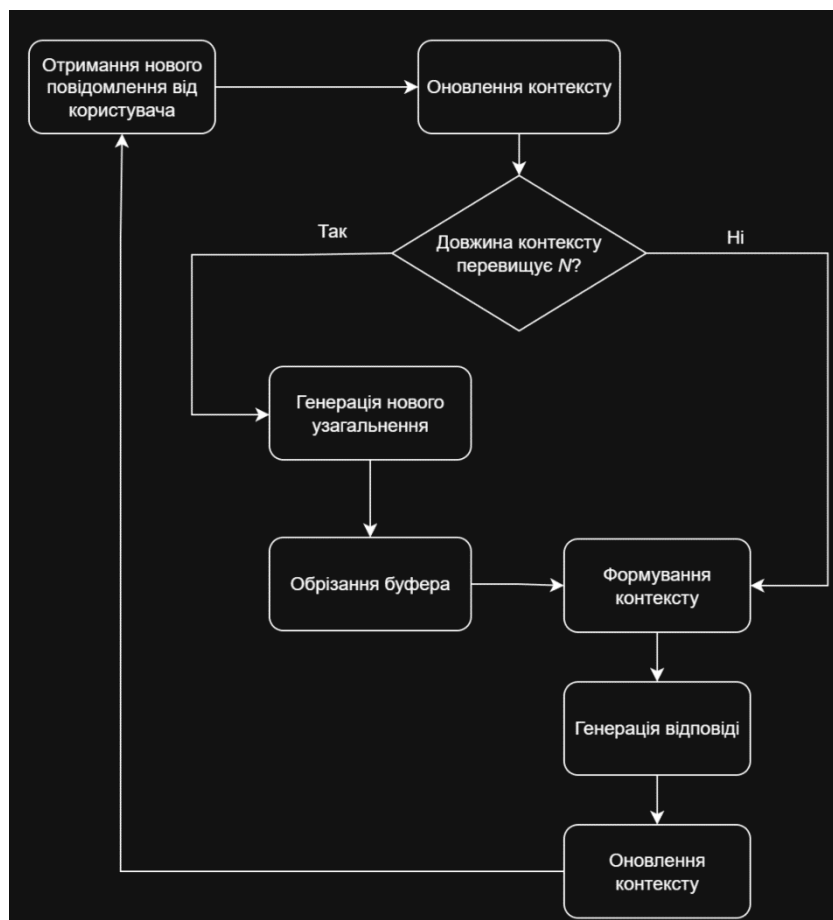


Рисунок 2.2 – Схематичне подання методу Summarized (Episodic) Memory

Метод Summarized (Episodic) Memory передбачає виконання таких етапів:

Етап 1. Ініціалізація: створюється порожній контекстний буфер фіксованого розміру N , а також змінна для узагальнення (S).

Етап 2. Отримання повідомлення: система перехоплює новий запит від користувача.

Етап 3. Оновлення контексту: узагальнення (S) та поточний контекстний буфер об'єднуються з новим запитом. Якщо загальний розмір перевищує N токенів, то необхідно передати частину контексту (наприклад, половину) у велику мовну модель для узагальнення. Потім замінити узагальнення (S) на нове, більш релевантне. Після цього обрізається контекстний буфер до заданого розміру N і до нього додається нове узагальнення.

Етап 4. Генерація відповіді: оновлений контекст з узагальненням і запитом передаються до LLM.

Етап 5. Збереження: згенерована відповідь додається до контекстного буфера.

Етап 6. Повторення: етапи 2-5 повторюються для кожного запиту.

2.3 Метод RAG

Розглянемо детально метод RAG, який передбачає виконання таких етапів:

Етап 1. Ініціалізація: створюється векторна база даних для майбутнього зберігання історії діалогу.

Етап 2. Отримання повідомлення: система перехоплює новий запит від користувача.

Етап 3. Векторизація: запит користувача перетворюється на векторне представлення (embedding) за допомогою моделі-кодувальника.

Етап 4. Пошук релевантних даних: отриманий вектор використовується для пошуку найбільш схожих записів у векторній базі даних.

Етап 5. Формування контексту: початковий запит користувача об'єднується із найбільш релевантними знайденими записами з векторної бази даних.

Етап 6. Генерація відповіді: сформований контекст передається у велику мовну модель для генерації відповіді.

Етап 7. Оновлення бази даних: новий запит і згенерована відповідь додаються до векторної бази даних для майбутнього використання.

Етап 8. Повторення: етапи 2-7 повторюються для кожного запиту.

Схематичне подання методу RAG зображено на рисунку 2.3.

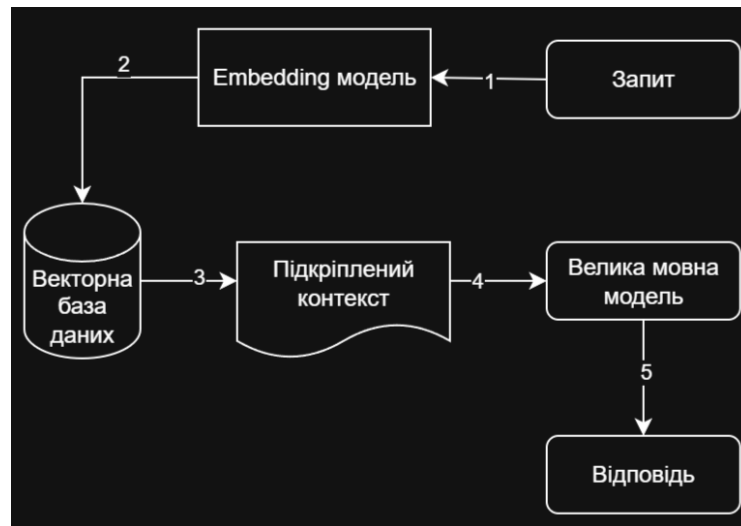


Рисунок 2.3 – Схематичне подання методу RAG

2.4 Формування методики для покращення контекстної пам'яті чат-боту на основі великих мовних моделей

2.4.1 Огляд переваг та недоліків обраних методів для формування ефективної методики покращення пам'яті чат-боту

Для того, щоб розробити нову методику, котра перевершить розглянуті методи покращення пам'яті чат-боту, необхідно здійснити порівняльний аналіз переваг та недоліків усіх обраних методів Sliding Window Context, Summarized (Episodic) Memory та RAG [19–21].

Виявлені сильні та слабкі сторони даних підходів з аналітичної точки зору зазначено у таблиці 2.1.

Таблиця 2.1 – Переваги та недоліки вибраних методів

Критерії	Sliding Window Context	Summarized (Episodic) Memory	RAG
1	2	3	4
Складність реалізації	Низька. Потребує лише базових операцій з рядками.	Середня. Вимагає інтеграції з LLM для узагальнення, а також логіки для відстеження розміру контексту.	Висока. Потребує налаштування та підтримки векторної бази даних, логіки для векторизації запитів.
Обчислювальні витрати	Низькі. Збільшуються лінійно з довжиною діалогу, але обрізання обмежує витрати.	Середні. Додаткові витрати на узагальнення, які можуть бути значними для довгих фрагментів діалогу.	Високі. Витрати на векторизацію запитів, пошук по базі даних та обробку великої кількості знайдених результатів.
Довготривала пам'ять	Інформація з початку діалогу безповоротно втрачається.	Зберігає узагальнену суть розмови, але втрачає специфічні деталі.	Має доступ до всієї історії діалогу, що зберігається в базі даних.

Продовження таблиці 2.1

1	2	3	4
Точність та зв'язність	Низька. Різко падає, коли потрібні старі деталі.	Середня. Достатня для підтримки логіки, але може бути неточною у деталях.	Умовно висока. «Заземлює» відповіді на конкретних фактах. Генеративна модель не отримує спеціалізованих промптів для аналізу контексту, специфічних інструкцій.
Сфери застосування	Короткі діалоги, прості запити, тимчасові взаємодії.	Узагальнені діалоги, консультації, де не потрібна висока точність у деталях.	Фактичні задачі, технічна підтримка, персональні помічники, що вимагають високої точності.
Автоматичне оновлення станів згідно до контексту	Відсутнє.	Відсутнє.	Відсутнє.

Посилаючись на проведений аналіз (табл. 2.1), можна зробити висновок, що кожен із розглянутих методів має свої унікальні переваги, але й, натомість, значні обмеження [22, 23].

Крім того, ще залишаються виклики, які не вирішує ні єдиний із розглянутих методів, а саме:

– генеративна модель не отримує спеціалізованих промптів для аналізу контексту, специфічних інструкцій та сталих метаданих, як саме використати релевантну інформацію, щоб згенерувати логічну відповідь;

– діалогова система не передбачає автоматичного оновлення станів: наприклад, у чат-боті рольових сценаріїв, де динамічно оновлюється контекст у процесі діалогу та з'являється висока залежність від емоційного стану, а рольові персони користувача та LLM розвиваються як особистості, є потреба у слідуванні за станами. У випадку із рольовим сценарієм, персони змінюють відношення один до одного (наприклад, підвищується коефіцієнт довіри або навпаки), ця динаміка має динамічно аналізуватися та змінюватися залежно від подій та розвитку.

Таким чином, щоб розробити високоефективну методику покращення пам'яті чат-боту, буде розглянуто можливість поєднання сильних сторін різних підходів у певний гібрид [24].

Кожен метод пропонує компроміс між простотою, швидкістю та ефективністю. Метод Sliding Window Context – є швидким і простим, але він безповоротно втрачає інформацію, що унеможлиблює його використання в довготривалих діалогах. На противагу йому, RAG забезпечує найвищу точність, але вимагає значних обчислювальних ресурсів [25]. Ця розбіжність формує запитання: чи можливо створити методику, яка поєднає швидкість і простоту з глибиною та точністю, уникнувши при цьому їхніх основних недоліків?

Іноді класичний RAG може видавати відповіді, котрі є точними, але відчуються механічними, оскільки він покладається виключно на семантичні фрагменти [26]. Інформація, отримана з бази даних, є набором схожих за семантикою старих повідомлень. Вони, у свою чергу, можуть не враховувати тонкий, безпосередній контекст недавніх взаємодій. Sliding Window Context, зберігаючи «свіжі» повідомлення у контексті, надає мовній моделі чітку зв'язність у відповідях [27, 28].

Враховуючи вищезазначене, цей своєрідний «тандем» двох протилежних підходів може компенсувати недоліки один одного та об'єднати свої переваги у ефективну методику.

Головний недолік Sliding Window Context – втрата інформації – повністю нівелюється завдяки RAG. Вся історія діалогу, навіть та, що була втрачена через обмеження розміру, зберігається у векторній базі даних. Отже, чат-бот зможе «згадати» будь-яку деталь, що була на початку розмови, а також генеруватиме логічні та зв'язні відповіді [29].

Таким чином, гібридна архітектура вирішить найважливішу проблему обмеженої пам'яті завдяки RAG, а природну та тонку контекстну зв'язність відповідей – завдяки Sliding Window Context.

Процес розроблення методу вимагає додаткових покращень. Це можна реалізувати шляхом додавання нових кроків, котрі безпосередньо впливають на якість «розуміння» контексту мовною моделлю, оновлюючи стани у реальному часі та маючи ефективні промпти із метаданими. Встановлено, що, якщо ігнорувати такі важливі аспекти, то існують великі ризики того, що чат-бот буде генерувати не консистентні та доволі «механічні» відповіді не завжди логічно та природньо поєднані контекстом взаємодій з користувачем. Отже, запропоноване удосконалення має сенс [30].

2.4.2 Розроблення гібридної методики

Оскільки архітектура гібридної методики базуватиметься на поєднанні RAG та Sliding Window Context методів, вирішено об'єднати їх ключові етапи. Передбачається удосконалення об'єднанням сильних сторін обох підходів, а також шляхом додавання нових етапів [31, 32].

Щоб удосконалити гібрид RAG та Sliding Window Context методів, буде впроваджено нові заходи щодо покращення контекстної пам'яті LLM, котрі не передбачаються у кроках цих підходів. Для вирішення знайдених прогалин методів RAG та Sliding Window Context буде розроблено чотирирівневу архітектуру побудови пам'яті. Підхід поєднуватиме різні рівні контексту, кожен з яких реалізує специфічні функції, що дозволить системі динамічно адаптуватися до потреб діалогу [33].

Гібридна методика буде ґрунтуватися на базі чотирирівневої архітектури системи пам'яті, котра синхронно оброблятиме та зберігатиме дані для формування максимально повного контексту:

Рівень 1. Статична пам'ять (база знань).

На цьому рівні відбуватиметься зберігання константних даних, котрі формуватимуть основу взаємодії. Це включатиме профілі ботів (наприклад, героїв рольових сценаріїв або спеціалістів технічної підтримки), загальні відомості про специфічне середовище (вигаданий світ, якщо це рольовий сценарій, або ж екосистема – якщо середовищем є компанія) та метадані. Інформація зберігатиметься в базі даних. Статична інформація буде відносно постійною і буде первинним джерелом достовірних відомостей.

Рівень 2. Динамічна пам'ять (історія взаємодій).

На цьому рівні відбуватиметься зберігання та інтелектуальне вилучення всієї історії діалогу – шляхом виконання кроків RAG методу: кожне повідомлення чату, збережене в історії, векторизуватиметься, а при отриманні нового запиту система виконає пошук, котрий знайде змістовно схожі повідомлення з усієї історії. Додатково, система застосує пріоритизацію: нещодавні та емоційно значущі події отримують вищий бал, що робить їх більш релевантними для включення в контекст.

Рівень 3. Контекстний аналіз.

На даному рівні динамічно оновлюватиметься інформація про стани, емоції та вплив середовища. Після кожного повідомлення запускатиметься спеціалізований ШІ-агент для аналізу. Цей агент аналізуватиме взаємодію, ідентифікуватиме учасників (якщо у чаті більше, ніж один користувач та більше, ніж один ШІ-бот) і, використовуючи повний контекст (профілі, історія, існуючі стани та відносини між персонами користувача та ШІ-бота), зможе розуміти приховані сенси та емоційний підтекст. На основі аналізу оновлюватимуться записи у базі даних, що стосуються відносин між персонами бота та користувача, їхнього емоційного стану та розвитку подій чату.

Рівень 4. Оперативна пам'ять (або поточний контекст).

На цьому рівні відбуватиметься зберігання найбільш актуальної інформації для відповіді. Даний рівень включатиме кроки методу Sliding Window Context. Найновіші повідомлення матимуть найвищий пріоритет серед усього контексту, що міститиме останні 15–25 повідомлень у хронологічному порядку, а також поточні емоції та активні стани. LLM завжди матиме доступ до найсвіжіших подій, що є ключовим для підтримки плавної та природної розмови.

Схематичне подання гібридного підходу зображене на рисунку 2.4.

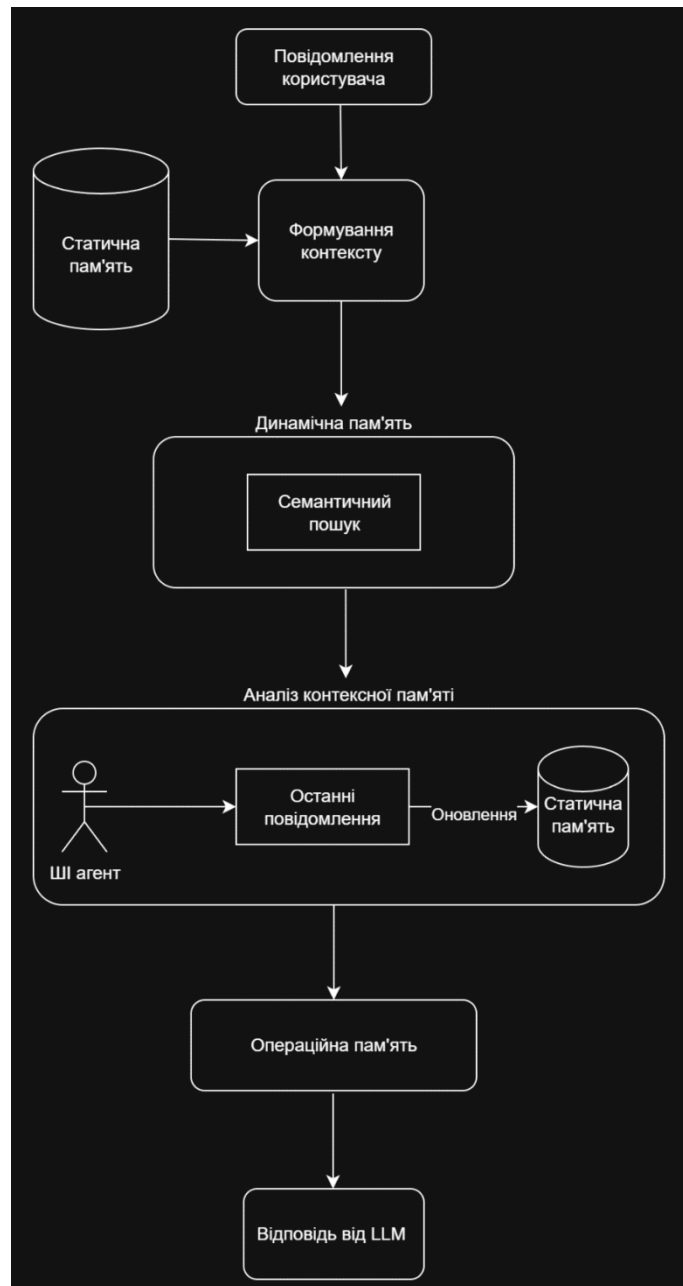


Рисунок 2.4 – Схематичне подання гібридного підходу

Процес формування промпта для LLM буде багат шаровим: він починається з найрелевантніших даних, отриманих за допомогою семантичного пошуку (рівень 2), доповнюється інформацією з статичної пам'яті (рівень 1), контекстного аналізу (рівень 3) та оперативної (рівень 4) пам'яті. Це забезпечить повноту та багатовимірність контексту, що стане основою для генерації високоякісних відповідей.

2.5 Моделювання структури програмного застосунку для покращення контекстної пам'яті чат-боту на основі великих мовних моделей

2.5.1 Моделювання структури бази даних

Для реалізації методів покращення контекстної пам'яті чат-боту буде створено моделі бази даних (БД).

Оскільки Sliding Window Context та Summarized (Episodic) Memory методи не вимагають складних структур БД для своєї роботи, а також мають багато спільного у своїх архітектурах, то для них буде спроектовано єдину БД.

Структура БД для цих методів буде гнучкою і зможе ефективно підтримувати як метод Sliding Window Context, так і Summarized (Episodic) Memory, що зробить її універсальною основою для діалогових систем [34].

Метод Sliding Window Context, що оперує останніми повідомленнями, БД використовуватиметься здебільшого для зберігання повної історії чату.

Схематичне подання БД для Sliding Window Context та Summarized (Episodic) Memory методів представлено на рисунку 2.5.

Ключовою таблицею є messages. Вона зберігає всі повідомлення діалогу в хронологічному порядку.

Для реалізації «ковзного вікна» система має звертатися до цієї таблиці, вибираючи лише останні N повідомлень. Використання поля created_at або id з ORDER BY дозволяє легко отримати найновіші повідомлення, що відповідає логіці методу.

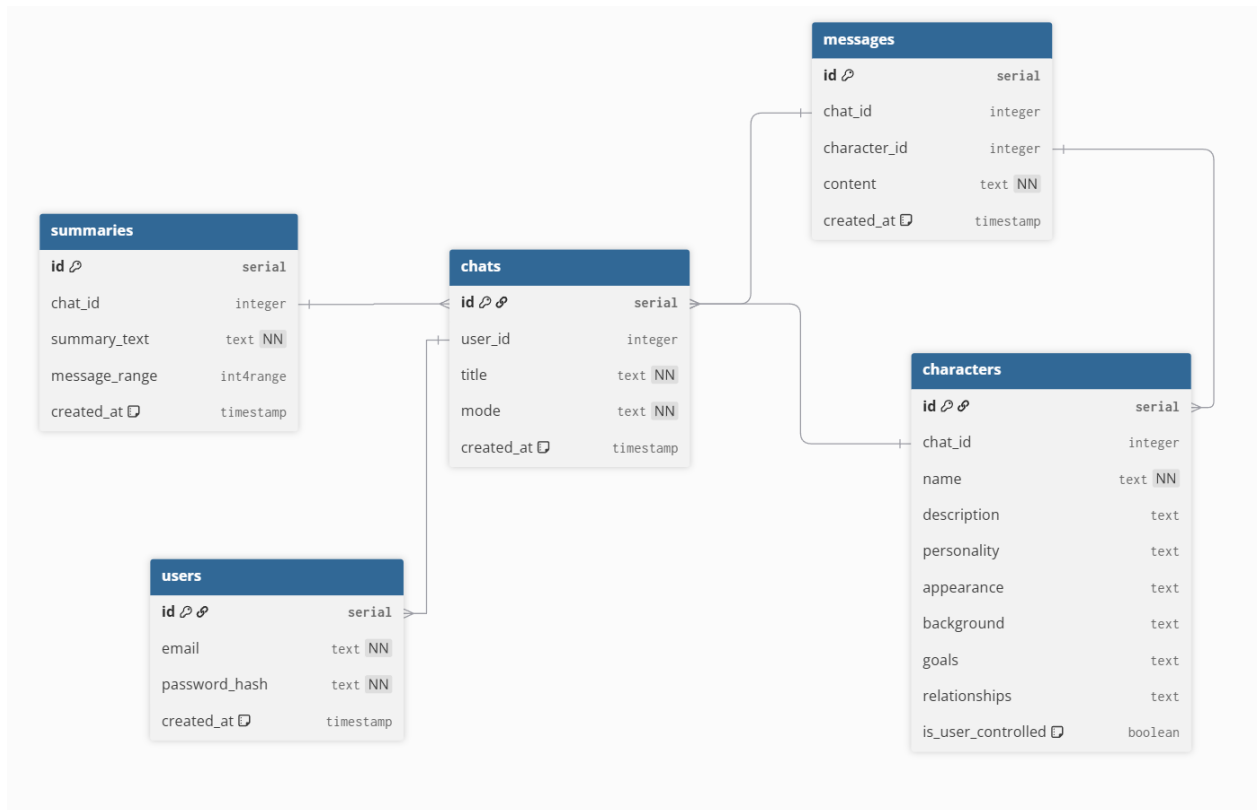


Рисунок 2.5 – Схематичне подання бази даних для Sliding Window Context та Summarized (Episodic) Memory методів

Summarized (Episodic) Memory вимагає додаткової таблиці для зберігання узагальнень, що дозволяє зберігати довготривалий контекст, не перевантажуючи робочу пам'ять моделі.

Як і у випадку з Sliding Window, таблиця messages використовується для зберігання повної історії діалогу.

Таблиця summaries – це додаткова, спеціалізована таблиця, що дозволяє реалізувати метод Summarized (Episodic) Memory. Ключовим полем є summary_text – текстове поле, у котре зберігається узагальнення попередньої частини діалогу. Поле message_range вказує, які повідомлення (за id) були включені до цього узагальнення, це допоможе уникнути дублювання і забезпечить логічну послідовність узагальнень.

Логіка роботи полягає в тому, що коли діалог у таблиці messages досягає певного порогу, частина старих повідомлень передається в LLM для узагальнення.

Згенерований `summary_text` разом із відповідним діапазоном повідомлень (`message_range`) зберігається в таблиці `summaries`. У майбутніх запитах, коли потрібно врахувати старий контекст, система може просто додати останнє узагальнення з таблиці `summaries` до поточного контексту з таблиці `messages`.

Таким чином, зазначена структура БД є достатньо гнучкою для підтримки обох підходів. `Sliding Window Context` використовує лише таблицю `messages`, тоді як `Summarized (Episodic) Memory` інтегрує додаткову таблицю `summaries`, що є ключовим компонентом методу, імплементуючи його головний функціонал.

На відміну від методів `Sliding Window Context` та `Summarized (Episodic) Memory`, які можуть використовувати одну й ту саму базову структуру, RAG вимагає унікального архітектурного підходу. Її ключова відмінність від попередніх схем полягає в інтеграції векторних подань даних, що є основою RAG.

Схему моделювання БД для методу RAG подано на рисунку 2.6.

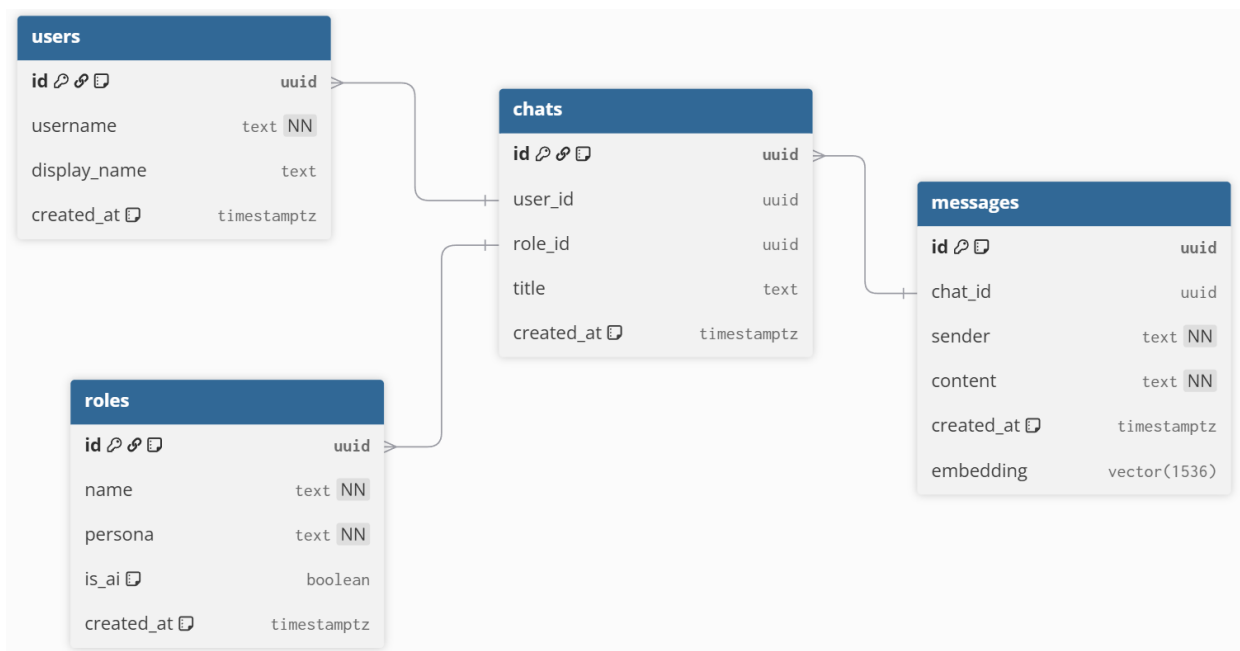


Рисунок 2.6 – Схематичне подання бази даних для RAG методу

Таблиця `messages` є центральним елементом для RAG. Крім стандартних полів (`id`, `chat_id`, `content`, `created_at`), вона містить критично важливе поле `embedding`.

Це поле зберігає векторне подання кожного повідомлення, що є цифровим «відбитком» його семантичного змісту. Наявність цього поля дозволяє виконувати семантичний пошук, котрий є основою RAG.

Таблиці `users`, `chats` та `roles` надають додаткові метадані, необхідні для організації та ідентифікації діалогів та їх учасників.

Гібридна методика базується на чотирирівневій архітектурі пам'яті, тож матиме свою БД, структура котрої врахує усі рівні контекстного наповнення. Архітектура БД є фундаментом для реалізації методики та повинна чітко розмежовувати різні типи пам'яті, таким чином, забезпечуючи їхню ефективну взаємодію [35].

Схематичне подання БД для гібридного методу зображено на рисунку 2.7.

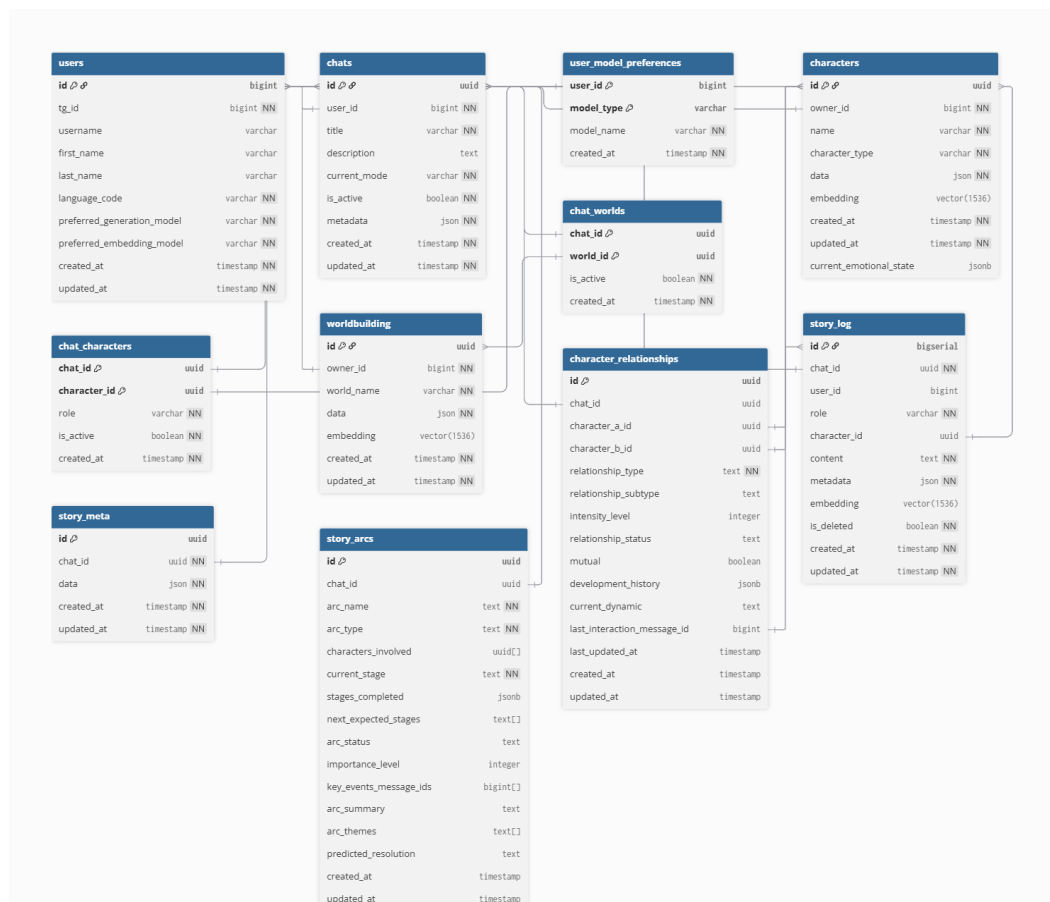


Рисунок 2.7 – Схематичне подання бази даних для гібридного методу

За рівень статичної пам'яті відповідають таблиці `worldbuilding`, `story_meta`, `characters`. Вони є основою константних даних. Ключовою особливістю є використання поля `embedding` для векторизації цієї інформації, що дозволяє системі швидко знаходити базові знання за допомогою семантичного пошуку. Таблиця `characters` містить детальні профілі, що є ядром статичної пам'яті, також з векторним представленням.

Таблиця `story_log` – це центральний компонент динамічної пам'яті системи. Вона зберігає повну історію діалогу, фіксуючи кожне повідомлення. Критично важливе поле `embedding` перетворює цю історію на векторний простір, що робить можливим високоточний RAG. Таблиця слугує зовнішньою, масштабованою пам'яттю, що дозволяє LLM звертатися до будь-якої частини розмови, незалежно від її давності.

Контекстний аналіз базується на таблицях `character_relationships` та `story_arcs`. Таблиці відображають динамічний розвиток подій та відносин у чаті. `character_relationships` зберігає еволюцію зв'язків між персонами, фіксуючи їхній тип та інтенсивність. `story_arcs` відстежує активні лінії подій, їхні стадії. Ці таблиці є результатом роботи ШІ-агента, котрий аналізує діалог і оновлює цю «живу» пам'ять, дозволяючи системі підтримувати глибоку, послідовну розмову.

Оперативна пам'ять реалізується через оптимізований запит до таблиці `story_log`. Система вибирає з неї останні 15–25 повідомлень, котрі мають найвищий пріоритет. Ця концепція «ковзного вікна» забезпечує негайний доступ до найактуальнішого контексту, що є ключовим для природності та швидкості відповіді.

Таким чином, схема БД є оптимальним архітектурним рішенням, котре об'єднує статичні дані, повну історію діалогу, динамічний аналіз відносин та оперативний доступ до останніх повідомлень. Це дозволяє системі не просто генерувати відповіді, а ще й підтримувати багатовимірну, реалістичну взаємодію.

2.5.2 Моделювання структури програмних застосунків щодо методів покращення контекстної пам'яті

Для реалізації методів покращення контекстної пам'яті чат-боту буде спроектовано архітектуру програмних застосунків, котрі реалізують методи покращення контекстної пам'яті.

Застосунки буде розроблено на основі архітектури, яка чітко розділяє відповідальність між рівнями. Такий підхід забезпечує модульність, що є ключовим для подальшої масштабованості та гнучкості.

Для методів Sliding Window Context та Summarized (Episodic) Memory буде розроблено єдиний застосунок з використанням спроектованої БД. На рівні бізнес-логіки буде створено пакет services, котрий стане ядром застосунку. Тут буде зосереджена вся бізнес-логіка, яка реалізує методику покращення пам'яті. Це розділення дозволяє змінювати логіку, наприклад, перемикатися між Sliding Window Context та Summarized (Episodic) Memory без впливу на маршрути чи БД.

Архітектура застосунку для методу RAG буде теж чітко розділена на модулі, кожен з яких матиме одну відповідальність. Бізнес-логіка також буде ключовим компонентом та буде зосереджена у пакеті services. Саме тут буде реалізовано алгоритм RAG. Цей модуль відповідатиме за збір контексту: отримання запиту від користувача, виклик функцій для семантичного пошуку в БД, отримання релевантних фрагментів даних, формування повного промπτу, котрий включає запит користувача та знайдені дані, передача промπτу до модуля генерації відповіді.

Застосунок для гібридної методики буде реалізовано шляхом побудови складної та багатомодульної архітектури. Архітектура бізнес-логіки буде спроектована також за принципом розділення відповідальності, де кожен модуль виконуватиме чітко визначену функцію. На рівнях модулів core та rag буде відбуватися ключова логіка застосунку, а саме – реалізація гібридної методики.

Замість одного монолітного сервісу, бізнес-логіка буде розділена на декілька спеціалізованих менеджерів. У пакеті core будуть реалізовані менеджери, котрі відповідатимуть за управління основними сутностями системи. Вони будуть виконувати логіку роботи з рівнем статичної пам'яті та частково з рівнем контекстного аналізу. Окремо під рівень контекстного аналізу будуть реалізовані менеджери, котрі відповідатимуть за складну логіку аналізу взаємодій та динамічне оновлення відносин та емоцій персон.

Спеціалізований пакет, котрий повністю інкапсулюватиме логіку RAG, буде відокремлений та буде реалізовувати перетворення тексту на векторні представлення. У цьому пакеті також буде реалізовано логіку семантичного пошуку по векторам у БД. Важливим компонентом стане модуль, відповідальний за формування промпту: у ньому збиратимуться усі необхідні дані (з усіх 4 рівнів пам'яті) в одному структурованому промпті для LLM. Це буде ключова точка, де буде відбуватися інтеграція всіх рівнів гібридної методики.

3 ДОСЛІДЖЕННЯ МЕТОДІВ ПОКРАЩЕННЯ КОНТЕКСТНОЇ ПАМ'ЯТІ ЧАТ-БОТУ НА ОСНОВІ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ

3.1 Вибір інструментальних засобів для реалізації поставлених задач

3.1.1 Вибір мов програмування та їх фреймворків для програмної реалізації поставлених задач

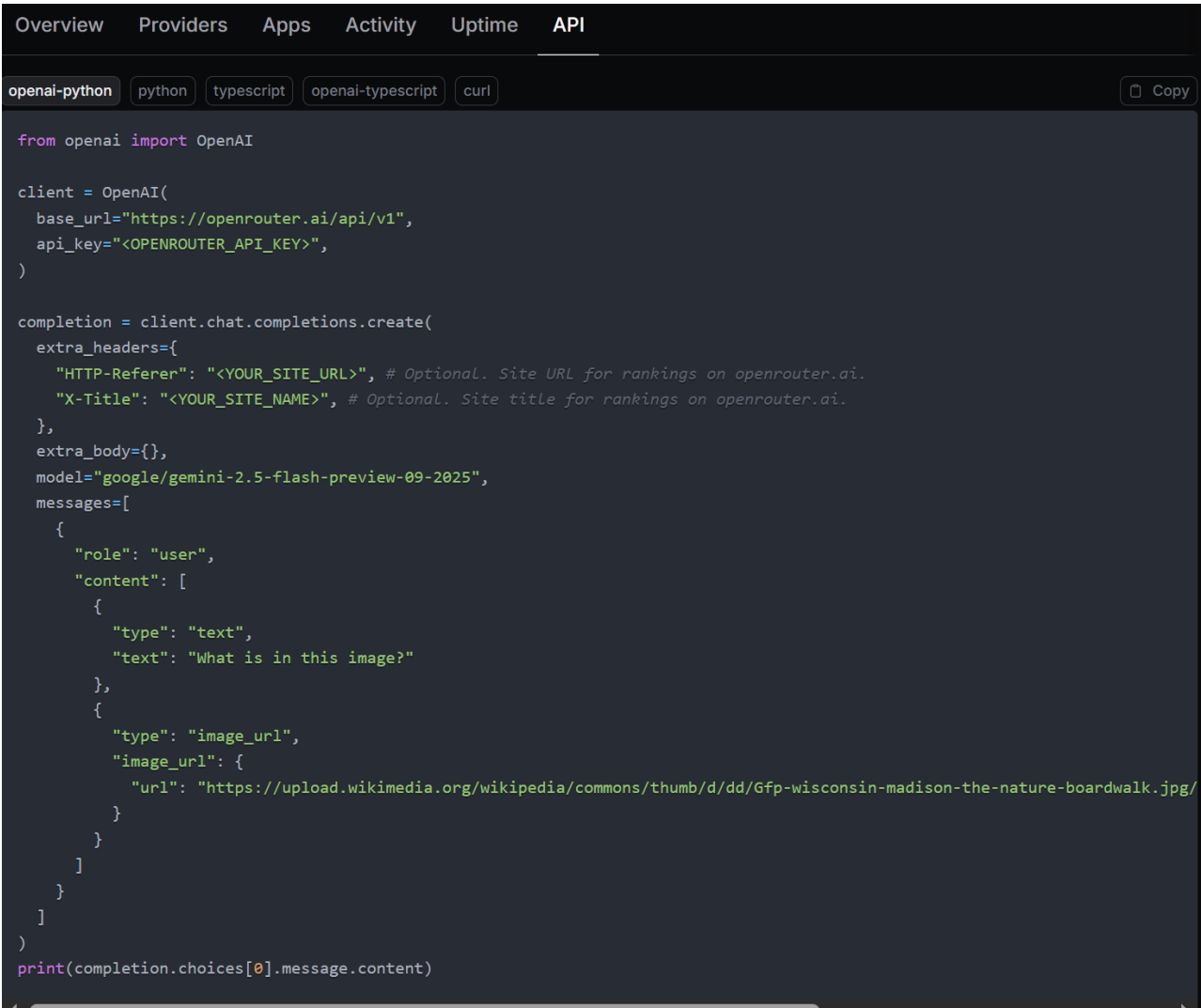
Для розроблення методів покращення контекстної пам'яті чат-боту на основі великих мовних моделей обрано мову програмування Python, оскільки вона дуже зручна у розробці, а також спеціалізована як під ШІ-орієнтовані задачі, так і для побудови бекенду. Python підтримує сучасні асинхронні вебфреймворки, що дозволяє ефективно будувати високопродуктивні бекенд-застосунки для API-взаємодії.

На рівні усіх застосунків будуть використовуватися великі мовні моделі для генерації відповідей, сумаризації та контекстного аналізу. Прийнято рішення щодо їхнього використання через API-інтерфейси, що забезпечує гнучкість та економічну ефективність.

Провайдер OpenRouter API обраний як уніфікований інтерфейс для доступу до різноманітних великих мовних моделей. Його ключова перевага полягає у мультимодельності та оптимізації витрат. OpenRouter дозволяє динамічно перемикатися між різними LLM (наприклад, GPT-4, Claude, Gemini), обираючи найбільш ефективну модель для конкретного завдання (генерація, сумаризація, аналіз). Це забезпечує надійність і гнучкість на етапі експериментального порівняння методів.

Провайдер OpenAI Embeddings API обрано для генерації векторних представлень, що є критично важливим для реалізації RAG та гібридного підходу, оскільки саме вектори дозволяють системі здійснювати швидкий і точний семантичний пошук в довготривалій пам'яті.

Приклад програмного коду на Python та API для підключення до LLM через OpenRouter зображено на рисунку 3.1 [36].



```
Overview Providers Apps Activity Uptime API
openai-python python typescript openai-typescript curl Copy

from openai import OpenAI

client = OpenAI(
    base_url="https://openrouter.ai/api/v1",
    api_key="<OPENROUTER_API_KEY>",
)

completion = client.chat.completions.create(
    extra_headers={
        "HTTP-Referer": "<YOUR_SITE_URL>", # Optional. Site URL for rankings on openrouter.ai.
        "X-Title": "<YOUR_SITE_NAME>", # Optional. Site title for rankings on openrouter.ai.
    },
    extra_body={},
    model="google/gemini-2.5-flash-preview-09-2025",
    messages=[
        {
            "role": "user",
            "content": [
                {
                    "type": "text",
                    "text": "What is in this image?"
                },
                {
                    "type": "image_url",
                    "image_url": {
                        "url": "https://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/Gfp-wisconsin-madison-the-nature-boardwalk.jpg/"
                    }
                }
            ]
        }
    ]
)

print(completion.choices[0].message.content)
```

Рисунок 3.1 – Приклад програмного коду на Python та API для моделі Gemini 2.5 Flash Preview 09-2025 через інтерфейс OpenRouter

Для ефективної реалізації та порівняльного аналізу методів покращення контекстуальної пам'яті обрано технологічний стек, який відповідає високим вимогам до швидкості розробки, обчислювальної ефективності та зручної інтеграції із LLMs. Для реалізації застосунків, у яких імплементуються методи Sliding Window Context, Summarized (Episodic) Memory та RAG, буде використано FastAPI вебфреймворк.

FastAPI повноцінно підтримує асинхронне програмування, що дозволяє ефективно управляти паралельними запитами користувачів без блокування. Це є ключовою перевагою при роботі з LLM, оскільки генерація відповідей може займати значний час.

FastAPI автоматично генерує документацію API (OpenAPI), що значно спрощує тестування, інтеграцію та верифікацію реалізованих методів. Тобто, фреймворк спрощує проєктування API, автоматично генеруючи документацію. Це сприяє чіткому розділенню рівнів та полегшує тестування.

Приклад API документації FastAPI показано на рисунку 3.2 [37].

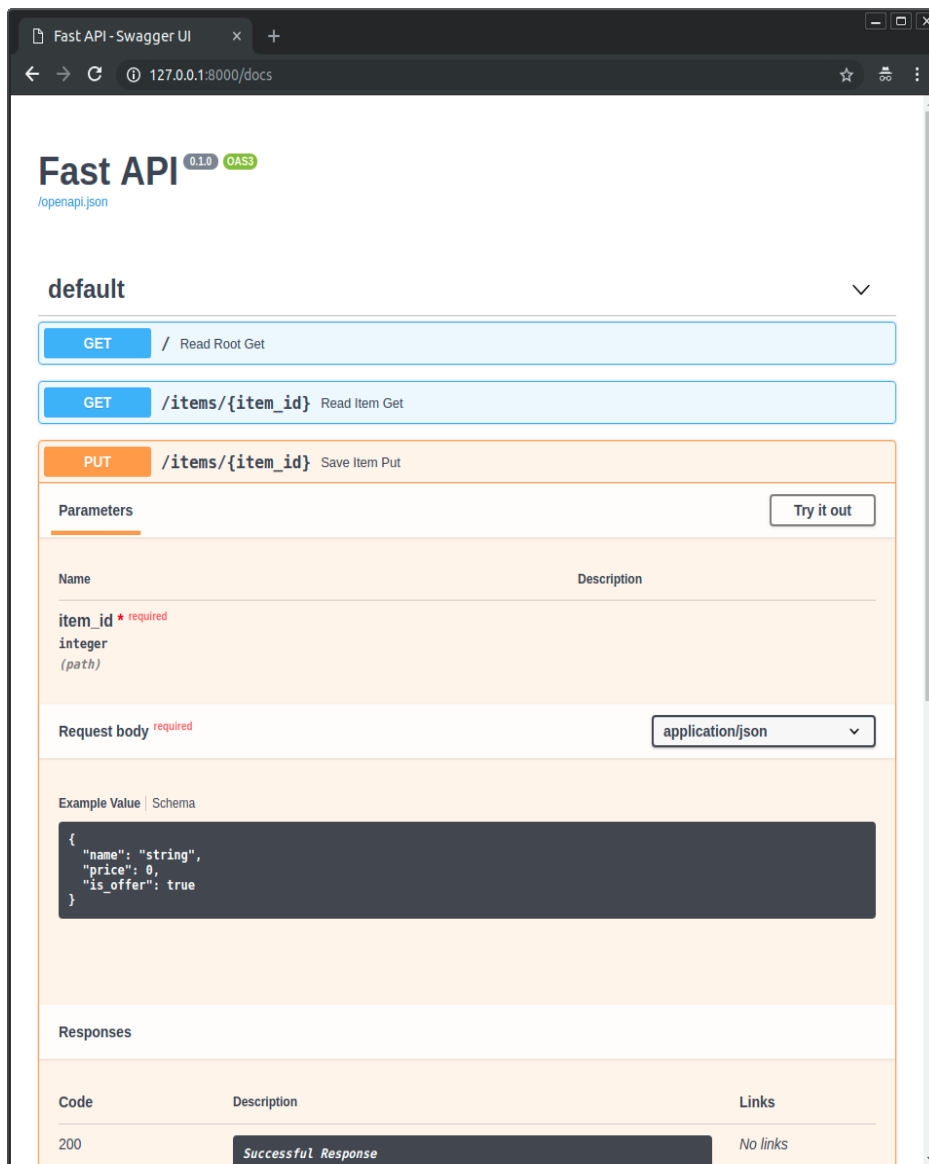


Рисунок 3.2 – Автоматична інтерактивна API документація FastAPI

Використання бібліотеки Pydantic інтегровано у FastAPI для суворої валідації та серіалізації даних. Це підвищує надійність застосунків і мінімізує помилки при обробці вхідних та вихідних структур.

Приклад коду FastAPI-застосунку зображено на рисунку 3.3 [37].

```

1  from fastapi import FastAPI
2  from pydantic import BaseModel
3
4  app = FastAPI()
5
6
7  class Item(BaseModel):
8      name: str
9      price: float
10     is_offer: bool = None
11
12
13 @app.get("/")
14 def read_root():
15     return {"Hello": "World"}
16
17
18 @app.get("/items/{item_id}")
19 def read_item(item_id: int, q: str = None):
20     return {"item_id": item_id, "q": q}
21
22
23 @app.put("/items/{item_id}")
24 def save_item(item_id: int, item: Item):
25     return {"item_name": item.pr, "item_id": item_id}
26

```

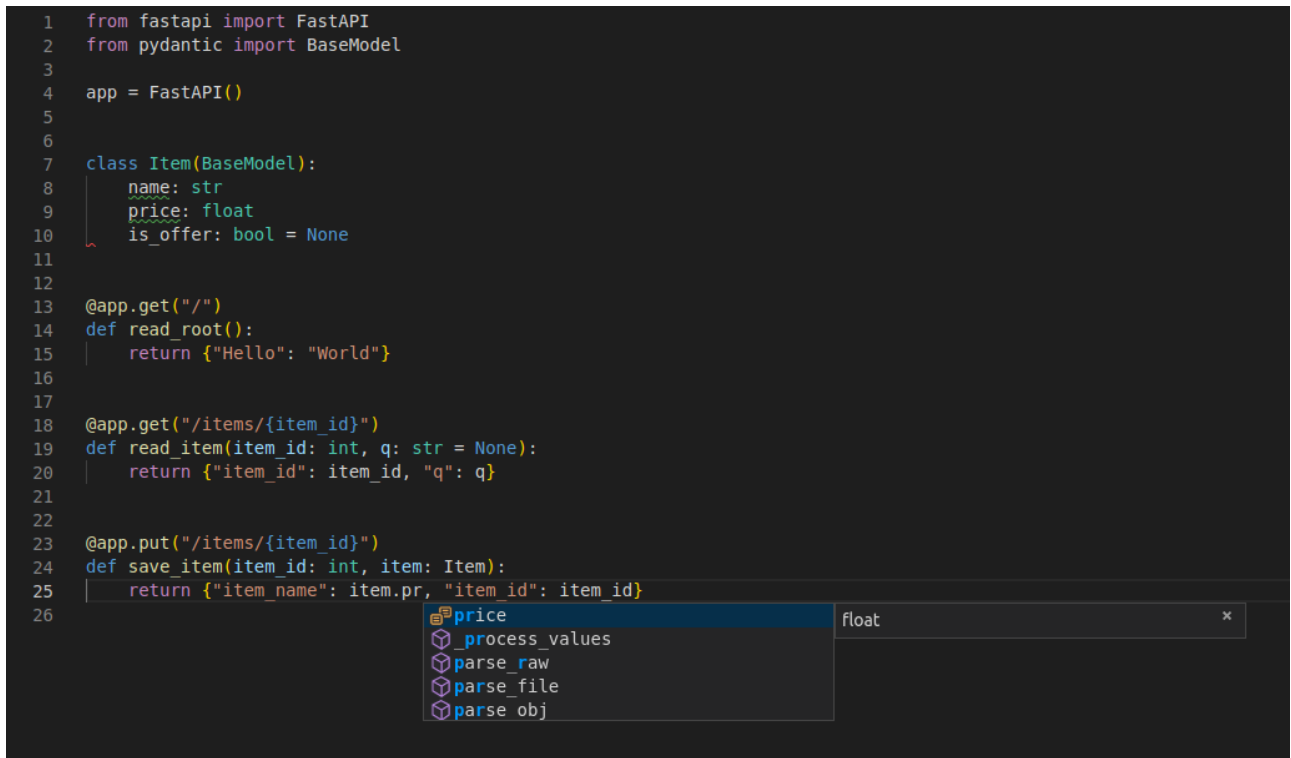


Рисунок 3.3 – Приклад коду точки входу у FastAPI-застосунок

FastAPI ідеально підходить для Sliding Window Context та Summarized Memory, оскільки ці методи вимагають мінімального часу відгуку при отриманні контексту з бази даних і швидкої передачі його до LLM. Він також чудово підходить і для RAG, оскільки його асинхронні можливості дозволяють ефективно керувати паралельними операціями: очікуванням відповіді від бази даних (векторний пошук) та очікуванням відповіді від LLM.

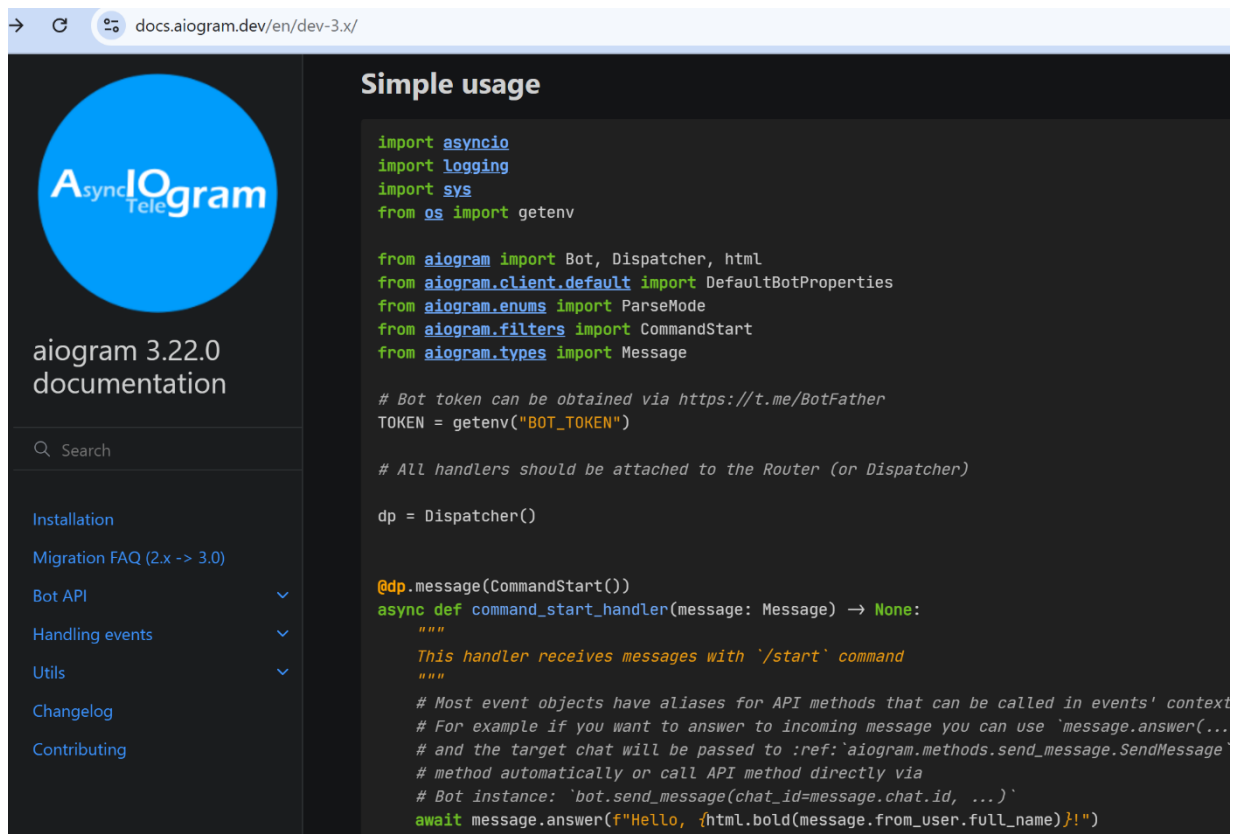
Для реалізації застосунку багатокomпонентної гібридної методики був обраний асинхронний фреймворк aiogram (для Telegram Bot API). Цей вибір обґрунтований вимогами до кінцевої платформи, необхідністю мінімізації архітектурної складності на рівні інтерфейсу користувача та забезпеченням умов для комплексного тестування.

Вибір aiogram замість традиційного вебфреймворку для гібридної методики базується на архітектурних та прикладних перевагах.

Застосування aiogram дозволяє інтегрувати складну бізнес-логіку (чотирирівнева пам'ять, RAG-запити, оновлення відносин) без необхідності розроблення окремого фронтенду (вебінтерфейсу), оскільки API-документації

може бути замало для повноцінного та зручного тестування складної архітектури, а розроблення фронтенду займає певний час та є вичерпним у рамках дослідження. Весь інтерфейс користувача забезпечується функціоналом Telegram (повідомленнями, кнопками, командами), що суттєво зменшує обсяг проектування та реалізації.

Програмний код чат-боту на aiogram зображено на рисунку 3.4 [38].



The image shows a screenshot of the aiogram 3.22.0 documentation page. The page title is "Simple usage". On the left side, there is a navigation menu with the aiogram logo and links for "Installation", "Migration FAQ (2.x -> 3.0)", "Bot API", "Handling events", "Utils", "Changelog", and "Contributing". The main content area displays a Python code snippet for setting up a simple chat bot.

```

import asyncio
import logging
import sys
from os import getenv

from aiogram import Bot, Dispatcher, html
from aiogram.client.default import DefaultBotProperties
from aiogram.enums import ParseMode
from aiogram.filters import CommandStart
from aiogram.types import Message

# Bot token can be obtained via https://t.me/BotFather
TOKEN = getenv("BOT_TOKEN")

# All handlers should be attached to the Router (or Dispatcher)

dp = Dispatcher()

@dp.message(CommandStart())
async def command_start_handler(message: Message) → None:
    """
    This handler receives messages with `/start` command
    """
    # Most event objects have aliases for API methods that can be called in events' context
    # For example if you want to answer to incoming message you can use `message.answer(...`
    # and the target chat will be passed to :ref:`aiogram.methods.send_message.SendMessage`
    # method automatically or call API method directly via
    # Bot instance: `bot.send_message(chat_id=message.chat.id, ...)`
    await message.answer(f"Hello, {html.bold(message.from_user.full_name)}!")

```

Рисунок 3.4 – Приклад програмного коду точки входу у чат-бот на aiogram

Таке рішення дозволяє сконцентрувати основні зусилля на рівні бізнес-логіки, де реалізуються наукова новизна та практична цінність дослідження.

Оскільки aiogram є асинхронним фреймворком, це критично важливо, так як гібридна методика теж є ресурсномісткою (векторизація, семантичний пошук у БД, аналіз відносин за допомогою ШІ-агента). Асинхронність дозволяє ефективно керувати паралельними операціями введення/виведення, такими як мережеві запити до LLM або очікування відповіді від векторної БД, що значно підвищує швидкість реакції чат-боту.

На відміну від традиційних REST API, де детальний опис документації (OpenAPI) є обов'язковим, aiogram забезпечує прямий канал для тестування складної логіки. Тобто, тестування проводиться через реальні діалогові сценарії зі зручним UI. Це дозволяє безпосередньо оцінити ефективність гібридної методики в умовах, наближених до кінцевого використання, що є більш інформативним, ніж тестування окремих API-ендпоінтів.

Таким чином, вибір aiogram є архітектурно обґрунтованим рішенням, яке забезпечує необхідний баланс між складністю внутрішньої бізнес-логіки та простотою її кінцевого впровадження та тестування.

3.1.2 Вибір IDE для програмної реалізації поставлених задач

Для реалізації усіх програмних застосунків обрано PyCharm. Вибір, насамперед, ґрунтується на вимогах проекту до якості програмного коду та підтримки складної архітектури.

PyCharm, як спеціалізоване IDE для Python, надає критично важливі функції, котрі оптимізують процес розроблення та дослідження.

PyCharm забезпечує статичний аналіз коду, виявляючи потенційні помилки та невідповідності стилю до виконання програми. Це значно підвищує надійність програмного забезпечення, що є життєво важливим при роботі зі складними багат шаровими архітектурами та асинхронним кодом. Функції безпечного рефакторингу дозволяють змінювати архітектурні елементи, мінімізуючи ризик внесення помилок.

Приклад сильних сторін PyCharm з точки зору зручності зображено на рисунку 3.5 [39].

IDE надає потужну підтримку для управління залежностями (virtual environments), що є важливим при роботі з великою кількістю бібліотек, характерних для LLM-розроблення (наприклад, для векторизації, роботи з БД та самої інтеграції LLM).

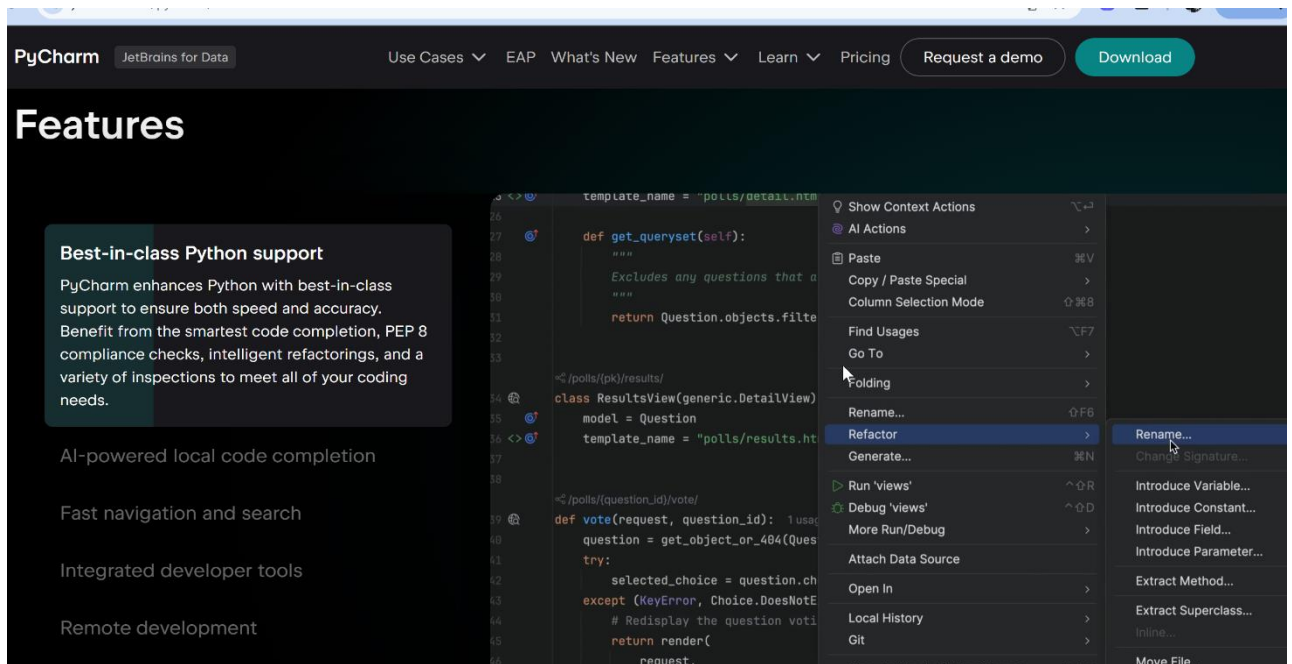


Рисунок 3.5 – Опис функціоналу PyCharm з офіційної документації

Вибір PyCharm, таким чином, є архітектурно обґрунтованим рішенням, що забезпечує найвищу якість, швидкість розроблення та ефективно налагодження складних програмних застосунків.

3.1.3 Вибір СУБД для програмної реалізації поставлених задач

Як СУБД обрано PostgreSQL разом із хмарною платформою Neon.tech. Така комбінація є технічно обґрунтованим рішенням для програмної реалізації всіх поставлених задач.

Вибір безпосередньо зумовлений вимогами методів, котрі поєднують традиційне управління структурованими даними із передовими технологіями векторного пошуку.

PostgreSQL є однією з небагатьох реляційних СУБД, яка завдяки розширенню pgvector ефективно підтримує векторне зберігання та пошук. Це критично важливо для реалізації RAG та гібридного методу, де потрібен високоточний семантичний пошук по векторизованим полям.

Це дозволяє використовувати єдину, уніфіковану базу даних як для реляційних, так і для векторних операцій, що спрощує архітектуру.

Завдяки підтримці типу даних JSONB, PostgreSQL дозволяє ефективно зберігати неструктуровані метадані, що є важливим для гібридного підходу.

Neon.tech є хмарним, безсерверним рішенням для PostgreSQL, яке пропонує чимало переваг. Оскільки інтенсивність використання системи може змінюватися (залежно від кількості користувачів та обсягу RAG-запитів), Neon забезпечує автоматичне масштабування ресурсів. Це гарантує, що векторний пошук (найбільш ресурсомісткий етап) завжди буде виконуватися швидко, а в періоди низької активності витрати будуть мінімальними.

Neon відокремлює обчислювальні ресурси від сховища. Це дозволяє швидко створювати клони БД для цілей тестування та розроблення без дублювання всього обсягу даних, що є важливим для дослідження.

Хмарне рішення Neon забезпечує високу доступність, що є критичним для застосунку на базі Telegram (aiogram), який вимагає постійного підключення. Приклад з офіційної документації Neon щодо підключення хмарної БД до застосунків на Python показано на рисунку 3.6 [40].

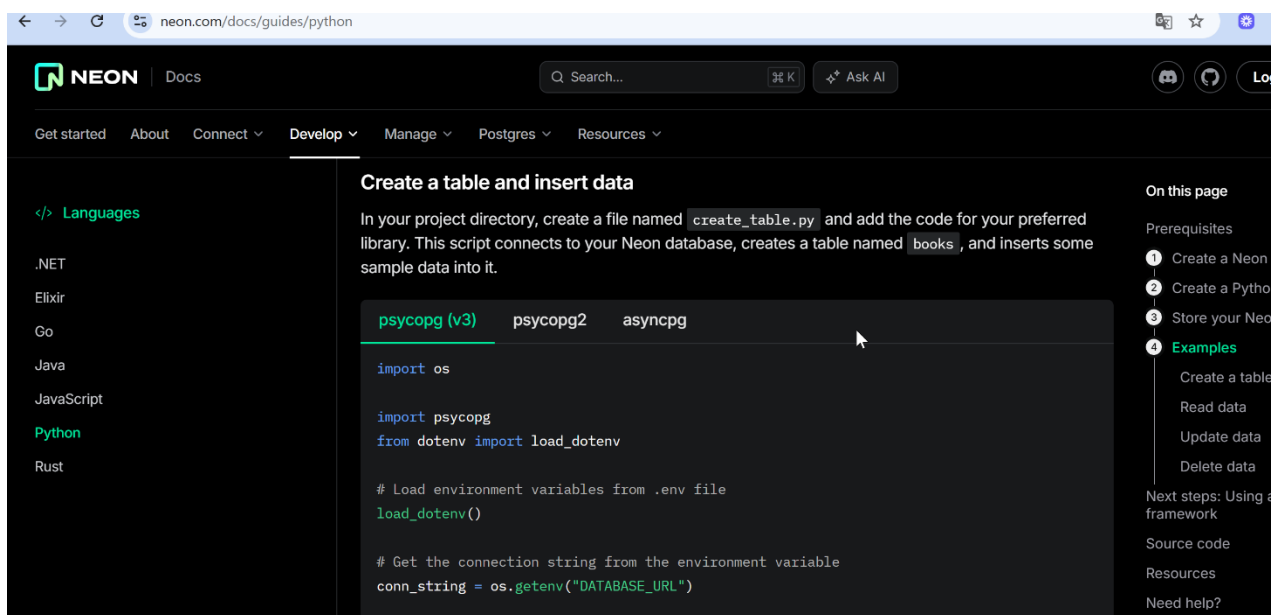


Рисунок 3.6 – Приклад з офіційної документації Neon щодо підключення хмарної бази даних до застосунків на Python

Таким чином, комбінація PostgreSQL (як потужного інструменту з векторною підтримкою) та Neon.tech (як гнучкої та масштабованої хмарної інфраструктури) є найкращим вибором для програмної реалізації усіх розглянутих методів покращення контекстної пам'яті.

3.2 Етапи програмної реалізації методів покращення контекстної пам'яті чат-боту на основі великих мовних моделей

3.2.1 Програмна реалізація Sliding Window Context та Summarized (Episodic) Memory методів

Програмна реалізація методів Sliding Window Context та Summarized (Episodic) Memory здійснена в модулі memory.py, котрий інкапсулює всю логіку управління контекстом [41].

Основу методу Sliding Window Context реалізовано функцією `_get_sliding_window_context`. Цей метод є найпростішим і забезпечує підтримку короткострокової пам'яті. Реалізацію функції наведено у лістингу 3.1.

Лістинг 3.1 Реалізація функції `_get_sliding_window_context`:

```

async def _get_sliding_window_context(
    self,
    db: Session,
    chat_id: int
) -> Dict[str, Any]:
    messages = db.query(Message).filter(
        Message.chat_id == chat_id

).order_by(desc(Message.created_at)).limit(self.sliding_window_size).all()
    messages = list(reversed(messages))
    context_messages = []

```

```

for msg in messages:
    character_name = msg.character.name if msg.character else "System"
    context_messages.append({
        "role": "user" if msg.character and
msg.character.is_user_controlled else "assistant",
        "content": f"{character_name}: {msg.content}",
        "character_name": character_name,
        "message_id": msg.id
    })
return {
    "messages": context_messages,
    "total_messages": len(messages),
    "mode": "sliding",
    "summaries": None
}

```

Розмір контекстного вікна встановлюється константою `self.sliding_window_size` (за замовчуванням 10 повідомлень).

Вибірка даних здійснюється завдяки прямому запиту до таблиці `Message`. Ключовим елементом є використання оператора `order_by(desc(Message.created_at))` та функції `limit(...)`. Це гарантує, що запит поверне лише N найновіших повідомлень, що є основою архітектури «ковзного вікна».

Оскільки повідомлення повертаються у зворотному хронологічному порядку (від найновішого до найстарішого), застосовується функція `list(reversed(messages))` для відновлення послідовності.

`Summarized (Episodic) Memory` метод реалізовано функцією `_get_episodic_context` та допоміжними функціями для генерації узагальнень.

Реалізацію функції `_get_episodic_context` наведено у лістингу 3.2.

Лістинг 3.2 Реалізація функції `_get_episodic_context`:

```

async def _get_episodic_context(
    self,
    db: Session,
    chat_id: int
) -> Dict[str, Any]:
    total_messages = db.query(Message).filter(Message.chat_id ==
chat_id).count()

    await self._check_and_create_summary(db, chat_id, total_messages)
    summaries = db.query(Summary).filter(
        Summary.chat_id == chat_id
    ).order_by(desc(Summary.created_at)).limit(self.episodic_summary_count).all
()

    recent_messages = db.query(Message).filter(
        Message.chat_id == chat_id
    ).order_by(desc(Message.created_at)).limit(self.episodic_recent_messages).all
()

    recent_messages = list(reversed(recent_messages))
    context_messages = []
    for summary in reversed(summaries):
        context_messages.append({
            "role": "system",
            "content": f"[Previous Events Summary]:
{summary.summary_text}",
            "summary_id": summary.id
        })
    for msg in recent_messages:
        character_name = msg.character.name if msg.character else "System"
        context_messages.append({
            "role": "user" if msg.character and
msg.character.is_user_controlled else "assistant",

```

```

        "content": f"{character_name}: {msg.content}",
        "character_name": character_name,
        "message_id": msg.id
    })
return {
    "messages": context_messages,
    "total_messages": total_messages,
    "mode": "episodic",
    "summaries": [
        {
            "id": s.id,
            "summary_text": s.summary_text,
            "message_range": s.message_range,
            "created_at": s.created_at
        } for s in summaries
    ]
}

```

Перед формуванням контексту система перевіряє, чи не перевищено порогове значення `self.episodic_summary_threshold` (за замовчуванням 15 повідомлень) з моменту останнього узагальнення [42].

Функція складно працює з полем `message_range`, щоб коректно визначити `max_covered_id` (останній ID повідомлення, включений в попереднє узагальнення).

Якщо поріг досягнуто, викликається функція `_create_summary`.

Реалізацію функції `_check_and_create_summary` наведено у лістингу 3.3.

Лістинг 3.3 Реалізація функції `_check_and_create_summary`:

```

async def _check_and_create_summary(
    self,

```

```

db: Session,
chat_id: int,
total_messages: int
) -> Optional[Summary]:
    summaries = db.query(Summary).filter(Summary.chat_id ==
chat_id).all()
    max_covered_id = 0
    if summaries:
        for summary in summaries:
            if summary.message_range:
                try:
                    if hasattr(summary.message_range, 'upper'):
                        # It's a Range object from PostgreSQL
                        end = summary.message_range.upper - 1
                    else:
                        range_str = str(summary.message_range).strip("[ ]")
                        start, end = map(int, range_str.split(", "))
                        max_covered_id = max(max_covered_id, end)
                except Exception as e:
                    print(f"Error parsing message range: {e}")
                    continue
            max_message_result = db.query(Message.id).filter(Message.chat_id ==
chat_id).order_by(desc(Message.id)).first()
            actual_max_id = max_message_result[0] if max_message_result else 0
            messages_since_summary = actual_max_id - max_covered_id
            if messages_since_summary >= self.episodic_summary_threshold:
                latest_summary = None
            if summaries:
                latest_summary = max(summaries, key=lambda s: (
                    s.message_range.upper if hasattr(s.message_range, 'upper')

```

```

        else int(str(s.message_range).strip("[]").split(",")[1])
    ) if s.message_range else 0)
    return await self._create_summary(db, chat_id, latest_summary)
return None

```

Генерація узагальнення реалізована функцією `_create_summary`. У ній визначається діапазон повідомлень для узагальнення, починаючи з першого непідсумованого ID (`start_id`) до кількості, визначеної порогом.

Повідомлення готуються для LLM, ініціюючи виклик до сервісу із штучного інтелекту: `summary_result = await ai_service.generate_summary(...)`.

Сформований текст узагальнення та діапазон повідомлень зберігаються у таблиці `Summary`.

Також для реалізації `Summarized (Episodic) Memory` використовується зовнішній виклик LLM для стиснення контексту.

Усі повідомлення, які необхідно узагальнити, об'єднуються в єдиний текстовий блок. Створюється системний промпт, котрий чітко інструктує LLM про вимоги до узагальнення (включення ключових подій, обмеження на 200 слів).

Далі здійснюється асинхронний HTTP-запит до сервісу LLM. Для цієї задачі використовується низькотемпературний режим (`"temperature": 0,3`), котрий сприяє генерації більш фактичних і менш креативних узагальнень.

Таким чином, програмна реалізація чітко відображає архітектурні відмінності: `Sliding Window Context` використовує просте обмеження запиту, тоді як `Summarized (Episodic) Memory` додає логіку узагальнення та асинхронну інтеграцію із зовнішнім ШІ-сервісом.

3.2.2 Програмна реалізація RAG методу

Програмна реалізація RAG методу базується на інтеграції семантичного пошуку в процес генерації відповіді. Ключова логіка розподілена між модулями `chat_service.py` (бізнес-логіка) та `ai_client.py` (робота з `embeddings`).

Першим і критично важливим кроком у RAG є перетворення тексту на векторне подання, це реалізовано функцією `get_embedding`.

Лістинг 3.4 Реалізація функції `get_embedding`:

```
def get_embedding(text: str):
    # openai.Embedding.create
    resp = openai.Embedding.create(model=EMBEDDING_MODEL,
input=text)
    return resp["data"][0]["embedding"]
```

Функція `get_embedding` використовує зовнішній API OpenAI для генерації векторного подання вхідного тексту. Цей вектор є високорозмірним числовим масивом, котрий фіксує семантичний зміст тексту. Векторизація відбувається як для запиту користувача, так і для відповідей ШІ, що дозволяє обом типам даних брати участь у подальшому пошуку.

Функція `send_message_and_get_ai` реалізує послідовність RAG-операцій.

Процес пошуку (Retrieval) складається з трьох ключових кроків:

Крок 1. Векторизація запиту: запит користувача `user_text` негайно векторизується за допомогою `q_emb = get_embedding(user_text)`.

Крок 2. Семантичний пошук (`retrieve_context`): вектор запиту використовується для пошуку в БД релевантного контексту.

Крок 3. Формування контексту: знайдені повідомлення об'єднуються у змінну `context_text`, котра потім буде додана до промпту LLM.

У лістингу 3.5 показано, як здійснюється прямий SQL-запит до PostgreSQL з розширенням `pgvector`: оператор `<#>` позначає косинусну відстань (яка

використовується для вимірювання схожості між векторами), а запит повертає k (за замовчуванням 5) повідомлень, котрі є найбільш семантично схожими на запит користувача.

Лістинг 3.5 Реалізація функції `retrieve_context`:

```
def retrieve_context(db, query_embedding, k: int = 5):
    emb_str = "[" + ",".join(str(x) for x in query_embedding) + "]"
    sql = f"""
        SELECT content
        FROM messages
        WHERE embedding IS NOT NULL
        ORDER BY embedding <#> '{emb_str}':vector
        LIMIT :k
    """
    rows = db.execute(text(sql), {"k": k}).fetchall()
    return [r[0] for r in rows]
```

Функція `generate_with_openrouter` відправляє підкріплені контекстом (`context_text`) промпти разом до LLM. Інтеграція знайденого контексту забезпечує релевантність відповіді на достовірних даних. Функція `generate_with_openrouter` показана у лістингу 3.6.

Лістинг 3.6 Реалізація функції `generate_with_openrouter`:

```
def generate_with_openrouter(system_prompt: str, user_prompt: str, context:
str = "") -> str:
    headers = {
        "Authorization": f"Bearer {OPENROUTER_API_KEY}",
        "Content-Type": "application/json"
    }
    # system prompt
```

```

system_content = system_prompt
if context:
    system_content += "\n\nRelevant context to use:\n" + context
...

```

3.2.3 Програмна реалізація гібридного методу

Програмна реалізація гібридної методики основана на багатошаровій, асинхронній системі, де пам'ять розділена на чотири динамічно взаємодіючі рівні.

Центральним компонентом, котрий і реалізує логіку об'єднання усіх рівнів пам'яті, є клас `ContextRetriever`. Різні типи пам'яті мають різну важливість. Це реалізовано через `smart thresholds` (розумні пороги).

Метод `retrieve_context` викликає гібридний пошук. Тобто, система використовує семантичний пошук (через вектори) для динамічної пам'яті та повний вибір для статичної пам'яті. Функція координує векторизацію запиту та делегує гібридний пошук до `VectorStore`.

Фрагменти реалізації класу `ContextRetriever` та його методу `retrieve_context` наведено в лістингах 3.7 та 3.8.

Лістинг 3.7 Реалізація класу `ContextRetriever`:

```

class ContextRetriever(LoggerMixin):
    def __init__(self, embedding_model: Optional[str] = None):
        self.embedding_model = embedding_model
        self.character_threshold = 0.01
        self.world_threshold = 0.01
        self.message_threshold = 0.7
        self.recent_priority = 10.
        self.max_retrieved_chunks = MAX_RETRIEVED_CHUNKS

```

Лістинг 3.8 Фрагмент реалізації методу `retrieve_context`:

```
# ... (inside retrieve_context)
async with EmbeddingGenerator(self.embedding_model) as embedder:
    query_embedding = await embedder.generate_embedding(query)
# ...
async with async_session() as session:
    async with VectorStore(session) as vector_store:
        search_results = await vector_store.hybrid_search(
            query_embedding=query_embedding,
            # ...
            character_threshold=self.character_threshold,
            world_threshold=self.world_threshold,
            message_threshold=self.message_threshold
        )
# ...
```

Контекстна пам'ять моделюється як структурований стан, котрий не залежить від векторного пошуку, але має найвищий пріоритет для LLM.

Динаміка відносин між персонами чат-боту реалізується через метод `retrieve_character_relationships` і програмний механізм забезпечує, що контекст завжди включає поточний стан взаємодій.

Використання `RelationshipManager` (класу, відповідального за маніпуляції з таблицею `character_relationships`) інкапсулює логіку SQL та обробку JSONB-полів (`current_dynamic`, `development_history`).

Для контексту замість сирих даних відносин використовується метод `get_relationship_summary_for_character`. Тобто, менеджер виконує стиснення графа відносин у єдиний, лаконічний текстовий опис, зменшуючи навантаження на контекстне вікно фінального промпта.

Присвоєння `similarity_score=18.0` є програмним трюком. Це значення є штучно високим (перевищує $\leq 1,0$ для векторного пошуку) та гарантує, що цей елемент (а також арки та емоції) завжди буде серед пріоритетних результатів, незалежно від семантичної схожості.

Ключовий програмний код методу `retrieve_character_relationships` наведено у лістингу 3.9.

Лістинг 3.9 Фрагмент методу `retrieve_character_relationships`:

```
async def retrieve_character_relationships(...):
    # ...
    from core.relationship_manager import RelationshipManager
    # ...
    relationships = await
relationship_manager.get_character_relationships(chat_id, character_id)
    # ...
    summary = await
relationship_manager.get_relationship_summary_for_character(chat_id,
character_id)
    return [RetrievalResult(
        entity_type="character_relationships",
        content=summary,
        similarity_score=18.0,
        # ...
    )]
```

Аналогічно до цього, функції `retrieve_story_arcs` та `retrieve_emotional_state` таким же чином витягують структуровані дані.

Клас `RelationshipAnalysisAgent` є програмною реалізацією автоматичного навчання та оновлення динамічних елементів.

Агент запускається після кожного діалогового обміну (повідомлення користувача + відповідь ШІ). Його мета – визначити, як ця взаємодія вплинула на `character_relationships` та `current_emotional_state`. Агент не просто аналізує останнє повідомлення. Він викликає `_build_full_context`, отримуючи доступ до всіх шарів пам'яті (RAG, відносини, арки). Це дозволяє LLM-агенту (використовуючи LLM) приймати обґрунтовані рішення про зміну відносин, розуміючи загальний сюжетний та емоційний фон.

Фінальний крок `_apply_relationship_updates` зберігає зміни, фактично забезпечуючи самооновлення контекстної пам'яті. Це гарантує, що пам'ять є динамічною і еволюціонує з кожним новим діалоговим кроком.

Фрагменти реалізації агента наведено у лістингах 3.10 та 3.11.

Лістинг 3.10 Клас `RelationshipAnalysisAgent`:

```
class RelationshipAnalysisAgent:
    def __init__(self):
        self.logger = logger
        self.context_retriever = ContextRetriever()
        self.analysis_models = [
            "google/gemini-2.5-flash",
            ... ]
        self.current_model_index = 0
```

Лістинг 3.11 Фрагмент методу `analyze_interaction`:

```
async def analyze_interaction(...):
    full_context = await self._build_full_context(...)
    analysis_prompt = await self._build_analysis_prompt(...)
    # ...
    parsed_analysis = await self._parse_analysis_response(analysis_response)
    update_results = await self._apply_relationship_updates(
```

```

    chat_id=chat_id,
    analysis=parsed_analysis,
    message_id=message_id
)
# ...

```

3.3 Порівняльний аналіз методів покращення контекстної пам'яті чат-боту на основі великих мовних моделей

3.3.1 Підготовка тестових даних для тестування методів покращення пам'яті

Для ретельного тестового покриття буде створено тестові дані на основі рольового сценарію, котрий продемонструє якість методів покращення пам'яті.

Тестування повинно бути прозорим, тому буде створено простий та лінійний сценарій, котрий вимагатиме логічної згадки фактів, які були введені на самому початку чату [43].

У таблицях 3.1–3.4 наведено опис сценарію, ключові аспекти та очікувані результати тестування за критеріями зв'язності.

Таблиця 3.1 – Аспекти тестового сценарію

Аспект	Опис	Функція
1	2	3
Сюжет	19 століття, Англія. Слідчий їде у візниці для розслідування пожежі в особняку Лорда Керролла та крадіжки родинного Талісмана (який насправді є картою скарбів).	Створення даних для тестування методів покращення пам'яті.

Продовження таблиці 3.1

1	2	3
Етапи сценарію	<p>Сценарій охоплює етапи:</p> <p>Етап 1. Поїздка з візником, який має унікальні ознаки.</p> <p>Етап 2. Самостійне розслідування, яке має «змістити» фокус мовної моделі та «загубити» факти з першого етапу.</p> <p>Етап 3. Фінальний поворот, коли свідок дає опис злочина, котрий відповідає візнику.</p>	<p>Перевірка здатності кожного методу зв'язувати давно забуті деталі з початку чату (ознаки візника) з критичним свідченням у кінці (опис злочина).</p>
Ключовий зв'язок (очікуваний результат)	<p>LLM отримує інформацію про ознаки візника і має використати її у своїй реакції, коли свідок згадає ці ж деталі в кінці сценарію.</p>	<p>Перевірка кожного методу на очікуваний результат.</p>

Таблиця 3.2 – Рольові персони та роль у сценарію

Персона	Мета у сценарії	Роль
1	2	3
Слідчий	Збір фактів на початку тестування та згадування великою мовною моделлю наприкінці із логічними висновками	1, 2 етапи – користувач. 3 етап – LLM
Візник	Подання ключових фактів	Користувач
Свідок (економка лорда Керролла)	Натяк у кінці сценарію на ключові факти, котрі були згадані на початку	Користувач
Лорд Керролл (потерпілий)	Фонова персона для збору фактів	Згадується у чаті

Таблиця 3.3 – Ключові факти та логічний зв'язок

Ключовий факт	Згадка свідка	Зв'язок
1	2	3
Візник має пов'язку на оці	У злочинця гетерохромія (очі різного кольору)	Візник носить пов'язку, щоб приховати ключову деталь зовнішності
Візник говорить німецькою мовою в емоційному стані	Злочинець закричав німецькою, коли побачив свідка під час злочину	Співпадіння ознак
Злочинець знав про розташування кімнат та таємницю талісману	Лорд у стані алкогольного оп'яніння їхав візницею 2 тижні тому, та візник заносив його додому	Візник дізнався про талісман зі слів лорда, адже той був у неадекватному стані, а також про розташування кімнат, коли допомагав дійти до маєтку

Таблиця 3.4 – Етапи тестування та діапазон повідомлень

№	Діапазон повідомлень	Опис
1	2	3
1	{1–6}	Поїздка з візником, який має унікальні ознаки. Модель отримує ключову інформацію.
2	{7–59}	Самостійне розслідування, котре має «змістити» фокус мовної моделі та «загубити» факти з першого етапу.
3	{60–63}	Фінальний поворот, коли свідок дає опис злодія, котрий відповідає візнику. Модель має «згадати» та «зіставити» факти із першого етапу.

Тестування буде вважатися успішно пройденим, якщо:

- мовна модель у згенерованій відповіді на факти із третього етапу «згадає» ключові факти із першого етапу;
- мовна модель логічно «пов’яже» між собою факти із першого та третього етапу;
- мовна модель зробить логічні висновки.

Очікувані результати оформлені у таблиці 3.5.

Таблиця 3.5 – Факти з третього етапу та очікувані результати

Факт	Реакція LLM (очікуваний результат)
1	2
У злочинця гетерохромія	LLM «згадала» про пов’язку на оці у візника
	LLM «зіставила» пов’язку та гетерохромію
Злочинець говорить німецькою в емоційному стані	LLM «згадала», що візник кричав на німецькій мові
	LLM «зіставила» співпадіння
Злочинець знав про розташування кімнат та таємницю талісману	LLM «врахувала» неадекватний стан лорда та можливість розказати таємницю візнику
	LLM «зіставила», що якщо візник допомагав дійти до дому лорду, то бачив розташування кімнат
Лорд їхав візницею додому 2 тижні тому	LLM робить висновок, що то був той самий візник

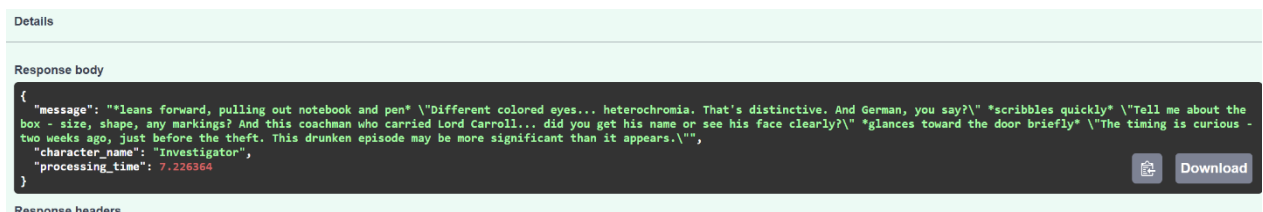
Якщо у згенерованій відповіді LLM згадає та зіставить перелічені логічні висновки, то пройде тестування на довготривалу пам’ять.

3.3.2 Тестування методів покращення контекстної пам'яті чат-боту

Оскільки для усіх методів тестовим набором є повідомлення, створено єдиний набір тестових повідомлень у форматі JSON (лістинг А.1 наведено у додатку А). Набір включає 63 повідомлення, котрі охоплюють усі тестові етапи.

Ефективність методів за очікуваними результатами буде оцінено у ключовому повідомленні, котрий LLM згенерує у відповідь. Семантичний зміст повідомлення буде оцінено на співпадіння до очікуваних результатів.

На рисунках 3.7–3.10 наведено результати згенерованих LLM-відповідей, а також у лістингах Б.1–Б.4 у додатку Б.

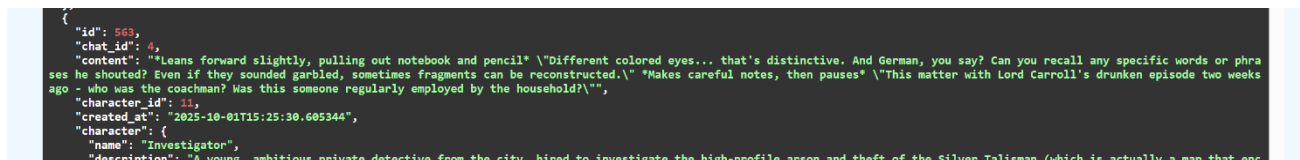


```

{
  "message": "*Leans forward, pulling out notebook and pen* \"Different colored eyes... heterochromia. That's distinctive. And German, you say?\" *scribbles quickly* \"Tell me about the box - size, shape, any markings? And this coachman who carried Lord Carroll... did you get his name or see his face clearly?\" *glances toward the door briefly* \"The timing is curious - two weeks ago, just before the theft. This drunken episode may be more significant than it appears.\"",
  "character_name": "Investigator",
  "processing_time": 7.226364
}

```

Рисунок 3.7 – Відповідь LLM (Sliding Window Context метод)

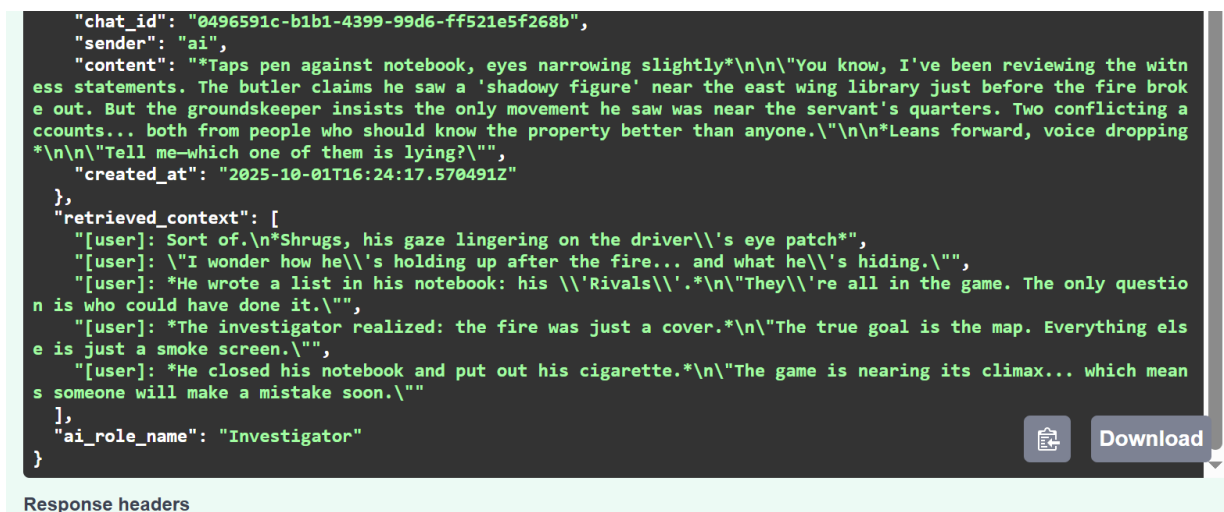


```

{
  "id": 569,
  "chat_id": 4,
  "content": "*Leans forward slightly, pulling out notebook and pencil* \"Different colored eyes... that's distinctive. And German, you say? Can you recall any specific words or phrases he shouted? Even if they sounded garbled, sometimes fragments can be reconstructed.\" *Makes careful notes, then pauses* \"This matter with Lord Carroll's drunken episode two weeks ago - who was the coachman? Was this someone regularly employed by the household?\"",
  "character_id": 11,
  "created_at": "2025-10-01T15:25:30.605344",
  "character": {
    "name": "Investigator",
    "description": "A young, ambitious private detective from the city, hired to investigate the high-profile arson and theft of the Silver Talisman (which is actually a map that enc"
  }
}

```

Рисунок 3.8 – Відповідь LLM (Summarized (Episodic) Memory метод)



```

{
  "chat_id": "0496591c-b1b1-4399-99d6-ff521e5f268b",
  "sender": "ai",
  "content": "*Taps pen against notebook, eyes narrowing slightly*\n\n\"You know, I've been reviewing the witness statements. The butler claims he saw a 'shadowy figure' near the east wing library just before the fire broke out. But the groundskeeper insists the only movement he saw was near the servant's quarters. Two conflicting accounts... both from people who should know the property better than anyone.\"\n\n*Leans forward, voice dropping*\n\n\"Tell me—which one of them is lying?\"",
  "created_at": "2025-10-01T16:24:17.570491Z",
  "retrieved_context": [
    "[user]: Sort of.\n*Shrugs, his gaze lingering on the driver's eye patch*",
    "[user]: \"I wonder how he's holding up after the fire... and what he's hiding.\"",
    "[user]: *He wrote a list in his notebook: his 'Rivals'.*\"They're all in the game. The only question is who could have done it.\"",
    "[user]: *The investigator realized: the fire was just a cover.*\n\n\"The true goal is the map. Everything else is just a smoke screen.\"",
    "[user]: *He closed his notebook and put out his cigarette.*\n\n\"The game is nearing its climax... which means someone will make a mistake soon.\"",
  ],
  "ai_role_name": "Investigator"
}

```

Рисунок 3.9 – Відповідь LLM (RAG метод)

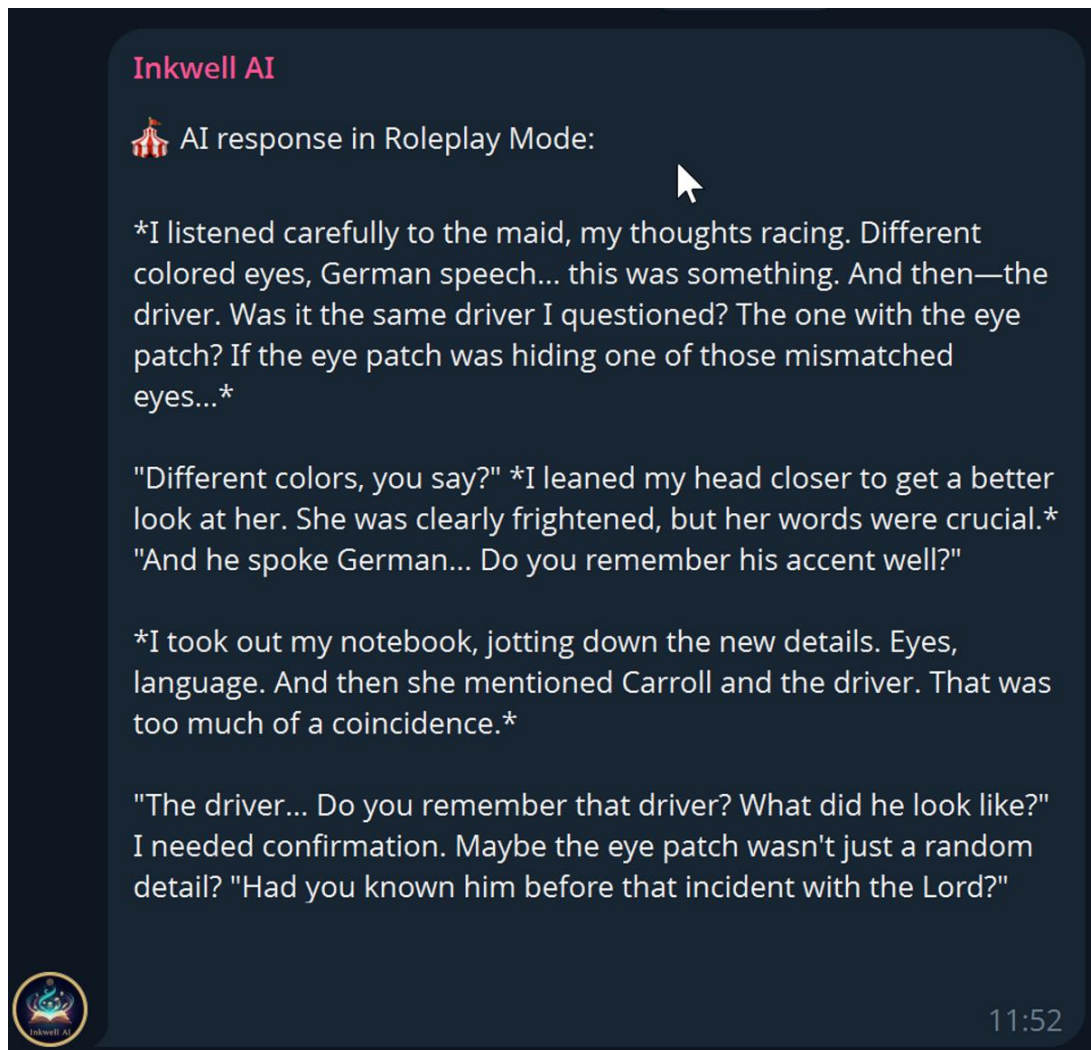


Рисунок 3.10 – Відповідь LLM (гібридний метод)

Переклад українською відповіді LLM (Sliding Window Context):
 «*нахиляється вперед, дістаючи блокнот і ручку* «Різні кольори очей... гетерохромія. Це помітна ознака. І німецька, ви кажете?» *швидко записує* «Розкажіть мені про скриньку – розмір, форма, будь-які позначки? І цей візник, що заносив Лорда Керролла... ви запам'ятали його ім'я або чітко бачили його обличчя?» *коротко кидає погляд у бік дверей* «Час збігу дивний – два тижні тому, якраз перед крадіжкою. Цей п'яний епізод може бути більш значущим, ніж здається.»».

Переклад українською відповіді LLM (Summarized (Episodic) Memory):
 «*Ледь нахиляється вперед, дістаючи блокнот і олівець* «Різні кольори очей... це помітно. І німецька, ви кажете? Ви можете пригадати будь-які конкретні слова

чи фрази, які він виголошував? Навіть якщо вони звучали нерозбірливо, іноді фрагменти можна відтворити.» *Робить обережні нотатки, потім зазначає* «Щодо цієї історії з пияцтвом Лорда Керролла два тижні тому – хто був той візник? Це був хтось, кого регулярно наймали у маєтку?»».

Переклад українською відповіді LLM (RAG): «*Стукає ручкою по блокноту, очі трохи звужуються* «Знаєте, я переглядав свідчення очевидців. Дворецький стверджує, що бачив «тіньову фігуру» біля бібліотеки східного крила безпосередньо перед початком пожежі. Але садівник наполягає, що єдиний рух, який він бачив, був біля службових приміщень. Два суперечливих свідчення... обидва від людей, які мали б знати маєток краще за будь-кого.» *Нахиляється вперед, голос стихає* «Скажіть мені – хто з них бреше?»».

Переклад українською відповіді LLM (гібридний метод): «*Я уважно слухав служницю, мої думки мчали галопом. Різні кольори очей, німецька мова... це вже щось. А потім – візник. Чи це той самий візник, якого я допитував? Той, що з пов'язкою на оці? Якщо пов'язка приховує одне з тих різнокольорових очей...* «Різних кольорів, кажете?» *Я схилив голову, щоб краще її роздивитися. Вона була явно налякана, але її слова були вирішальними.* «І він говорив німецькою... Ви добре запам'ятали його акцент?» *Я дістав свій блокнот, записуючи нові деталі. Очі, мова. А потім вона згадала Керролла і візника. Це було надто великим збігом.* «Візник... Ви пам'ятаєте того візника? Як він виглядав?» *Мені потрібне було підтвердження. Можливо, пов'язка на оці була не просто так?* «Ви були з ним знайомі до того випадку з Лордом?»».

Проаналізуємо відповіді кожного методу.

Аналіз отриманих результатів тестування методу Sliding Window Context наведено у таблиці 3.6.

Метод Sliding Window Context не пройшов тестування на довготривалу пам'ять.

На момент фінального свідчення (Етап 3), початкові ключові факти про пов'язку на оці та німецьку мову візника (Етап 1, повідомлення 1–6) витіснені з контекстного вікна наступними повідомленнями про розслідування (Етап 2).

Таблиця 3.6 – Результати тестування методу Sliding Window Context

Факт	Реакція LLM (очікуваний результат)	Аналіз відповіді	Статус
1	2	3	3
У злочинця гетерохромія	LLM «згадала» про пов'язку на оці у візника	LLM не «згадала» про пов'язку на оці у візника (факт був витіснений з контекстного вікна).	Провалено
	LLM «зіставила» пов'язку та гетерохромію	LLM лише констатує: «Різні кольори очей... гетерохромія.»	Провалено
Злочинець говорить німецькою в емоційному стані	LLM «згадала», що візник кричав на німецькій мові	LLM не згадала про першу згадку німецької візником (факт був витіснений).	Провалено
	LLM «зіставила» співпадіння	LLM лише констатує: «І німецька, ви кажете?»	Провалено
Злочинець знав про розташування кімнат та таємницю талісману	LLM «врахувала» неадекватний стан лорда та можливість розказати таємницю візнику	LLM робить висновок про значущість інциденту: «Цей п'яний епізод може бути більш значущим, ніж здається.» однак не пов'язує факти	Провалено
	LLM «зіставила», що якщо візник допомагав дійти дому лорду, то бачив розташування кімнат	LLM ігнорує	Провалено
Лорд їхав візницею додому 2 тижні тому	LLM робить висновок, що то був той самий візник	LLM хоча і зробила висновок про зв'язок і питає про ім'я: «ви запам'ятали його ім'я або чітко бачили його обличчя?», але це не стосується візника із першого етапу.	Провалено

Модель змогла реагувати лише на лише найближчі факти (візник + п'яний лорд + час перед злочинном), оскільки згадка про візника була лише за кілька повідомлень до фіналу (або ж цей факт був у межах 10-повідомлень останнього вікна, оскільки він згаданий свідком).

Вердикт: Sliding Window Context не забезпечив довготривалу пам'ять для згадки та зв'язування фактів між Етапом 1 і Етапом 3.

Проаналізуємо отримані результати тестування методу Summarized (Episodic) Memory (табл. 3.7).

Таблиця 3.7 – Результати тестування методу Summarized (Episodic) Memory

Факт	Реакція LLM (очікуваний результат)	Аналіз відповіді	Статус
1	2	3	4
У злочинця гетерохромія	LLM «згадала» про пов'язку на оці у візника	LLM не «згадала» про пов'язку на оці у візника	Провалено
	LLM «зіставила» пов'язку та гетерохромію	LLM лише констатує	Провалено
Злочинець говорить німецькою в емоційному стані	LLM «згадала», що візник кричав на німецькій мові	LLM не згадала про німецьку візника з Етапу 1. Саммарі #1 містить про колишнє візника у Мюнхені, але цього недостатньо для зіставлення з криком німецькою.	Провалено
	LLM «зіставила» співпадіння	LLM лише констатує: «І німецька, ви кажете?»	Провалено

Продовження таблиці 3.7

1	2	3	4
Злочинець знав про розташування кімнат та таємницю талісману	LLM «врахувала» неадекватний стан лорда та можливість розказати таємницю візнику	LLM зосереджується на цьому: «Щодо цієї історії з пияцтвом Лорда Керролла два тижні тому – хто був той візник?»	Пройдено
	LLM «зіставила», що якщо візник допомагав дійти дому лорду, то бачив розташування кімнат	LLM ігнорує	Провалено
Лорд їхав візницею додому 2 тижні тому	LLM робить висновок, що то був той самий візник	LLM ігнорує	Провалено

Метод Summarized (Episodic) Memory (навіть із узагальненням) провалив перевірку на пряме зіставлення зовнішніх ознак (гетерохромія / німецька мова), оскільки ключові деталі («кричав німецькою», «пов'язка на оці») були недостатньо чітко відображені у згенерованому узагальненні діапазону перших 15 повідомлень (лістинг Б.5 у додатку Б). Таким чином, факти «загубилися» на етапі узагальнення, так як є дрібними деталями.

Вердикт: Summarized (Episodic) Memory не забезпечив довготривалу пам'ять для згадки та зв'язування фактів між Етапом 1 і Етапом 3.

Проаналізуємо якість отриманих результатів тестування методу RAG (табл. 3.8).

Незважаючи на те, що RAG успішно витягнув ключовий факт («eye patch») у свій контекст, сама мовна модель вирішила їх ігнорувати, концентруючись лише на останній задачі (вирішити протиріччя свідчень), а не на синтезі з довготривалою пам'яттю.

Таблиця 3.8 – Результати тестування методу RAG

Факт	Реакція LLM (очікуваний результат)	Аналіз відповіді	Статус
1	2	3	4
У злочинця гетерохромія	LLM «згадала» про пов'язку на оці у візника	RAG витягнув повідомлення про пов'язку на оці з Етапу 1. Це є прямим «гачком» для зіставлення	Пройдено
	LLM «зіставила» пов'язку та гетерохромію	LLM ігнорує	Провалено
Злочинець говорить німецькою в емоційному стані	LLM «згадала», що візник кричав на німецькій мові	Семантичний пошук RAG не знайшов повідомлення про «Verflucht!» (крик німецькою) як релевантне	Провалено
	LLM «зіставила» співпадіння	Без вихідного факту зіставлення неможливе.	Провалено
Злочинець знав про розташування кімнат та таємницю талісману	LLM «врахувала» неадекватний стан лорда та можливість розказати таємницю візнику	LLM проігнорувала	Провалено
	LLM «зіставила», що якщо візник допомагав дійти дому лорду, то бачив розташування кімнат	LLM ігнорує	Провалено
Лорд їхав візницею додому 2 тижні тому	LLM робить висновок, що то був той самий візник	LLM ігнорує	Провалено

Це демонструє, що RAG покращує доступ до пам'яті, але не гарантує її використання, якщо модель вирішує, що старі факти не є семантично релевантними для конкретної миті діалогу.

Вердикт: RAG не забезпечив довготривалу пам'ять для згадки та зв'язування фактів між Етапом 1 і Етапом 3.

Проаналізуємо отримані результати тестування гібридного методу (табл. 3.9).

Таблиця 3.9 – Результати тестування гібридного методу

Факт	Реакція LLM (очікуваний результат)	Аналіз відповіді	Статус
1	2	3	4
У злочинця гетерохромія	LLM «згадала» про пов'язку на оці у візника	LLM згадала факт про пов'язку на оці («Той, що з пов'язкою на оці?») у внутрішньому монологі.	Пройдено
	LLM «зіставила» пов'язку та гетерохромію	LLM зробила пряме зіставлення у внутрішньому монологі: «Якщо пов'язка приховує одне з тих різнокольорових очей...»	Пройдено
Злочинець говорить німецькою в емоційному стані	LLM «згадала», що візник кричав на німецькій	LLM згадала німецьку мову і зіставила її з візником у монологі («Очі, мова»).	Пройдено
	LLM «зіставила» співпадіння	LLM активно запитує про деталі: «І він говорив німецькою... Ви добре запам'ятали його акцент??»	Пройдено

Продовження таблиці 3.9

1	2	3	4
Злочинець знав про розташування кімнат та таємницю талісману	LLM «врахувала» неадекватний стан лорда та можливість розказати таємницю візнику	LLM робить висновок про надмірну випадковість	Пройдено
	LLM «зіставила», що якщо візник допомагав дійти дому лорду, то бачив розташування кімнат	LLM не робить прямого висновку, але акцентує про співпадіння візника з тим, що допомагав лорду: «вона згадала Керролла і візника»	Частково пройдено
Лорд їхав візницею додому 2 тижні тому	LLM робить висновок, що то був той самий візник	LLM прямо ставить під сумнів особу візника: «Візник... Ви пам'ятаєте того візника? Як він виглядав?» *Мені потрібне було підтвердження. Можливо, пов'язка на оці була не просто так?*»	Пройдено

Гібридний метод успішно пройшов тестування на довготривалу пам'ять за усіма критеріями.

Модель не лише згадала всі три ключові, давно забуті факти (пов'язку, німецьку мову), а ще й логічно зіставила їх у своєму внутрішньому монолозі і використала для формування наступних, дуже критично важливих запитань.

Це підтверджує, що для складних діалогових систем, де потрібна як логіка обставин, так і точність деталей, гібридний підхід є найбільш ефективним рішенням.

Вердикт: гібридний метод забезпечив довготривалу пам'ять для згадки та зв'язування фактів між Етапом 1 і Етапом 3.

Додатково, протягом місяця було проведено тестування в умовах навантаження на контекстне вікно великих мовних моделей, де кількість повідомлень перевершує 500 (табл. 3.10).

Таблиця 3.10 – Результати навантажувального тестування

Критерій	Sliding Window Context	Summarized (Episodic) Memory	RAG	Гібридний метод
1	2	3	4	5
Згадування фактів з початку діалогу (створених 100 повідомлень тому)	-	-	+	+
Логічна зв'язність згаданих фактів	-	-	-	+
Оновлення статичного контексту відповідно до «згадок»	-	-	-	+

Порівняння методів шляхом тестування кожного з них дозволило дійти висновків щодо ефективності кожного з них. Аналіз результатів тестування продемонстрував, що жоден із протестованих ізольованих методів (Sliding Window, Summarized Memory, RAG) не забезпечив успіху у зв'язуванні критичних фактів із Етапу 1 та Етапу 3 тестового сценарію. Найкращий результат, що підтверджує успішне використання довготривалої пам'яті, досягнуто за допомогою запропонованого гібридного методу.

3.4 Перспективи подальшої роботи

Результати проведеного дослідження методів покращення контекстної пам'яті чат-боту на основі великих мовних моделей підтвердили, що саме гібридна архітектура є найбільш ефективною для забезпечення довготривалої контекстуальної зв'язності у складних діалогових системах. Оскільки додаткове впровадження ШІ-агенту сприяло значному поліпшенню логічних «згадок» мовною моделлю окремих «старих» фактів, можна дійти висновку, що стратегія інтеграції з агентами є перспективною.

Отже, подальше підвищення надійності та якості управління пам'яттю передбачає поглиблений пошук можливостей впровадження технологій ШІ-агентів у процес формування та вибірки контексту.

Перспективним є удосконалення гібридної методики шляхом розширення логіки розроблення спеціалізованих ШІ-агентів, а саме – створення такого агента, функціонал якого був би зосереджений на удосконаленні формування контексту для великої мовної моделі шляхом застосування механізмів оцінки важливості фактів.

Контекстно-залежне зважування дозволило би агентові імплементувати алгоритми, котрі надають підвищену значущість (вагу або теги) фактам, які є унікальними (наприклад, візуальні маркери), гарантуючи їхнє включення та пріоритетну позицію при формуванні промпту.

Окрім уніфікованого векторного подання для кожного повідомлення та статичної пам'яті, є перспективним надання агентові можливості генерувати множинні вектори, наприклад, для ключових сутностей, подій, нових конфліктів. Перш за все, може мінімізувати ризики семантичної втрати та істотно підвищувати точність вибірки релевантних фрагментів RAG-системою альтернативним шляхом (не занижуючи семантичну подібність).

Є перспектива масштабування агенту, котрий аналізує відносини персон та стани у діалоговій системі. Наприклад, з точки зору розширення функціоналу, додаючи автоматичне формування пріоритетів та логіку тегування тригерів

(створення системи ключових слів, які будуть тригерити певний клас реакцій від агенту) при відстежуванні змін у взаємодії між персонами.

Для посилення ефективності вирішення проблеми потенційного ризику ігнорування мовною моделлю витягнутої пам'яті (коли LLM має доступ до факту, але випадково може не інтегрувати його у свій висновок), пропонується розробити механізм активного згадування [44].

Механізм активного згадування можна розробляти шляхом створення проміжного модуля, котрий аналізує згенеровану відповідь LLM на предмет ігнорування критичних, витягнутих із пам'яті фактів. У разі ігнорування, модуль повинен інтерполювати вхідний контекст додатковим «нагадуванням» до LLM, стимулюючи необхідну логічну інтеграцію (наприклад, примусове порівняння витягнутих фактів).

Отже, зазначені напрямки масштабування ШІ-агентів є перспективними для розвитку активної когнітивної системи, котра суттєво підвищить глибину та достовірність діалогового досвіду користувача.

ВИСНОВКИ

Таким чином, у кваліфікаційній роботі досліджено методи покращення контекстної пам'яті чат-боту на основі великих мовних моделей та вирішено такі завдання:

- проведено аналіз сучасних методів покращення контекстної пам'яті чат-ботів на основі великих мовних моделей, що дало можливість детально вивчити їх недоліки та переваги для розроблення поліпшеної методики покращення пам'яті;

- вивчено та проведено аналіз літературних джерел щодо апробації методів, що забезпечують довготривалий контекст у діалогових системах, що дало можливість виявити сучасний стан дослідженої проблематики, недоліки та переваги існуючих методів;

- сформовано покрокові алгоритми для кожного із вибраних методів управління пам'яттю та візуалізовано їх у вигляді блок-схем, що дало можливість розробити програмну реалізацію кожного розглянутого методу;

- сформовано покроковий алгоритм для удосконаленого методу управління пам'яттю, котрий врахував переваги існуючих методів, що дало можливість розробити програмну реалізацію покращеної методики;

- розроблено програмні застосунки, на основі котрих проведено тестування ефективності методів покращення пам'яті, що дозволило провести необхідне дослідження;

- проведено тестування удосконаленої методики та порівняно її ефективність із обраними для дослідження методами за ключовими метриками, зокрема, за точністю та зв'язністю відповідей у довготривалих діалогах, що дозволило досягти мети кваліфікаційної роботи;

- на основі порівняльного аналізу та оцінки ефективності зроблено висновки щодо використання методів в залежності від вимог до діалогової системи, що дало можливість вирішити поставлені цілі дослідження.

У рамках кваліфікаційної роботи проведено дослідження методів Sliding Window Context, Summarized (Episodic) Memory та RAG, шляхом програмної реалізації застосунків, котрі надали можливість протестувати розглянуті методи на предмет ефективності покращення пам'яті чат-боту на основі великих мовних моделей.

Для кожного з методів побудовано та візуалізовано покрокові алгоритми за допомогою блок-схем.

Створено унікальний набір тестових даних, що включає 63 тестові повідомлення, поділені на три логічні етапи для перевірки кожного методу на контекстне «згадування» фактів.

Проведено тестування застосунків та проаналізовано результати ефективності контекстної довготривалої пам'яті для кожного із розглянутих методів. На основі результатів доведено перевагу гібридної методики в ефективності та якості пам'яті шляхом порівняння результатів тестування усіх досліджуваних методів.

Розроблено зручний UI для гібридної методики, за допомогою котрого користувач має змогу вести складні діалоги з чат-ботом із ефективною довготривалою системою пам'яті.

Наукова новизна роботи полягає у розробленні гібридної методики покращення контекстної пам'яті, враховуючи сильні та слабкі сторони існуючих методів.

Результати роботи апробовано у вигляді 2 тез доповідей під час XXIX Міжнародного молодіжного форуму «Радіоелектроніка і молодь у XXI столітті» [45] та II Міжнародної науково-практичної студентської конференції «ІТ-простір сьогодення: тенденції, інновації та перспективи розвитку» [46], а також у вигляді статті у зарубіжному науковому журналі «International Journal of Academic Information Systemums Research» [47].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Xiong, Z., Lin, Y., Xie, W., He, P., Tang, J., Lakkaraju, H., & Xiang, Z. (2025). How Memory Management Impacts LLM Agents: An Empirical Study of Experience-Following Behavior. arXiv. URL: <https://arxiv.org/abs/2505.16067> (дата звернення 01.09.2025).
2. Zeng, R., Fang, J., Liu, S., & Meng, Z. (2024). On the structural memory of llm agents. arXiv. URL: <https://arxiv.org/abs/2412.15266> (дата звернення 01.09.2025).
3. Liu, P., Yuan, G., Lin, X., Wang, Q., Hu, Z., Liu, X., ... & Cui, P. (2023). A Survey of Large Language Models. arXiv. URL: <https://arxiv.org/abs/2303.18223> (дата звернення 01.09.2025).
4. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv. URL: <https://arxiv.org/abs/2005.11401> (дата звернення 01.09.2025).
5. Karakonstantyn, D., & Tvoroshenko, I. (2024). About the issue of optimization the performance of the server part of the information system.
6. Wang, J., Wang, J., Chen, Z., Li, J., Wang, Z., Wen, Z., & Yang, M. (2023). A Survey of Large Language Models for Dialogue Systems. arXiv. URL: <https://arxiv.org/abs/2311.16789> (дата звернення 01.09.2025).
7. Zhang, Y., Yu, Z., Jiang, W., Shen, Y., & Li, J. (2023, November). Long-term memory for large language models through topic-based vector database. *In 2023 International Conference on Asian Language Processing (IALP)* (pp. 258-264). IEEE.
8. Gu, J., Liu, H., Li, S., Zhang, W., Shen, W., Zhang, K., ... & Zha, M. (2024). Long-context LLMs: A Comprehensive Survey. arXiv. URL: <https://arxiv.org/abs/2404.09886> (дата звернення 02.09.2025).
9. Wang, Q., Fu, Y., Cao, Y., Wang, S., Tian, Z., & Ding, L. (2023). Recursively Summarizing Enables Long-Term Dialogue Memory in Large Language Models. arXiv. URL: <https://arxiv.org/abs/2308.15022> (дата звернення 02.09.2025).

10. Gao, Y., Ma, Z., Lin, J., Zhang, C., Liu, Z., Wang, X., Han, R., Liu, H., Li, P., Li, J., Feng, S., & Li, C. (2024). Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv. URL: <https://arxiv.org/abs/2312.10997> (дата звернення 02.09.2025).

11. Humanloop. (2024, March 26). RAG architectures: An overview of retrieval-augmented generation architectures. URL: <https://humanloop.com/blog/rag-architectures> (дата звернення 02.09.2025).

12. Zhao, Y., Cao, H., Zhao, X., & Ou, Z. (2024, November). An empirical study of retrieval augmented generation with chain-of-thought. In *2024 IEEE 14th International Symposium on Chinese Spoken Language Processing (ISCSLP)* (pp. 436-440). IEEE.

13. Wang, Z., Teo, S. X. M., Ouyang, J., Xu, Y., & Shi, W. (2024). M-RAG: Reinforcing large language model performance through retrieval-augmented generation with multiple partitions. arXiv. URL: <https://arxiv.org/abs/2405.16420> (дата звернення 02.09.2025).

14. Zhang, Y., Wu, H., Xu, Z., Gu, C., Ma, S., & Li, M. (2024). Evaluating Very Long-Term Conversational Memory of LLM Agents. arXiv. URL: <https://arxiv.org/abs/2402.17753> (дата звернення 02.09.2025).

15. Karunathilake, K. A. G. G., Abhayawardhana, C. G., De Silva, M. I. W. R., & Gamage, S. N. (2023). The Right to be Forgotten in the Era of Large Language Models: Implications, Challenges, and Solutions. arXiv. URL: <https://arxiv.org/abs/2307.03941> (дата звернення 02.09.2025).

16. Tvoroshenko, I., Pomazan, V., Gorokhovatskyi, V., & Kobylin, O. (2023). Application of video data classification models using convolutional neural networks.

17. Tvoroshenko, I., Gorokhovatskyi, V., Kobylin, O., & Tvoroshenko, A. (2023). Application of deep learning methods for recognizing and classifying culinary dishes in images.

18. Pomazan, V., Tvoroshenko, I., & Gorokhovatskyi, V. (2023). Development of an application for recognizing emotions using convolutional neural networks.

19. Pomazan, V., Tvoroshenko, I., & Gorokhovatskyi, V. (2023). Handwritten character recognition models based on convolutional neural networks.
20. Gorokhovatskyi, V., Tvoroshenko, I., Yakovleva, O., Hudáková, M., & Gorokhovatskyi, O. (2024). Application a committee of Kohonen neural networks to training of image classifier based on description of descriptors set. *IEEE Access*, 12, 73376-73385.
21. Gorokhovatskyi, V., Tvoroshenko, I., Yakovleva, O., & Hudáková, M. (2025). Image description compression in classification structural methods. *IEEE Access*, 13, 43631-43641.
22. Gorokhovatskyi, V., & Tvoroshenko, I. (2024). Transforming image descriptions as a set of descriptors to construct classification features.
23. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., Gadetska, S., & Al-Dhaifallah, M. (2023). Statistical data analysis models for determining the relevance of structural image descriptions. *IEEE Access*, 11, 126938-126949.
24. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., & Zeghid, M. (2022). Tools for fast metric data search in structural methods for image classification. *IEEE Access*, 10, 124738-124746.
25. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., & Zeghid, M. (2024). Improving the Effectiveness of Image Classification Structural Methods by Compressing the Description According to the Information Content Criterion. *Computers, Materials & Continua*, 80(2).
26. Tvoroshenko, I., & Gorokhovatskyi, V. (2022). The Application of Hybrid Intelligence Systems for Dynamic Data Analysis.
27. Ayaz, A. M., Gorokhovatskyi, V., Tvoroshenko, I., Vlasenko, N., & Khalid, M. S. (2021). The research of image classification methods based on the introducing cluster representation parameters for the structural description.
28. Gorokhovatskyi, V., Tvoroshenko, I., & Chmutov, Y. (2022). Застосування систем ортогональних функцій для формування простору ознак у методах класифікації зображень. *Advanced Information Systems*, 6(3), 5-12.

29. Ibrahim, D. Y., Gorokhovatskyi, V., Tvoroshenko, I., & Mujahed, A. D. (2022). Classification of Images Based on a System of Hierarchical Features.
30. Tvoroshenko, I., & Kukharchuk, V. (2021). Current state of development of applications for recognition of faces in the image and frames of video captures.
31. Tvoroshenko, I., & Tkachenko, D. (2020). Mechanisms of image classification based on descriptors of local features.
32. Tvoroshenko, I., & Dziubenko, M. (2020). Modern methods of analysis of the movement scheme using video detection of vehicles.
33. Tvoroshenko, I., & Babochkin, O. (2021). Object identification method based on image keypoint descriptors.
34. Yakovleva O., Matúšová S., Tvoroshenko I., and Isaiev Y. (2024). Visitor counting based on video stream analysis from surveillance cameras to solve various business problems, *Verejná správa a regionálny rozvoj ekonómia, manažment a marketing*, XX(1), pp. 67-87.
35. Gorokhovatskyi V., Tvoroshenko I., Yakovleva O. (2024) Transforming image descriptions as a set of descriptors to construct classification features, *Indonesian Journal of Electrical Engineering and Computer Science*, 33 (1), 113-125.
36. OpenRouter. (2025, September). Sample code and API for Gemini 2.5 Flash Preview 09-2025. URL: <https://openrouter.ai/google/gemini-2.5-flash-preview-09-2025/api> (дата звернення 19.09.2025).
37. FastAPI. (2025, September). FastAPI framework, high performance, easy to learn, fast to code, ready for production. URL: <https://fastapi.tiangolo.com/> (дата звернення 19.09.2025).
38. aiogram 3.22.0 documentation. (2025, September). Simple usage. URL: <https://docs.aiogram.dev/en/dev-3.x/> (дата звернення 19.09.2025).
39. PyCharm. The only Python IDE you need. (2025, September). Features. URL: <https://www.jetbrains.com/pycharm/> (дата звернення 19.09.2025).
40. Connect a Python application to Neon Postgres. (2025, September). Create a table and insert data. URL: <https://neon.com/docs/guides/python#examples> (дата звернення 19.09.2025).

41. Gorokhovatskyi V., Chmutov Y., Tvoroshenko I., and Kobylin O. (2025) Reducing computational costs by compressing the structural description in image classification methods, *Advanced Information Systems*, vol. 9, no. 1, pp.5-12.

42. Gorokhovatskyi V., and Tvoroshenko I. (2024) An effective method for transforming an image description into a compact vector for classification. *Information Technology and Implementation (Satellite): Conference Proceedings*, November 21, 2024, Kyiv, Ukraine, Publishing House «Caravela», pp. 25-28.

43. Gorokhovatskyi V., Tvoroshenko I. (2023) Identification of visual objects by the search request. *Int. scientific symp. «Intelligent Solutions-S». Computational intelligence. Decision making theory: proceedings of the international symposium*, September 28, 2023, Kyiv-Uzhorod, Ukraine, 25-27.

44. Богдан, Н. І. (2024). Розробка вебзастосунку генерації творчих ідей.

45. Богдан Н. І. (2025). Аналіз особливостей Character.ai: проблема довготривалої пам'яті у генеративному штучному інтелекті. *Радіоелектроніка та молодь у XXI столітті: тези доповідей 29-го Міжнародного молодіжного форуму (Харків, 16–19 квітня 2025 р.)*. Харків: ХНУРЕ. Т. 7. С. 14-16.

46. Богдан Н. І. (2025). Порівняльний аналіз архітектурних рішень для інтеграції довготривалої пам'яті у діалогові системи. *II Міжнародній науково-практичній студентській конференції «IT-простір сьогодення: тенденції, інновації та перспективи розвитку»*

47. Bohdan N., Tvoroshenko I., Gorokhovatskyi V., and Kobylin O. (2025) Development of a hybrid method to enhance context memory for a chatbot application based on large language models, *International Journal of Academic Information Systems Research*, 9(10), pp. 7-18.