Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

До захисту допущено
Кафедрою комп'ютерних систем та робототехніки
протокол № __ від __ грудня 2025р.

завідувач кафедри _____ Максим ХРУСЛОВ
(підпис)

«___»_____ 2025 р.

Кваліфікаційна робота

здобувача другого (магістерського) рівня вищої освіти

«АДАПТИВНЕ УПРАВЛІННЯ КОНТЕКСТОМ У RAG-СИСТЕМАХ ДЛЯ ПЕРСОНАЛІЗОВАНИХ AI-АСИСТЕНТІВ»

Спеціальність ***123 – Комп'ютерна інженерія.***
Освітня програма ***Комп'ютерна інженерія***

Виконавець _____ **Андрій Супрун**
(підпис)

Науковий керівник _____ **Ніна Бакуменко**
(підпис)

Харків – 2025

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи магістра складається зі вступу, трьох розділів, висновків, списку використаних джерел і двох додатків. Загальний обсяг роботи складає 91 сторінки, з них основної частини 62 сторінок, з 7 рисунками, 8 таблицями, 4 лістингами, списком використаних джерел із 19 найменувань та п'ятьма додатками.

Метою кваліфікаційної роботи є розробка системи адаптивного управління контекстом, яка динамічно формує діалоговий контекст у RAG-архітектурі, забезпечуючи баланс між історією діалогу та зовнішніми знаннями з урахуванням персональних характеристик користувача.

Об'єкт дослідження – процес управління діалоговим контекстом у RAG-системах для персоналізованих AI-асистентів.

Предмет дослідження – методи та алгоритми адаптивного формування контексту з урахуванням історії діалогу, профілю користувача та специфіки предметної області. Розроблено архітектуру системи з чотирма компонентами та реалізовано алгоритм динамічного балансування. Програмна реалізація використовує FastAPI, PostgreSQL з pgvector, Redis та OpenAI API. Експериментальна оцінка на 100 діалогових сесіях показала покращення релевантності та coherence на 13%, економію токенів на 22%. Область застосування – розробка персоналізованих AI-асистентів, систем підтримки клієнтів та віртуальних консультантів на основі великих мовних моделей.

Ключові слова*: retrieval-augmented generation, управління контекстом, персоналізація, великі мовні моделі, діалогові системи, векторний пошук, компресія історії, AI-асистенти, RAG-архітектура, адаптивні алгоритми.*

ABSTRACT

The explanatory note to the master's qualification work consists of an introduction, three chapters, conclusions, a list of references, and two appendices. The total volume of the work is 91 pages, of which the main part is 62 pages, with 7 figures, 8 tables, 4 code listings, a list of references with 19 items, and five appendices.

The purpose of the qualification work is to develop an adaptive context management system that dynamically forms dialogue context in RAG architecture, ensuring a balance between dialogue history and external knowledge taking into account personal user characteristics.

The object of research is the process of dialogue context management in RAG systems for personalized AI assistants.

The subject of research is methods and algorithms for adaptive context formation taking into account dialogue history, user profile, and domain specifics. A system architecture with four components was developed and a dynamic balancing algorithm was implemented. The software implementation uses FastAPI, PostgreSQL with pgvector, Redis, and OpenAI API. Experimental evaluation on 100 dialogue sessions showed improvements in relevance and coherence by 13%, token savings of 22%. The field of application is the development of personalized AI assistants, customer support systems, and virtual consultants based on large language models.

Key words: *retrieval-augmented generation, context management, personalization, large language models, dialogue systems, vector search, history compression, AI assistants, RAG architecture, adaptive algorithms.*

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ACMS – Adaptive Context Management System, система адаптивного управління контекстом

AI – Artificial Intelligence, штучний інтелект

ANN – Approximate Nearest Neighbors, приблизний пошук найближчих сусідів

API – Application Programming Interface, програмний інтерфейс застосування

BM25 – Best Matching 25, алгоритм ранжування документів

CPU – Central Processing Unit, центральний процесор

DPR – Dense Passage Retrieval, щільний пошук фрагментів

FAISS – Facebook AI Similarity Search, бібліотека для пошуку подібностей

HNSW – Hierarchical Navigable Small World, ієрархічний навігований малий світ

IVF – Inverted File Index, інвертований файловий індекс

JSON – JavaScript Object Notation, текстовий формат обміну даними

KG – Knowledge Graph, граф знань

kNN-LM – k-Nearest Neighbors Language Model, мовна модель k найближчих сусідів

LLM – Large Language Model, велика мовна модель

NLP – Natural Language Processing, обробка природної мови

RAG – Retrieval-Augmented Generation, генерація з доповненням пошуком

REST – Representational State Transfer, архітектурний стиль для розподілених систем

RETRO – Retrieval-Enhanced Transformer, трансформер з підтримкою пошуку

TF-IDF – Term Frequency-Inverse Document Frequency, частота терміну-обернена частота документу

TTL – Time To Live, час життя

UI – User Interface, користувацький інтерфейс

α – коефіцієнт балансування між історією діалогу та зовнішніми знаннями

β – ваговий коефіцієнт для персональної релевантності

γ – ваговий коефіцієнт для доменної релевантності

λ – параметр експоненціального згладжування

ВСТУП

Сучасні системи штучного інтелекту на основі великих мовних моделей (Large Language Models, LLM) демонструють вражаючі можливості у розумінні та генерації природної мови, що відкриває нові горизонти для створення інтелектуальних діалогових систем. За останні роки спостерігається стрімке зростання використання AI-асистентів у різноманітних галузях: від клієнтської підтримки та фінансового консультування до медичної діагностики та освітніх платформ. Користувачі очікують від таких систем не лише точних відповідей, а й здатності «пам'ятати» контекст попередніх розмов, враховувати індивідуальні уподобання та адаптуватися до рівня експертизи співрозмовника.

Однак застосування LLM у реальних персоналізованих AI-асистентах стикається з фундаментальними обмеженнями. Незважаючи на значне розширення контекстних вікон у останніх моделях – від 4 тисяч токенів у GPT-3 до понад мільйона у Gemini 1.5 Pro – проблема ефективного використання доступного контексту залишається невирішеною [1]. Великі мовні моделі не володіють власною довготривалою пам'яттю про попередні взаємодії з користувачем, а просте збільшення розміру контекстного вікна не вирішує проблему релевантності та когерентності інформації у ньому. Більш того, емпіричні дослідження демонструють, що моделі схильні ігнорувати інформацію, розташовану в середині великого контексту, навіть якщо вона є критично важливою для формування відповіді.

Архітектура Retrieval-Augmented Generation (RAG), вперше формалізована Lewis et al. [2], виникла як перспективний підхід до подолання обмежень базових мовних моделей шляхом динамічного залучення зовнішніх знань під час генерації відповідей. RAG-системи поєднують параметричну пам'ять нейронної мережі з непараметричною пам'яттю у вигляді корпусу документів, що дозволяє суттєво підвищити фактичну точність відповідей без

необхідності перенавчання базової моделі. Цей підхід виявився особливо ефективним для задач, що вимагають актуальних знань або спеціалізованої доменної інформації, яка може бути відсутня у параметрах моделі. Подальший розвиток цих ідей представлено у роботі Borgeaud et al. з архітектурою RETRO [3], що демонструє можливість глибокої інтеграції механізму retrieval безпосередньо у шари трансформера.

Проте у довготривалих діалогових сесіях виникає критична проблема балансування контексту. По мірі розгортання діалогу історія спілкування накопичується, а кількість потенційно релевантних документів з бази знань зростає. Система має одночасно враховувати контекст попередніх повідомлень (для підтримки когерентності діалогу), релевантні документи з бази знань (для забезпечення фактичної точності), персональні характеристики користувача (для адаптації стилю та рівня деталізації) та специфіку предметної області (для правильної інтерпретації термінології). Ефективне управління цими компонентами у межах обмеженого контекстного вікна є складною оптимізаційною задачею.

Актуальність теми дослідження обумовлена стрімким зростанням використання AI-асистентів у різних галузях економіки та необхідністю забезпечення високої якості персоналізованих відповідей у довготривалих діалогах. За оцінками аналітиків, ринок розмовного штучного інтелекту демонструє експоненціальне зростання, а компанії різних секторів активно впроваджують AI-асистентів для автоматизації клієнтської підтримки, консультаційних послуг та внутрішніх бізнес-процесів. Водночас користувачі стають більш вимогливими до якості взаємодії: вони очікують, що асистент буде «пам'ятати» контекст попередніх розмов, враховувати їхні уподобання та надавати персоналізовані рекомендації.

По мірі розгортання діалогу історія спілкування накопичується, а кількість потенційно релевантних документів з бази знань зростає. Пряме

включення всієї історії діалогу та результатів векторного пошуку призводить до кількох критичних проблем. По-перше, навіть за наявності великого контекстного вікна, його неефективне використання призводить до значних обчислювальних витрат – вартість обробки запиту зростає пропорційно до кількості токенів у контексті. По-друге, перевантаження контексту нерелевантною або застарілою інформацією знижує якість відповідей через ефект «lost in the middle» [1] – дослідження показують, що модель схильна ігнорувати важливу інформацію, розташовану всередині великого контексту. По-третє, зростає час відгуку системи, що погіршує користувацький досвід та робить систему непридатною для інтерактивних застосувань.

Економічний аспект проблеми є особливо гострим для комерційних застосувань. Витрати на API-виклики до комерційних мовних моделей становлять значну частину операційних витрат компаній, що розробляють AI-асистенти. При типовому навантаженні у 10 000 запитів на день та середньому розмірі контексту 8 000 токенів, місячні витрати можуть сягати десятків тисяч доларів. Тому навіть 20–30% економія токенів має безпосередній економічний ефект та може визначати комерційну життєздатність продукту.

Наукова новизна роботи полягає у розробці інтегрованого підходу до адаптивного управління контекстом, що поєднує три рівні персоналізації: на рівні відбору історичних фрагментів діалогу, на рівні пріоритизації знань з бази даних та на рівні динамічного балансування між історією і знаннями. На відміну від існуючих підходів, таких як MemGPT [5], Recomp [4] чи методи sliding window, запропонована система враховує не лише семантичну близькість до поточного запиту, але й профіль користувача та доменну специфіку, що дозволяє формувати більш релевантний та збалансований контекст. Персоналізаційний шар використовує векторні подання для оцінки відповідності кожного фрагмента контексту як до поточного запиту, так і до профілю користувача, що дозволяє системі адаптивно визначати, які частини

історії варто зберегти у повному вигляді, які слід компресувати, а які можна відкинути.

Практична цінність роботи полягає у створенні готової до впровадження мікросервісної системи, яка може бути інтегрована у існуючі RAG-архітектури. Запропонований підхід не вимагає модифікації або перенавчання базової мовної моделі і працює на рівні формування контексту, що робить його універсальним та легко адаптованим до різних застосувань. Система реалізована з використанням сучасних технологій (FastAPI, PostgreSQL з pgvector, Redis, OpenAI API) та може масштабуватися для обробки великої кількості одночасних діалогових сесій. Результати апробації демонструють практичну ефективність підходу: скорочення витрат на токени на 22% при одночасному підвищенні релевантності відповідей на 14% та покращенні когерентності контексту на 13%.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Постановка задачі дослідження

Об'єктом дослідження є процес управління діалоговим контекстом у RAG-системах для персоналізованих AI-асистентів.

Предметом дослідження виступають методи та алгоритми адаптивного формування контексту з урахуванням історії діалогу, профілю користувача та специфіки предметної області.

Метою роботи є розробка та реалізація системи адаптивного управління контекстом (Adaptive Context Management System, ACMS), яка динамічно формує діалоговий контекст у RAG-архітектурі, забезпечуючи баланс між збереженням релевантної історії діалогу та залученням зовнішніх знань з урахуванням персональних характеристик користувача.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Проаналізувати існуючі підходи до управління контекстом у діалогових системах та RAG-архітектурах, виявити їх переваги та недоліки.
2. Розробити архітектуру системи адаптивного управління контекстом, що поєднує механізми ковзного вікна, LLM-компресії історії та персоналізаційний шар.
3. Реалізувати алгоритми динамічного відбору та ранжування фрагментів контексту на основі векторних подань профілю користувача та доменної специфіки.
4. Створити мікросервісну систему з використанням FastAPI, PostgreSQL з розширенням pgvector, Redis та OpenAI API.
5. Провести апробацію системи на синтетичних діалогах у різних предметних областях з використанням протоколу LLM-as-a-Judge [13].

6. Оцінити ефективність запропонованого підходу за метриками relevance@5, context coherence, Token Economy та Answer Win-Rate.

1.2 Проблематика управління контекстом у LLM-системах

Великі мовні моделі демонструють значний прогрес у розумінні та генерації природної мови, однак їх застосування у реальних діалогових системах стикається з фундаментальними обмеженнями, пов'язаними з управлінням контекстом. Незважаючи на розширення контекстних вікон у сучасних моделях, проблема ефективного використання доступного контексту залишається критичною для практичних застосувань.

1.2.1 Обмеження контекстного вікна

Контекстне вікно визначає максимальну кількість токенів, яку модель може обробити за один запит, включаючи як вхідний промпт, так і згенеровану відповідь. Обчислювальна складність трансформерних архітектур [9] зростає квадратично відносно довжини послідовності, що призводить до суттєвого збільшення часу обробки запитів при зростанні контексту. Вартість API-викликів до комерційних моделей прямо пропорційна кількості оброблених токенів, тому неефективне використання контекстного вікна має безпосередній економічний вплив на операційні витрати AI-систем. У довготривалих діалогах історія спілкування природно накопичується, і без належних механізмів управління контекстом система швидко вичерпує доступний ліміт.

Емпіричні дослідження показують, що навіть коли технічно можливо включити всю історію діалогу у контекстне вікно, це не гарантує високої якості відповідей [1]. Більш того, збільшення розміру контексту понад певну межу може призводити до погіршення якості відповідей через зниження щільності релевантної інформації та зростання шуму у вхідних даних.

1.2.2 Ефект «lost in the middle»

Критичною проблемою, що виникає при роботі з великими контекстами, є феномен «lost in the middle», детально досліджений Liu et al. у 2023 році [1]. Автори провели серію експериментів з різними мовними моделями, включаючи GPT-3.5-Turbo та Claude, та виявили систематичний паттерн: модель демонструє високу точність при використанні інформації з початку та кінця контексту, але значно гірше справляється з інформацією, розташованою у середині довгого промпту. Цей феномен є наслідком особливостей механізму уваги (attention) у трансформерних архітектурах, де позиційне кодування та патерни уваги створюють неявне зміщення до крайніх позицій послідовності.

У експериментах тестували здатність моделей відповідати на запитання, коли релевантна інформація була розміщена у різних позиціях всередині контексту довжиною від 4 до 32 тисяч токенів. Результати показали, що точність відповідей може падати на 20–30% для інформації з середини контексту порівняно з початком або кінцем. Цей ефект проявляється незалежно від загального розміру контекстного вікна моделі – навіть моделі з підтримкою 100+ тисяч токенів демонструють подібну деградацію якості. Важливо зазначити, що ефект посилюється при наявності семантично подібної, але фактично нерелевантної інформації у контексті, коли модель схильна до «блукання» між різними фрагментами тексту.

Феномен «lost in the middle» має особливе значення для RAG-систем, де релевантні документи з бази знань динамічно додаються до контексту. Якщо ці документи розміщені у середині великого контексту між системним промптом та історією діалогу, модель може фактично ігнорувати критично важливу інформацію, навіть якщо вона присутня у контексті. Це призводить до парадоксальної ситуації: збільшення кількості наданої інформації може погіршувати якість відповідей через розмивання фокусу уваги моделі.

Механізми уваги (attention) у трансформерних архітектурах [9], незважаючи на їх теоретичну здатність фокусуватися на будь-якій частині

вхідної послідовності, на практиці демонструють систематичне зміщення до початкових та кінцевих токенів. Це створює необхідність у розробці спеціалізованих стратегій розміщення інформації у контексті, що враховують ці особливості поведінки моделей.

1.2.3 Економічні аспекти

Управління контекстом має безпосередній вплив на економічну ефективність AI-систем. Ціноутворення більшості комерційних API мовних моделей базується на кількості оброблених токенів, причому розрізняють вхідні (input) та вихідні (output) токени. Для високонавантажених додатків з тисячами користувачів та мільйонами запитів на день витрати на API можуть становити значну частину операційного бюджету, що робить оптимізацію розміру контексту критично важливою бізнес-задачею.

Неефективне управління контекстом призводить до включення великої кількості нерелевантної або надлишкової інформації у кожен запит. Якщо типовий діалог містить 50 повідомлень, і система наївно включає всю історію у кожен запит, то при середній довжині повідомлення 100 токенів контекст зростає до 5000 токенів лише для історії діалогу. Додавання системних інструкцій, профілю користувача та результатів RAG-пошуку може збільшити загальний розмір контексту до 10–15 тисяч токенів на запит. За 10000 запитів на день це призводить до обробки 100–150 мільйонів токенів, що створює значні операційні витрати.

Окрім прямих фінансових витрат, розмір контексту впливає на швидкість відгуку системи. Час обробки запиту зростає приблизно квадратично з довжиною контексту через механізми self-attention у трансформерних архітектурах. Для користувачів це означає збільшення латентності – часу між надсиланням запиту та отриманням відповіді. Дослідження користувацького досвіду показують, що затримки понад 2–3 секунди значно погіршують сприйняття якості інтерактивних систем. У

критичних застосуваннях, таких як підтримка клієнтів реального часу або медичне консультування, висока латентність може робити систему непридатною для використання.

Додатковим фактором є обмеження швидкості (rate limits), що накладаються провайдерами API. Більшість сервісів обмежують не лише кількість запитів за хвилину, але й загальну кількість токенів, які можна обробити за певний період. Системи з неефективним управлінням контекстом швидше досягають цих лімітів, що вимагає або зниження якості обслуговування, або додаткових інвестицій у придбання вищих рівнів доступу до API.

1.3 Архітектури Retrieval-Augmented Generation

1.3.1 Базова концепція RAG

Фундаментальна ідея Retrieval-Augmented Generation була формалізована Lewis et al. у роботі 2020 року [2]. Автори запропонували архітектуру, що поєднує нейронний retriever для пошуку релевантних документів з генеративною мовною моделлю для формування відповіді. Ключовою інновацією підходу є те, що обидва компоненти можуть навчатися end-to-end, оптимізуючи якість кінцевої відповіді. RAG-системи поєднують параметричну пам'ять нейронної мережі (знання, закодовані у вагах моделі під час навчання) з непараметричною пам'яттю (зовнішня база документів, яка може оновлюватися без перенавчання моделі).

Базова архітектура RAG складається з трьох основних компонентів. Encoder-модель (зазвичай BERT-подібна архітектура) перетворює документи з корпусу знань у щільні векторні подання (dense embeddings). Ці вектори індексуються у спеціалізованій структурі даних для ефективного пошуку найближчих сусідів. Для вхідного запиту користувача система використовує ту ж encoder-модель для отримання векторного подання запиту, після чого виконує пошук k найбільш семантично близьких документів за метрикою

косинусної подібності або евклідової відстані. Знайдені документи конкатенуються з оригінальним запитом та подаються як вхід до генеративної моделі, яка формує фінальну відповідь.

У оригінальній роботі Lewis et al. продемонстрували два варіанти архітектури: RAG-Sequence, де один набір документів використовується для генерації всієї відповіді, та RAG-Token, де для кожного токену відповіді може використовуватися різний набір документів. Емпіричні результати на задачах question answering показали, що RAG-Sequence перевершує чисто параметричні моделі аналогічного розміру, особливо на запитаннях, що вимагають фактичних знань, які не були частиною навчального набору моделі.

1.3.2 RETRO та глибока інтеграція retrieval

Подальший розвиток ідей RAG представлено у роботі Borgeaud et al., де запропоновано архітектуру RETRO (Retrieval-Enhanced TRansfOrmer) [3]. Ключова відмінність RETRO полягає у глибокій інтеграції механізму retrieval безпосередньо у шари трансформера, а не простому конкатенуванні знайдених документів з промптом.

RETRO використовує chunked cross-attention – механізм, що дозволяє моделі на кожному шарі звертатися до релевантних фрагментів з отриманих документів. Емпіричні результати показали, що RETRO з 7 мільярдами параметрів може досягати якості, порівнянної з моделями у 25+ мільярдів параметрів без retrieval механізму. Це демонструє потенціал підходів на основі retrieval для створення більш компактних та економічно ефективних моделей.

1.3.3 Векторні бази даних та семантичний пошук

Ефективність RAG-систем критично залежить від якості компонента retrieval. Традиційні методи пошуку на основі ключових слів (BM25, TF-IDF) виявилися недостатніми для семантичного розуміння запитів, що призвело до розвитку dense retrieval підходів. Основна перевага dense retrieval полягає у

здатності виявляти семантичну близькість між текстами, що не мають спільних ключових слів, але передають подібний зміст.

Dense retrieval базується на перетворенні як документів, так і запитів у векторні подання у спільному семантичному просторі, де близькість векторів відображає семантичну близькість текстів [15]. Для генерації embeddings використовуються спеціально навчені encoder-моделі, такі як DPR (Dense Passage Retrieval) [14, 15] або Sentence-BERT. Ці моделі навчаються на великих датасетах пар запит-документ максимізувати косинусну подібність між релевантними парами та мінімізувати для нерелевантних.

Технічною проблемою є ефективний пошук найближчих сусідів у просторі високої розмірності (зазвичай 768–1536 вимірів) серед мільйонів або мільярдів векторів. Точний пошук має складність $O(n \cdot d)$, де n – кількість документів, d – розмірність векторів, що робить його непридатним для великих баз даних. Це призвело до розвитку спеціалізованих векторних баз даних та алгоритмів приблизного пошуку найближчих сусідів (ANN – Approximate Nearest Neighbors).

Серед популярних рішень можна виділити FAISS (Facebook AI Similarity Search) [11], що реалізує різні алгоритми ANN, включаючи IVF (Inverted File Index) та HNSW (Hierarchical Navigable Small World) [12]. Pinecone та Weaviate надають managed сервіси векторних баз даних з підтримкою розподілених індексів та горизонтального масштабування. Для помірних обсягів даних (до мільйонів документів) широко використовується pgvector [17] – розширення PostgreSQL для зберігання та пошуку векторів, що дозволяє інтегрувати векторний пошук у існуючі реляційні бази даних.

Сучасні підходи до retrieval також включають гібридні методи, що поєднують dense та sparse пошук. Наприклад, можна використовувати BM25 для первинної фільтрації кандидатів, після чого застосовувати dense retrieval для фінального ранжування. Це дозволяє використовувати переваги обох

підходів: стійкість до точних збігів термінів від BM25 та семантичне розуміння від dense retrieval.

1.3.4 Проблеми класичного RAG у довгих діалогах

Незважаючи на успіхи базових RAG-архітектур у задачах single-turn question answering, їх застосування у multi-turn діалогових системах виявляє низку фундаментальних проблем. Головна складність полягає у необхідності балансувати між збереженням контексту діалогу та інтеграцією зовнішніх знань у обмеженому контекстному вікні.

У класичному RAG запит користувача використовується для пошуку релевантних документів. Однак у діалозі поточний запит часто є контекстно-залежним та не може бути правильно інтерпретований без урахування попередніх повідомлень. Наприклад, запит «А що щодо побічних ефектів?» має сенс лише у контексті попереднього обговорення конкретного медичного препарату. Пошук документів виключно за цим запитом призведе до отримання нерелевантних результатів.

Існує кілька підходів до вирішення цієї проблеми. Перший – використовувати всю історію діалогу як запит для retrieval, що призводить до зростання розміру запиту та потенційного «розмивання» семантичного вектора через змішування різних тем з історії. Другий – застосовувати query rewriting, де LLM переформулює поточний запит з урахуванням контексту у самодостатню форму. Це вимагає додаткового виклику моделі та збільшує латентність системи. Третій – використовувати спеціалізовані encoder-моделі, навчені враховувати діалоговий контекст, але такі моделі вимагають значних датасетів анотованих діалогів для навчання.

Додатковою проблемою є накопичення документів з різних кроків діалогу. Якщо на кожному кроці система отримує топ-5 документів з бази знань, то за 10 кроків діалогу накопичується 50 потенційно релевантних документів. Навіть якщо кожен документ становить лише 200 токенів,

загальний обсяг отриманої інформації досягає 10000 токенів – значної частини доступного контекстного вікна. При цьому документи з ранніх кроків діалогу можуть вже не бути релевантними до поточної теми розмови.

Проблема посилюється ефектом «lost in the middle»: якщо релевантна інформація з недавнього RAG-пошуку розміщена всередині великого контексту між історією діалогу та поточним запитом, модель може фактично її проігнорувати. Це призводить до ситуації, коли система технічно має доступ до правильної інформації, але не використовує її при генерації відповіді.

1.4 Методи управління діалоговим контекстом

Для вирішення проблеми зростання контексту у довгих діалогах дослідники запропонували різноманітні підходи, що можна класифікувати на кілька категорій: sliding window методи, compression-based підходи та системи з явною довготривалою пам'яттю.

1.4.1 Sliding Window підходи

Найпростішим методом управління контекстом є підхід ковзного вікна, де зберігається лише фіксована кількість останніх повідомлень діалогу. Наприклад, система може зберігати останні N=10 повідомлень, відкидаючи старіші. Це гарантує, що розмір контексту залишається обмеженим та передбачуваним незалежно від тривалості діалогу.

Переваги sliding window очевидні: простота реалізації, передбачуваність витрат на API, відсутність додаткових обчислювальних витрат. Метод особливо ефективний для діалогів, де релевантна інформація концентрується у недавніх повідомленнях, а старіші частини розмови можна безпечно відкинути без втрати якості відповідей.

Однак підхід має суттєві обмеження. По-перше, він призводить до повної втрати інформації зі старих частин діалогу, навіть якщо вона є критично важливою. Наприклад, якщо користувач на початку діалогу повідомив

важливу контекстну інформацію (бюджетні обмеження, специфічні вимоги, персональні уподобання), ця інформація буде втрачена після N кроків. По-друге, фіксований розмір вікна не адаптується до характеру діалогу – деякі розмови можуть вимагати більшого контексту, інші меншого.

Варіації sliding window включають адаптивне вікно, де розмір N динамічно змінюється залежно від доступного бюджету токенів після додавання системних інструкцій та результатів RAG. Можна також використовувати hierarchical windowing, де зберігаються останні N повідомлень повністю, а попередні M повідомлень у стисненій формі. Такий підхід дозволяє частково зберегти інформацію зі старих частин діалогу при контрольованому розмірі контексту.

1.4.2 Compression-based методи

Альтернативним підходом є стиснення старих частин діалогу у компактні резюме, що зберігають ключову інформацію при зменшенні кількості токенів. Цей підхід базується на гіпотезі, що більшість діалогового контексту є надлишковим, і можна виділити семантично важливу інформацію у стиснутій формі.

Recomp [4] запропонував extractive та abstractive компресію для RAG-систем. Extractive компресія відбирає найбільш релевантні речення з отриманих документів, відкидаючи менш важливі частини. Abstractive компресія використовує LLM для генерації стислого резюме, що передає суть документів меншою кількістю токенів. Експерименти показали, що можна досягти скорочення на 40–60% при збереженні якості відповідей на рівні, близькому до використання повних документів.

LLMLingua розвиває ідею компресії промптів через видалення менш важливих токенів на основі оцінки їх внеску у семантичне значення. Система використовує невелику мовну модель (наприклад, GPT-2) для обчислення perplexity кожного токену у контексті, після чого видаляє токени з низькою

information density. Підхід дозволяє досягти високих коефіцієнтів компресії (до 80% скорочення) при помірній деградації якості відповідей.

Загальна ідея context compression полягає у використанні LLM для створення стислих резюме старих частин діалогу. Наприклад, кожні 10 повідомлень можуть бути замінені на параграф-резюме, що містить ключові факти та рішення з цієї частини діалогу. Це вимагає додаткових викликів моделі для генерації резюме, але дозволяє зберігати важливу інформацію з усього діалогу у обмеженому контексті.

Критичним питанням для compression-based методів є те, яку інформацію зберігати у резюме. Агресивна компресія може видалити деталі, що виявляться важливими пізніше у діалозі. З іншого боку, занадто консервативна компресія дає лише незначне скорочення розміру контексту. Автоматичне визначення оптимального рівня деталізації залишається відкритою дослідницькою проблемою.

1.4.3 Системи з явною пам'яттю

Третій клас підходів базується на створенні явних систем пам'яті, що організовують діалоговий контекст у структуровані сховища різних типів. Ці системи намагаються імітувати людську пам'ять, розрізняючи короткочасну та довготривалу пам'ять з відповідними механізмами збереження та доступу до інформації.

MemGPT [5] запропонував архітектуру, інспіровану ієрархією пам'яті в операційних системах. Система розрізняє робочу пам'ять (working memory) – обмежений контекст, безпосередньо доступний моделі, та довготривалу пам'ять (long-term memory) – необмежене зовнішнє сховище, організоване як векторна база даних. LLM навчається генерувати спеціальні команди для управління пам'яттю: переміщення інформації між робочою та довготривалою пам'яттю, пошук релевантних спогадів, оновлення існуючих записів.

Ключова інновація MemGPT полягає у тому, що модель сама приймає рішення про те, що зберігати у довготривалій пам'яті та коли звертатися до неї. Це досягається через fine-tuning моделі на синтетичних діалогах з анотаціями оптимальних стратегій управління пам'яттю. Експерименти показали, що MemGPT може підтримувати когерентні діалоги протяжністю тисячі кроків, ефективно використовуючи інформацію з різних частин історії.

LangChain Memory [16] надає набір абстракцій для управління пам'яттю у діалогових застосуваннях. Бібліотека включає реалізації різних стратегій: ConversationBufferMemory (зберігання всієї історії), ConversationBufferWindowMemory (sliding window), ConversationSummaryMemory (compression-based), ConversationKGMemory (knowledge graph-based). Розробники можуть комбінувати різні типи пам'яті для створення гібридних систем, наприклад, зберігаючи останні повідомлення у повному вигляді та резюме старіших частин.

Підхід на основі графів знань (knowledge graph) дозволяє витягувати структуровану інформацію з діалогу та зберігати її у формі триплетів (entity-relation-entity). Це дозволяє системі відповідати на запитання типу «Що користувач сказав про X?» навіть якщо оригінальне повідомлення було давно та не включене у поточний контекст.

Khandelwal et al. [6] запропонували підхід kNN-LM, що розширює мовні моделі через доступ до великого зовнішнього сховища прикладів. Для кожного токену, що генерується, система шукає найбільш подібні контексти у базі даних попередніх прикладів та використовує розподіл токенів, що слідували за цими контекстами, для коригування ймовірностей базової моделі. Переваги kNN-LM включають можливість використовувати необмежені обсяги історичних даних та автоматичну адаптацію до доменно-специфічних паттернів без перенавчання моделі. Недоліками є додаткова обчислювальна складність пошуку найближчих сусідів при кожній генерації токену.

1.5 Персоналізація у діалогових системах

Ефективна персоналізація є критичною для створення AI-асистентів, здатних адаптуватися до індивідуальних потреб, уподобань та контексту конкретного користувача. Персоналізація у діалогових системах охоплює широкий спектр аспектів – від адаптації стилю спілкування до врахування специфічних знань про користувача при формуванні відповідей.

1.5.1 Персоналізований пошук

Zhang et al. [7] досліджували проблему персоналізованого dense retrieval у контексті довгих діалогових історій. Автори відзначають, що стандартні retrieval моделі, навчені на загальних датасетах, не враховують специфічні характеристики користувача та контекст його попередніх взаємодій з системою. Це призводить до отримання документів, що є релевантними до запиту в загальному сенсі, але можуть не відповідати конкретним потребам та рівню знань користувача.

Запропонований підхід включає створення персоналізованих векторних подань запитів, що враховують історію взаємодій користувача. Для цього навчається спеціальна encoder-модель, яка приймає на вхід не лише поточний запит, але й резюме попередніх діалогів користувача, його явні уподобання та профіль. Модель навчається максимізувати подібність між персоналізованим запитом та документами, що користувач вважав корисними у минулому.

Експерименти на датасетах діалогів продемонстрували суттєве покращення якості retrieval при використанні персоналізації. Зокрема, метрика Recall@5 покращилась на 12–18% порівняно з базовим підходом без персоналізації. Більш важливо, якісний аналіз показав, що персоналізована система краще адаптується до рівня експертизи користувача – для початківців надаються більш базові пояснення, для експертів – технічно деталізована інформація.

1.5.2 Підтримка множинних сесій

Mazaré et al. [8] розглядали проблему навчання персоналізованих діалогових агентів, здатних підтримувати когерентність через множинні сесії спілкування з одним користувачем. У реальних застосуваннях користувачі взаємодіють з AI-асистентом не у рамках одного безперервного діалогу, а через серію окремих сесій, розділених у часі.

Критична проблема полягає у збереженні та активації релевантної інформації з попередніх сесій. Користувач може очікувати, що асистент «пам'ятає» важливі факти з минулих розмов – наприклад, його уподобання, попередньо обговорені плани, або контекст поточного проекту. Наївне включення всієї історії всіх сесій у контекст швидко стає непрактичним через обмеження розміру контексту.

Автори запропонували архітектуру з явною персональною пам'яттю, організованою як набір key-value записів. Ключі відповідають типам інформації (уподобання, факти про користувача, поточні цілі), значення містять відповідні дані. Спеціальний механізм attention дозволяє моделі вибірково звертатися до релевантних частин персональної пам'яті при обробці поточного запиту.

Навчання системи здійснюється на датасетах, що імітують множинні сесії з одним користувачем. Модель отримує сигнал нагороди за використання інформації з попередніх сесій для надання більш персоналізованих та контекстно-відповідних відповідей. Експерименти показали, що система значно краще справляється з підтримкою довготривалих взаємодій порівняно з baseline моделями без явної міжсесійної пам'яті.

1.5.3 User profiling techniques

Створення та підтримка профілю користувача є фундаментальною задачею для персоналізації. Профіль може включати різні типи інформації: демографічні дані (вік, локація, професія), явні уподобання (вибрані

користувачем налаштування), неявні уподобання (виведені з поведінки), історію взаємодій, рівень експертизи у різних доменах.

Існують різні підходи до представлення профілю. Структурований підхід використовує фіксовану схему з набором полів та їх значень. Це дозволяє легко інтегрувати профіль у промпт у текстовій формі або використовувати його для фільтрації та ранжування контенту. Недоліком є негнучкість – складно додавати нові типи інформації без модифікації схеми.

Векторний підхід представляє профіль користувача як embedding у тому ж просторі, що й документи та запити. Профіль може формуватися як середнє векторних подань усіх повідомлень користувача або документів, з якими він позитивно взаємодіяв. Це дозволяє використовувати семантичну подібність між профілем та потенційним контентом для персоналізації. Проте векторне подання є менш інтерпретованим та важче контролюється користувачем.

Гібридні підходи поєднують структуровані та векторні представлення. Наприклад, можна зберігати явні факти про користувача у структурованій формі, а його загальні інтереси та преференції – як vector embedding. Актуальною проблемою є privacy – профілі містять чутливу персональну інформацію, що вимагає належних заходів безпеки та прозорості щодо того, як ці дані збираються та використовуються.

1.6 Порівняльний аналіз та обґрунтування підходу

Аналіз розглянутих підходів виявляє, що кожен з них має специфічні переваги та обмеження, що робить їх ефективними у певних сценаріях, але недостатніми для комплексного вирішення проблеми адаптивного управління контекстом.

Sliding window підходи забезпечують простоту реалізації та передбачуваність, але жертвують потенційно важливою інформацією зі старих частин діалогу. Вони можуть бути ефективними для коротких транзакційних діалогів, де контекст швидко змінюється та старі повідомлення втрачають

релевантність, однак непридатні для складних консультаційних сценаріїв, де користувач може повертатися до тем, обговорених на початку розмови. Наприклад, якщо користувач на початку діалогу повідомив про бюджетні обмеження або специфічні вимоги, ця інформація буде безповоротно втрачена після N кроків.

Compression-based методи (Recomp [4], LLMLingua) представляють більш інтелектуальний підхід, зберігаючи суть старого контексту у стисненій формі. Вони досягають значного скорочення токенів при помірній втраті якості. Проте ці методи стикаються з фундаментальною проблемою: рішення про те, яку інформацію зберегти у резюме, приймається без знання майбутніх запитів користувача. Інформація, що здавалася нерелевантною при компресії, може виявитися критичною пізніше у діалозі. Додатково, кожен акт компресії вимагає виклику LLM, що збільшує латентність та витрати.

MemGPT [5] та подібні системи з явною організацією пам'яті надають найбільшу гнучкість, дозволяючи моделі самій вирішувати, що зберігати у довготривалій пам'яті та коли звертатися до неї. Це наближається до того, як люди управляють власною пам'яттю під час розмов. Однак навчання моделі ефективному використанню цих механізмів вимагає великих датасетів анотованих діалогів та значних обчислювальних ресурсів для fine-tuning. Більше того, складність системи робить її поведінку менш передбачуваною та важче контрольованою, що може бути критичним для комерційних застосувань.

Персоналізовані підходи до retrieval [7] демонструють суттєві покращення якості для користувачів з багатою історією взаємодій, але вимагають накопичення достатньої кількості даних про користувача перед тим, як персоналізація стає ефективною. Це створює проблему «холодного старту» для нових користувачів. Крім того, більшість досліджень персоналізації фокусуються виключно на компоненті retrieval, не розглядаючи

його інтеграцію з управлінням діалоговим контекстом, що обмежує практичну застосовність цих підходів.

Обґрунтування необхідності інтегрованого підходу

Аналіз існуючих методів показує, що жоден з них окремо не вирішує всього спектру проблем, що виникають при створенні персоналізованих RAG-асистентів для довготривалих діалогів. Ефективна система повинна одночасно вирішувати кілька взаємопов'язаних задач: ефективно управляти зростаючим діалоговим контекстом, балансуючи між збереженням історії та економією токенів; інтегрувати динамічний пошук зовнішніх знань, не перевантажуючи контекст надлишковими документами; адаптуватися до персональних характеристик користувача, його уподобань та рівня експертизи; враховувати специфіку предметної області, пріоритизуючи доменно-релевантну інформацію; зберігати прозорість та контрольованість поведінки для практичного застосування.

Це обґрунтовує необхідність інтегрованого підходу, що поєднує елементи різних методів у когерентну систему. Запропонована у даній роботі Adaptive Context Management System (ACMS) синтезує ідеї з sliding window (для збереження актуальних повідомлень), compression (для стиснення старої історії за допомогою LLM), персоналізованого retrieval (для адаптації до користувача) та доменної специфікації у єдину архітектуру. На відміну від існуючих рішень, ACMS працює як проміжний шар (middleware) між користувацьким інтерфейсом та мовною моделлю, не вимагаючи fine-tuning базової моделі, що забезпечує універсальність та легкість впровадження.

Ключовою інновацією є персоналізаційний шар, що діє на всіх етапах формування контексту: при відборі фрагментів історії для компресії чи видалення, при ранжуванні документів з бази знань, та при визначенні балансу між історією діалогу та зовнішніми знаннями. Персоналізаційний шар використовує векторні подання профілю користувача для оцінки релевантності кожного фрагмента контексту не лише до поточного запиту, а й

до довгострокових інтересів та характеристик користувача. Це дозволяє системі приймати інформовані рішення про управління контекстом з урахуванням специфіки конкретного користувача та задачі, замість застосування універсальних евристик.

Практична орієнтація підходу проявляється у виборі простих, але ефективних компонентів: використання GPT-4o-mini для компресії замість складного fine-tuning спеціалізованих моделей, векторні бази даних на базі pgvector замість складних розподілених систем, явні коефіцієнти балансування замість непрозорих механізмів attention. Це робить систему доступною для впровадження у реальні продукти при збереженні значних переваг у якості порівняно з базовими підходами.

Висновки до розділу 1

У першому розділі проведено аналіз предметної області та сформульовано задачі дослідження.

Виявлено ключові проблеми управління контекстом у LLM-системах: квадратична обчислювальна складність, феномен «lost in the middle» та високі операційні витрати при обробці великих контекстів.

Проаналізовано архітектури RAG та методи управління діалоговим контекстом. Встановлено, що sliding window підходи втрачають важливу інформацію, compression-based методи не можуть передбачити майбутню релевантність, а системи з явною пам'яттю вимагають складного навчання.

Показано, що існуючі підходи не забезпечують комплексного вирішення проблем персоналізованих діалогових систем, оскільки розглядають окремі аспекти ізольовано.

Обґрунтовано необхідність розробки інтегрованої системи адаптивного управління контекстом, що поєднує механізми ковзного вікна, LLM-компресії та персоналізаційний шар для ефективної роботи у довготривалих діалогах з урахуванням характеристик користувача та специфіки предметної області.

РОЗДІЛ 2
РОЗРОБКА СИСТЕМИ АДАПТИВНОГО УПРАВЛІННЯ КОНТЕКСТОМ

2.1 Постановка задачі та декомпозиція

2.1.1 Формалізація проблеми управління контекстом

Задача адаптивного управління контекстом у RAG-системах може бути формалізована наступним чином. Нехай маємо діалогову сесію S, що складається з послідовності пар повідомлень користувача та відповідей асистента: $S = \{(u_1, a_1), (u_2, a_2), ..., (u_n, a_n)\}$, де u_i – і-те повідомлення користувача, a_i – відповідна відповідь системи.

Для генерації відповіді a_{n+1} на новий запит u_{n+1} система має сформувати контекст C, який подається разом із запитом до мовної моделі. У класичному підході цей контекст складається з усієї історії діалогу та результатів пошуку у базі знань. Однак таке рішення призводить до експоненціального зростання розміру контексту, що створює кілька фундаментальних проблем.

По-перше, існує жорстке обмеження на максимальний розмір контексту L_max, визначений архітектурою мовної моделі. Навіть для сучасних моделей з розширеним контекстним вікном (наприклад, 128K токенів для GPT-4), це обмеження стає критичним у довготривалих діалогах або при роботі з великими документами. По-друге, вартість обробки запиту C(|C|) зростає лінійно або квадратично (залежно від реалізації attention механізму) відносно розміру контексту |C|, що робить обробку великих контекстів економічно нераціональною. По-третє, емпіричні дослідження демонструють ефект "lost in the middle", коли модель приділяє непропорційно менше уваги інформації, розташованій всередині довгого контексту, порівняно з початком та кінцем.

Таким чином, основна задача полягає у визначенні функції відбору контексту f_select, яка для поточного запиту u_{n+1}, історії діалогу $S_{1:n} = \{(u_1, a_1),$

..., (u_n, a_n)}, бази знань K та профілю користувача P формує оптимальний контекст:

$$C^* = f\_select(u_{n+1}, S_{1:n}, K, P) \quad (2.1)$$

при наступних обмеженнях де:

$|C^*| \leq L\_max$ (обмеження розміру)

$C(|C^*|) \leq B$ (бюджетне обмеження)

$Q(C^*, u_{n+1}) \rightarrow max$ (максимізація якості відповіді)

$R(C^*, P) \rightarrow max$ (максимізація релевантності до профілю)

де Q – функція оцінки якості контексту для генерації відповіді, R – функція оцінки персональної релевантності.

Ключова складність полягає у тому, що ці критерії часто конфліктують між собою: збільшення обсягу контексту може підвищити якість відповіді (більше інформації), але одночасно зменшити релевантність (розмивання уваги моделі) та збільшити витрати. Крім того, оптимальний баланс між цими критеріями залежить від специфіки запиту, предметної області та характеристик конкретного користувача.

2.1.2 Декомпозиція задачі на підзадачі

Для вирішення сформульованої задачі проведемо її декомпозицію на чотири взаємопов'язані підзадачі, кожна з яких має власні вхідні дані, обмеження та критерії оптимізації.

Підзадача 1: Управління діалоговою історією (Dialog History Management). Необхідно визначити, які повідомлення з історії діалогу $S_{1:n}$ включити до контексту, в якій формі та як організувати цю інформацію. Формально, потрібно визначити функцію:

$$H = f\_history(S_{1:n}, u_{n+1}, P) = \{h\_recent, h\_compressed\} \quad (2.2)$$

де h_recent – підмножина недавніх повідомлень у повній формі, h_compressed – стиснене представлення старішої історії.

Критерії оптимізації: збереження часової когерентності, максимізація релевантності до поточного запиту (пріоритет семантично близьким до u_{n+1} повідомленням), мінімізація втрати критичної інформації (персональні факти, обмеження, специфічні вимоги).

Підзадача 2: Отримання та фільтрація знань (Knowledge Retrieval and Filtering). З бази знань K потрібно отримати найбільш релевантні документи та відфільтрувати надлишкову інформацію. Функція пошуку враховує поточний запит, контекст діалогу та профіль користувача:

$$D = f\_retrieval(u_{n+1}, H, P, K, k) \quad (2.3)$$

де D – множина з k найбільш релевантних документів/фрагментів.

Ключові виклики: балансування між повнотою охоплення теми та точністю (уникнення слабо релевантних документів). При роботі з довгими діалогами потрібен механізм видалення застарілої інформації.

Підзадача 3: Персоналізаційний відбір та ранжування (Personalized Selection and Ranking). На основі профілю P необхідно переоцінити та перерангувати фрагменти історії H і документи D для пріоритизації інформації, релевантної конкретному користувачу:

$$W = f\_personalize(H, D, P, u_{n+1}) \quad (2.4)$$

де W – вектор ваг для кожного елемента контексту.

Персоналізація враховує: рівень експертизи (початківець потребує детальніших пояснень, експерт-технічної глибини), специфічні інтереси, поведінкові патерни, доменні пріоритети.

Підзадача 4: Оптимальне композиція контексту (Context Composition). Маючи H, D та W, необхідно скомпонувати фінальний контекст C*:

$$C^* = f\_compose(H, D, W, u_{n+1}, P) = \{c\_system, c\_profile, c\_history, c\_knowledge, c\_query\} \quad (2.5)$$

де компоненти включають: c_system – системні інструкції; c_profile – резюме профілю; c_history – відібрана історія; c_knowledge – релевантні фрагменти; c_query – поточний запит.

Ключове питання – баланс між історією діалогу та зовнішніми знаннями. Для запитів, що продовжують попередню тему, пріоритет – історії; для нових тем – зовнішнім знанням. Коефіцієнт $\alpha \in [0,1]$ визначає частку контексту під історію.

2.1.3 Архітектурні обмеження та вимоги

При проектуванні системи необхідно врахувати ряд технічних та бізнесових обмежень, які суттєво впливають на вибір архітектурних рішень.

Обмеження продуктивності. Система має забезпечувати формування контексту зі швидкістю для підтримки інтерактивного діалогу. Прийнятним вважається латентність не більше 200-300 мс для процесу формування контексту (без врахування часу генерації відповіді мовною моделлю). При обробці 100 одночасних сесій система не повинна деградувати у продуктивності більш ніж на 20%. Це вимагає ефективних алгоритмів індексування та кешування проміжних результатів.

Обмеження масштабованості. Архітектура має підтримувати горизонтальне масштабування для зростаючої кількості користувачів та обсягу бази знань. Система має ефективно працювати з базою знань розміром до 1 млн документів (приблизно 500 млн токенів) та підтримувати до 100,000 активних користувачів з персональними профілями. Додавання нових користувачів або документів не повинно вимагати перебудови системи або значного зростання обчислювальних ресурсів.

Фінансові обмеження. Використання комерційних API створює операційні витрати, пропорційні кількості токенів. При середній вартості $0.01 за 1K токенів для GPT-4o-mini[19], діалогова система з 10,000 активних користувачів та середньою інтенсивністю 20 повідомлень на день генерує

значні місячні витрати. Оптимізація розміру контексту на 20-30% має прямий фінансовий ефект і є важливою бізнес-вимогою.

Вимоги до точності та якості. Система не повинна втрачати критичну інформацію при компресії чи фільтрації контексту. Це особливо важливо для персональних даних (бюджетні обмеження користувача, специфічні вимоги, попередні домовленості) та ключових фактів з предметної області. Метрика збереження критичної інформації (Critical Information Retention Rate) має становити не менше 95%.

Вимоги до прозорості та контрольованості. На відміну від fine-tuning або складних attention механізмів, система має забезпечувати аудит та пояснення рішень про формування контексту. Розробники та персонал повинні розуміти, чому фрагмент історії включений чи виключений з контексту, які ваги присвоєні компонентам. Це важливо для відлагодження, покращення системи та довіри користувачів.

2.1.4 Загальна стратегія вирішення

Для вирішення сформульованої задачі пропонується інтегрований підхід - Adaptive Context Management System (ACMS). Ключова ідея - застосування різних стратегій на різних рівнях управління контекстом та їх координація через персоналізаційний шар.

Стратегія 1: Sliding Window для недавньої історії. Останні N повідомлень (типово N=5-10) зберігаються у повній формі без компресії. Це забезпечує збереження деталізованого контексту для підтримання когерентності діалогу. Розмір вікна N визначається динамічно на основі доступного бюджету токенів після виділення місця під системні інструкції, профіль користувача та результати RAG.

Стратегія 2: LLM-based Compression для старої історії. Повідомлення, старіші за sliding window, агрегуються у стислі тематичні резюме за допомогою мовної моделі. Замість відкидання старої історії система генерує

параграф-резюме з ключовими фактами, рішеннями та контекстною інформацією. Для компресії використовується легка модель (GPT-4o-mini), що забезпечує баланс між якістю резюме та вартістю.

Стратегія 3: Personalization Layer для адаптивного відбору. Персоналізаційний шар є центральним компонентом та діє на всіх етапах формування контексту. Він використовує векторні подання профілю користувача та доменних конфігурацій для оцінки релевантності кожного фрагмента. Система приймає рішення про те, які елементи зберегти у повній формі, які компресувати чи відкинути; які документи мають найвищий пріоритет; як розподілити бюджет токенів між історією та зовнішніми знаннями.

Стратегія 4: Domain Prioritization для балансування знань. Система враховує специфіку предметної області через доменні конфігурації. Різні домени (фінанси, медицина, технічна підтримка) вимагають різного балансу між загальними принципами та конкретними фактами, між історією діалогу та зовнішніми знаннями.

Формування фінального контексту відбувається за формулою:

$$C^* = c\_system \oplus c\_profile \oplus h\_compressed \oplus h\_recent \oplus d\_personalized \quad (2.6)$$

де $\oplus$ позначає конкатенацію з урахуванням ваг та обмеження на загальний розмір.

Кожен компонент має власну вагу:

$$w\_system = 0.05 \text{ (системні інструкції)}$$
$$w\_profile = 0.05 \text{ (резюме профілю)}$$
$$w\_history = \alpha \times 0.40 \text{ (історія діалогу, де } \alpha \text{ – адаптивний коефіцієнт)}$$
$$w\_knowledge = (1-\alpha) \times 0.40 \text{ (зовнішні знання)}$$
$$w\_query = 0.10 \text{ (поточний запит)} \quad (2.7)$$

де коефіцієнт α динамічно адаптується на основі: типу запиту (продовження теми vs. новий запит), довжини діалогової сесії, наявності релевантних документів, персональних налаштувань користувача.

Така стратегія дозволяє гнучко балансувати між збереженням історії та залученням нових знань, між персоналізацією та загальною корисністю, між економією ресурсів та якістю відповідей. Кожна підзадача вирішується спеціалізованим модулем, а координація здійснюється через Adaptive Context Manager, який приймає рішення на основі стану діалогу, профілю користувача та доменних вимог.

2.2 Архітектура системи

2.2.1 Загальна архітектурна схема

Система ACMS реалізована як частина мікросервісної архітектури персоналізованого AI-асистента. Вибір мікросервісного підходу обумовлений необхідністю масштабованості, незалежного розгортання компонентів та ізоляції відмов. Архітектура складається з шести основних компонентів. *(рисунок Г.1)*

User Interface Layer (Шар користувацького інтерфейсу) представлений веб-застосунком або API endpoints для взаємодії користувачів із системою. Шар відповідає за прийом повідомлень, валідацію даних, управління сесіями та відображення відповідей. Підтримує синхронний режим (request-response) та асинхронний (streaming відповідей).

AI Orchestrator (Оркестратор AI) є центральним координуючим компонентом. Він отримує запит користувача, визначає послідовність дій, координує виклики сервісів та формує фінальну відповідь. Workflow обробки запиту: отримання повідомлення та ідентифікатора сесії; виклик ACMS для формування контексту; виклик Vector Search Service для отримання знань; формування промпту з урахуванням контексту; виклик OpenAI API для генерації відповіді; збереження у історію; повернення відповіді користувачу.

Оркестратор також відповідає за обробку помилок, retry логіку для зовнішніх сервісів, логування та моніторинг продуктивності.

Adaptive Context Management System (ACMS) – ядро розробленого рішення, основний внесок даної роботи. ACMS отримує ідентифікатор сесії та поточний запит, після чого виконує: завантаження історії діалогу з бази даних або кешу Redis; класифікацію повідомлень на недавні (для sliding window) та старі (для компресії); формування compressed summary для старих повідомлень; завантаження профілю користувача та доменної конфігурації; застосування персоналізаційного шару для оцінки релевантності; композицію фінального контексту з урахуванням бюджету токенів.

ACMS реалізований як окремий FastAPI сервіс з REST API. Детальна внутрішня архітектура описана у підрозділі 2.2.2.

Vector Search Service (Сервіс векторного пошуку) відповідає за пошук релевантних документів у базі знань. Сервіс використовує PostgreSQL з pgvector для зберігання та індексування векторних подань. Робота включає: отримання запиту та контексту; генерацію векторного подання через embedding модель (text-embedding-3-small); виконання пошуку k найближчих сусідів; опціональне re-ranking з урахуванням контексту; повернення топ-N документів з оцінками релевантності.

Реалізовано гібридний пошук, що комбінує векторний пошук (семантична подібність) з full-text search (точне співпадіння ключових слів). Це підвищує якість для запитів зі специфічними термінами.

User Profile Service (Сервіс профілів користувачів) управляє профілями та доменними конфігураціями. Профіль містить: ідентифікатор та метаінформацію; векторне подання інтересів (формується на основі історії взаємодій); доменні переваги; поведінкові метрики (середня довжина запитів, частота взаємодій, типи запитань); явні налаштування персоналізації.

Сервіс підтримує доменні конфігурації – набори правил для предметних областей (фінанси, медицина, техпідтримка), що визначають оптимальний баланс між історією та знаннями, патерни структурування відповідей, критичні аспекти персоналізації.

Data Storage Layer (Шар зберігання даних) об'єднує спеціалізовані сховища. PostgreSQL primary database зберігає структуровані дані: діалогові сесії, повідомлення, профілі, метадані документів. PostgreSQL + pgvector зберігає векторні подання документів з індексами для швидкого пошуку. Redis cache зберігає активні діалогові контексти, compressed summaries, векторні подання профілів для швидкого доступу.

Кешування критично важливе для продуктивності. Замість завантаження повної історії з PostgreSQL та генерації summary при кожному запиті, система використовує кеш у Redis, що зменшує латентність з 200-300ms до 20-50ms для активних діалогів.

2.2.2 Внутрішня архітектура ACMS

ACMS має модульну архітектуру, що відображає декомпозицію задачі на підзадачі. Кожен модуль реалізує специфічну функцію з чітким інтерфейсом взаємодії. *(рисунок Г.2)*

Dialog History Manager (Менеджер історії діалогу) відповідає за завантаження, організацію та обробку діалогової історії. При запиті на формування контексту виконує:

1. Завантаження історії сесії з PostgreSQL або Redis
2. Розділення історії на: Recent messages (N останніх пар, N=5-10), Middle history (для компресії), Ancient history (вже скомпресовані)
3. Обчислення метадані для кожного повідомлення: номер, timestamp, довжина в токенах, наявність критичних сутностей
4. Формування кандидатів для sliding window

Менеджер відстежує compressed summary – резюме старих частин діалогу. При появі нових повідомлень ініціює оновлення через Compression Module.

Compression Module (Модуль компресії) використовує мовну модель для створення стислих резюме фрагментів історії.

Extractive compression відбирає найбільш інформативні повідомлення на основі TF-IDF векторизації та оцінки релевантності. Стратегія швидка (не вимагає LLM) але може втрачати зв'язність.

Abstractive compression генерує параграф-резюме, що передає суть діалогу меншою кількістю токенів. Промпт сконструйований для збереження: ключових фактів та числових даних; персональних вподобань та обмежень; важливих рішень; контексту поточної теми.

Типовий промпт для компресії виглядає наступним чином:

Summarize the following conversation history, preserving: *1. Key facts, numbers, and dates mentioned by the user* *2. User's explicit preferences, requirements, or constraints* *3. Important decisions or agreements reached* *4. Current topic and its context* *Keep the summary concise (max 150 tokens) but complete.* *[CONVERSATION HISTORY]* *...* *[SUMMARY]*	*(2.8)*

GPT-4o-mini використовується для компресії, забезпечуючи прийнятну якість при низькій вартості ($0.075 за 1M input tokens та $0.30 за 1M output tokens). GPT-4-turbo коштує у 10 разів дорожче при незначному покращенні якості.

Personalization Layer (Персоналізаційний шар) є найбільш інноваційний компонент ACMS. Застосовує персоналізацію на трьох рівнях: відбір фрагментів історії, ранжування результатів RAG, балансування між історією та знаннями.

На рівні історії оцінює релевантність кожного повідомлення за формулою:

$score\_history(m\_i) = \alpha \times sim\_semantic(m\_i, q) + \beta \times sim\_profile(m\_i, P) + \gamma \times recency(m\_i)$	*(2.9)*

де:

- m_i – і-те повідомлення з історії
- q – поточний запит користувача
- P – векторне подання профілю користувача
- sim_semantic – косинусна подібність між векторами повідомлення та запиту
- sim_profile – косинусна подібність між повідомленням та профілем
- recency – функція, що зменшується з віком повідомлення
- α, β, γ – вагові коефіцієнти (типово α=0.5, β=0.3, γ=0.2)

Повідомлення з найвищими оцінками отримують пріоритет, що дозволяє зберегти не лише недавні, а й семантично релевантні повідомлення зі старої історії.

На рівні RAG результатів перераховує документи з урахуванням профілю:

$score\_doc(d\_j) = \lambda \times score\_retrieval(d\_j) + (1-\lambda) \times sim\_profile(d\_j, P)$	*(2.10)*

де score_retrieval – базова оцінка від векторного пошуку, λ – коефіцієнт балансу (типово λ=0.7).

На рівні балансування динамічно визначає коефіцієнт α для співвідношення історії та знань на основі: типу запиту; наявності релевантних документів у RAG; довжині сесії; персональних налаштувань користувача.

Context Composer (Композитор контексту) виконує фінальне збирання компонентів у промпт для мовної моделі.

Спершу визначається доступний бюджет токенів. Від L_max віднімаються фіксовані компоненти (системний промпт, поточний запит, резервний буфер), залишок розподіляється між історією та знаннями згідно з α. Потім кожен компонент форматується у текстовий блок з розділовими маркерами *(лістинг Д.1)*. Така структура забезпечує моделі чітке розуміння різних типів інформації.

Наприкінці композитор обчислює фактичний розмір контексту та перевіряє відповідність обмеженням. При перевищенні L_max виконується інкрементальне видалення найменш важливих елементів (спочатку RAG документи, потім стиснена історія).

Token Budget Manager (Менеджер бюджету токенів) відповідає за підрахунок токенів та розподіл простору між компонентами. Підтримує кілька токенізаторів (tiktoken для OpenAI, SentencePiece для Meta) та вибирає відповідний згідно з цільовою моделлю. Реалізує greedy allocation з пріоритизацією: системний промпт (обов'язковий); поточний запит (обов'язковий); резюме профілю (висока важливість); недавні повідомлення sliding window (висока важливість); топ-K документів RAG (середня важливість); compressed history (низька важливість). При нестачі місця застосовує truncation політику: документи обрізаються до перших N токенів; compressed summary додатково компресується; sliding window зменшується. Критична інформація (системні інструкції, поточний запит, профіль) ніколи не обрізається.

2.2.3 Інтеграція компонентів та потоки даних

Робота системи при обробці користувацького запиту включає складну взаємодію між компонентами. Розглянемо детальний workflow обробки типового запиту. *(рисунок Г.3)*

Крок 1: Прийом запиту. Користувач відправляє повідомлення через UI. Request містить текст повідомлення, ідентифікатор сесії (або створюється нова), опціональні метадані (тип пристрою, часова зона).

Крок 2: Аналіз запиту (Orchestrator). AI Orchestrator приймає запит та виконує попередній аналіз: класифікація типу запиту (продовження теми, новий запит, followup); виявлення intent (інформаційний запит, виконання задачі, творчий запит); визначення необхідності RAG.

Крок 3: Формування контексту (ACMS). Orchestrator викликає ACMS з параметрами: session_id, current_query, max_context_length, target_model. ACMS виконує процес, описаний у підрозділі 2.2.2, та повертає сформований контекст C* з метаданими (розподіл токенів, використаний α, компоненти контексту).

Крок 4: Пошук знань (Vector Search). Паралельно або послідовно відбувається звернення до Vector Search Service. Сервіс отримує запит та опціональний контекст діалогу, генерує embedding запиту, виконує векторний пошук у pgvector[17], застосовує re-ranking з урахуванням контексту, повертає топ-K документів з оцінками релевантності.

Крок 5: Інтеграція знань у контекст (ACMS). Повернуті документи передаються до ACMS для інтеграції у контекст з урахуванням персоналізації та бюджету токенів. На цьому етапі може відбутися перерозподіл простору: якщо документи дуже релевантні, α може бути зменшено для виділення їм більше місця.

Крок 6: Генерація відповіді (OpenAI API). Сформований контекст C* та запит відправляються до OpenAI API. Використовуються параметри: модель GPT-4o для складних запитів або GPT-4o-mini для простих; temperature=0.7 для балансу між креативністю та детермінізмом; max_tokens=1000 (обмеження для економії); опціональні function_calling для структурованих відповідей.

Крок 7: Пост-обробка та збереження. Отримана відповідь проходить пост-обробку: перевірка на некоректний контент; форматування (додавання посилань на використані документи); логування для аналітики. Повідомлення користувача та відповідь зберігаються у PostgreSQL для історії діалогу. Контекст оновлюється у Redis кеші для наступного запиту.

Крок 8: Повернення результату. Фінальна відповідь повертається користувачу через UI з опціональними метаданими (використані джерела, рівень впевненості, час обробки).

Важливим аспектом є обробка помилок та fallback механізми на кожному етапі. Якщо ACMS не може сформувати контекст через помилку (недоступна база даних), система використовує спрощену стратегію з останніми N повідомленнями. Якщо Vector Search недоступний, запит обробляється без зовнішніх знань (pure dialog mode). Якщо OpenAI API повертає помилку, використовується retry з exponential backoff або fallback на альтернативну модель.

2.3 Алгоритми та методи

2.3.1 Алгоритм адаптивного управління контекстом

Центральний алгоритм ACMS координує всі компоненти системи та приймає рішення про формування оптимального контексту. Алгоритм представлений у вигляді псевдокоду: *(лістинг Д.2)*

Ключові аспекти алгоритму:

Адаптивність досягається через динамічне визначення коефіцієнта α на основі типу запиту, історії діалогу та персональних налаштувань. Це дозволяє системі гнучко реагувати на зміну характеру розмови.

Пріоритизація реалізована на кількох рівнях: персоналізовані оцінки для повідомлень історії, re-ranking документів з бази знань, ієрархічне виділення бюджету (критичні компоненти отримують місце першими).

Ефективність забезпечується через використання кешованих результатів (compressed summary), інкрементальне оновлення контексту замість повного перерахунку, відкладене виконання дорогих операцій (компресія історії відбувається тільки коли необхідно).

Надійність досягається через fallback механізми на кожному етапі, перевірки інваріантів (розмір контексту не перевищує ліміт), graceful degradation (якщо недостатньо місця, система скорочує менш важливі компоненти, але не падає).

2.3.2 Метод персоналізованого ранжування

Персоналізаційний шар використовує векторні подання для оцінки релевантності контенту до профілю користувача. Розглянемо детальніше математичний апарат цього методу.

Векторне подання профілю. Профіль користувача представлений як точка у багатовимірному семантичному просторі. Це подання формується на основі агрегації векторів усіх повідомлень користувача у історії взаємодій:

$$P = Aggregate(\{emb(u_1), emb(u_2), ..., emb(u_n)\}) \quad (2.11)$$

де $emb(u_i)$ – embedding і-го повідомлення, отриманий від моделі text-embedding-3-small (розмірність 1536)[20]. Функція Aggregate може бути реалізована кількома способами:

Простим усередненням (Centroid method):

$$P = (1/n) \times \Sigma_i\, emb(u_i) \quad (2.12)$$

Зваженим усередненням з експоненціальним спаданням (Exponential decay):

$$P = \Sigma_i\, w_i \times emb(u_i), \text{ де } w_i = e^{(-\lambda \times age(u_i))} \quad (2.13)$$

Кластерним представленням (Multiple centroids):

$$P = \{C_1, C_2, ..., C_k\}, \text{ де } C_j \text{ – центроїди тематичних кластерів запитів користувача} \quad (2.14)$$

У поточній реалізації використовується експоненціальне спадання з $\lambda = 0.01$, що надає більшу вагу недавнім взаємодіям, але не ігнорує повністю старі.

Доменне подання. Додатково до персонального профілю, система підтримує доменні вектори, що представляють характеристики предметних областей:

D_domain = { *"finance": emb("financial planning investment risk portfolio"),* *"medicine": emb("diagnosis treatment symptoms patient health"),* *"tech_support": emb("troubleshooting error bug fix configuration"),* *...* *}*	*(2.15)*

Це дозволяє оцінювати релевантність контенту не лише до конкретного користувача, а й до предметної області в цілому.

*Функція персоналізованої оцінки***.** Для повідомлення з історії діалогу оцінка релевантності обчислюється як:

score_pers(m) = α × cos_sim(emb(m), emb(q)) + // semantic similarity to query *β × cos_sim(emb(m), P) + // similarity to user profile* *γ × cos_sim(emb(m), D_domain) + // domain relevance* *δ × recency_weight(m) // temporal factor*	*(2.16)*

де:

- emb(m) – embedding повідомлення
- emb(q) – embedding поточного запиту
- P – векторне подання профілю користувача
- D_domain – векторне подання активного домену
- cos_sim(a, b) = (a · b) / (||a|| × ||b||) – косинусна подібність[15]
- recency_weight(m) = e^(-λ × Δt), де Δt – час з моменту повідомлення
- α, β, γ, δ – вагові коефіцієнти (підбираються емпірично)

Типові значення коефіцієнтів: α = 0.4 (семантична близькість до запиту), β = 0.3 (відповідність профілю), γ = 0.2 (доменна релевантність), δ = 0.1 (свіжість).

Для документів з бази знань застосовується подібна формула, але без temporal factor (документи не мають часової мітки у контексті діалогу):

score_doc(d) = λ × score_retrieval(d) + // base retrieval score *μ × cos_sim(emb(d), P) + // profile similarity* *ν × cos_sim(emb(d), D_domain) // domain relevance*	*(2.17)*

score_doc(d) = λ × score_retrieval(d) + # base retrieval score μ × cos_sim(emb(d), P) + # profile similarity ν × cos_sim(emb(d), D_domain) # domain relevance

де score_retrieval(d) – базова оцінка від векторного пошуку (вже враховує подібність до запиту).

Адаптивні ваги. Важливим аспектом методу є те, що вагові коефіцієнти α, β, γ не є фіксованими, а адаптуються залежно від контексту:

Для нових користувачів (cold start) зменшується β, оскільки профіль ще недостатньо інформативний, а збільшується α та γ (більше уваги семантиці запиту та домену).

Для запитів типу "NEW_TOPIC" збільшується γ (доменна релевантність), оскільки користувач переходить до нової теми.

Для запитів типу "CONTINUATION" збільшується β (персональна релевантність), оскільки продовження теми має враховувати специфіку користувача.

Адаптація реалізована через просту heuristic *(лістинг Д.3)*.

Ефективність обчислень. Пряме обчислення косинусної подібності для всіх пар (повідомлення, запит) може бути повільним для довгих історій. Оптимізація досягається через:

1. Попереднє обчислення та кешування embedding для всіх повідомлень у історії (виконується один раз при збереженні повідомлення)
2. Batch обчислення подібностей через матричні операції (NumPy/PyTorch)
3. Фільтрацію кандидатів через approximate nearest neighbors (FAISS)[11] для дуже довгих історій

2.3.3 Метод компресії діалогової історії

Компресія історії є критично важливим компонентом для підтримки довготривалих діалогів. Розглянемо детальніше різні стратегії компресії та їх реалізацію. *(рисунок Г.4)*

Extractive compression (вилучаюча компресія). Ця стратегія відбирає найбільш інформативні повідомлення без зміни їх формулювання. Алгоритм базується на TextRank – графовому методі ранжування речень[10]. *(лістинг Д.4)*

Переваги extractive compression: швидкість (не потребує виклику LLM), збереження оригінального формулювання (немає ризику галюцинацій), простота реалізації. Недоліки: може порушувати зв'язність (вибіркове включення повідомлень створює "дірки" у діалозі), обмежений коефіцієнт компресії (не можна стиснути більше ніж до кількості повідомлень), втрата контекстних зв'язків.

Abstractive compression (абстрактна компресія). Використовує мовну модель для генерації стислого резюме, що передає суть діалогу меншою кількістю токенів.

Промпт для abstractive compression структурований так, щоб максимізувати збереження критичної інформації:

You are compressing a conversation history while preserving essential information. *CONVERSATION TO COMPRESS:* *[list of messages with timestamps and roles]* *CREATE A CONCISE SUMMARY (max 200 tokens) that MUST include:* *1. KEY FACTS mentioned by the user (numbers, dates, names, specific requirements)* *2. USER'S EXPLICIT PREFERENCES or constraints stated earlier* *3. IMPORTANT DECISIONS or agreements reached in the conversation* *4. CURRENT TOPIC and its immediate context* *Format: Single cohesive paragraph. Use past tense. Be specific with numbers and dates.* *SUMMARY:*	*(2.18)*

Для оцінки якості компресії використовується метрика information retention:

$IR = \|CriticalFacts(compressed) \cap CriticalFacts(original)\| / \|CriticalFacts(original)\|$	*(2.19)*

де CriticalFacts – множина критичних фактів (числа, дати, імена, обмеження), виявлених через named entity recognition.

Hybrid compression (гібридна компресія). Комбінує обидва підходи для оптимального результату:

1. Extractive фаза відбирає 40-50% найбільш інформативних повідомлень
2. Abstractive фаза генерує резюме з відібраних повідомлень
3. Результат об'єднується: критичні факти з extractive + зв'язний текст з abstractive

Цей підхід дозволяє досягти коефіцієнта компресії 60-70% при information retention > 90%.

Інкрементальна компресія. Для оптимізації продуктивності компресія виконується інкрементально, а не для всієї історії відразу. Коли sliding window зміщується і нові повідомлення стають "старими", вони додаються до існуючого compressed summary.

2.3.4 Алгоритм динамічного балансування контексту

Визначення оптимального балансу між історією діалогу та зовнішніми знаннями є складною задачею, що вимагає врахування множини факторів. Розроблений алгоритм адаптивно визначає коефіцієнт α на основі характеристик запиту та діалогу. *(рисунок Г.5 та рисунок Г.6)*

Ключові інсайти алгоритму:

Контекстуальна адаптація. Коефіцієнт α не є фіксованим для домену, а адаптується до кожного конкретного запиту. Це забезпечує гнучку реакцію на зміну характеру діалогу.

Множинні фактори. Рішення базується на комбінації факторів: семантичної близькості до попередніх повідомлень, якості результатів RAG, стадії діалогу, персональних налаштувань. Це робить систему більш робастною порівняно з heuristic на основі одного критерію.

Обмеження екстремальних значень. Навіть при сильних показниках у одному напрямку, α обмежений межами [0.2, 0.8]. Це означає, що завжди присутня як деяка частка історії (мінімум 20%), так і знань (мінімум 20%). Повне ігнорування однієї зі складових призводить до погіршення якості відповідей.

Прозорість рішень. Детальне логування всіх факторів та їх внесків дозволяє аналізувати поведінку системи та налаштовувати вагові коефіцієнти на основі емпіричних даних.

2.4 Технічна реалізація

2.4.1 Технологічний стек

Система реалізована з використанням наступних технологій:

Backend: FastAPI (Python 3.11+) – асинхронний фреймворк з автоматичною генерацією OpenAPI документації та вбудованою валідацією даних через Pydantic.

База даних: PostgreSQL 15+ з розширенням pgvector для зберігання та індексування векторних подань[17]. Основні таблиці включають dialog_sessions, messages (з embedding vector(1536)), user_profiles та knowledge_base.

Кешування: Redis 7+ для зберігання активних контекстів сесій (TTL 1 година), compressed summaries (TTL 24 години) та профілів користувачів (TTL 6 годин).

LLM API: OpenAI GPT-4o для генерації відповідей та GPT-4o-mini[19] для компресії історії діалогу.

Embeddings: text-embedding-3-small (розмірність 1536, вартість $0.02/1M токенів) для векторних подань запитів та документів.

Моніторинг: Prometheus для метрик, Grafana для візуалізації, ELK Stack для централізованого логування.

Вибір pgvector замість спеціалізованих векторних баз даних обумовлений консолідацією даних у єдиній базі, зниженням операційної складності та достатньою продуктивністю для пошуку у мільйонах векторів.

2.4.2 Ключові компоненти реалізації

Система організована як мікросервісна архітектура з наступними основними сервісами:

ACMS Service – ядро системи, реалізує п'ять основних модулів:

- HistoryManager – завантаження та розділення історії
- CompressionModule – abstractive та extractive компресія

- PersonalizationLayer – персоналізоване ранжування
- ContextComposer – формування фінального промпту
- TokenBudgetManager – управління бюджетом токенів

Vector Search Service – векторний пошук у базі знань з використанням pgvector IVFFlat індексу та гібридним пошуком (векторний + full-text).

Orchestrator Service – координація workflow обробки запитів, виклик ACMS та Vector Search, інтеграція з OpenAI API.

Profile Service – управління профілями користувачів та доменними конфігураціями.

Висновки до розділу 2

У цьому розділі представлено детальний опис розробленої системи адаптивного управління контекстом (ACMS) для персоналізованих RAG-асистентів. Основні результати включають:

1. Формалізація задачі: Проведено математичну формалізацію проблеми управління контекстом з визначенням функції відбору контексту та множини обмежень (розмір, вартість, якість, персоналізація).
2. Декомпозиція на підзадачі: Задача розділена на чотири взаємопов'язані підзадачі (управління історією, отримання знань, персоналізація, композиція), кожна з яких має власні критерії оптимізації та методи вирішення.
3. Архітектура системи: Розроблено мікросервісну архітектуру з шістьма основними компонентами, що забезпечує масштабованість, надійність та можливість незалежного розгортання. Детально описано внутрішню архітектуру ACMS з п'ятьма спеціалізованими модулями.
4. Алгоритми та методи: Представлено центральний алгоритм адаптивного управління контекстом, метод персоналізованого ранжування на основі векторних подань, стратегії компресії діалогової історії та алгоритм динамічного балансування між історією та знаннями.

5. Технічна реалізація: Обґрунтовано вибір технологічного стеку (FastAPI, PostgreSQL + pgvector, Redis, OpenAI API), описано схему бази даних, структуру проекту та підхід до розгортання в Kubernetes.

Розроблена система забезпечує адаптивне управління контекстом на кількох рівнях: на рівні відбору історичних повідомлень (персоналізовані оцінки релевантності), на рівні отримання знань (персоналізований re-ranking документів), на рівні балансування (динамічний коефіцієнт α). Це дозволяє формувати контекст, оптимізований для конкретного користувача, запиту та предметної області, при дотриманні обмежень на розмір та вартість обробки.

РОЗДІЛ 3
ЕКСПЕРИМЕНТАЛЬНА ОЦІНКА СИСТЕМИ

3.1 Методологія експериментальної оцінки

3.1.1 Формування тестового набору

Для оцінки ефективності ACMS сформовано тестовий набір із 100 діалогових сесій у двох предметних областях: фінансове консультування та технічна підтримка. Кожна сесія містить 10-25 повідомлень, що відповідає типовим консультаційним діалогам.

Таблиця 3.1.

Характеристики тестового набору

Параметр	**Фінанси**	**Техпідтримка**	**Загалом**
Кількість сесій	50	50	100
Середня довжина сесії	18.3	15.7	17.0
Діапазон повідомлень	12-25	10-22	10-25
Унікальних користувачів	25	25	50
Документів у базі знань	487	523	1010

Профілі користувачів включають три рівні експертизи (початківець, середній, експерт) та історію попередніх взаємодій від 0 до 50 сесій.

Таблиця 3.2.

Порівняння конфігурацій Baseline та ACMS

Компонент	**Baseline**	**ACMS**
Управління історією	Повна історія	Sliding window + компресія

RAG-пошук	Топ-5 за подібністю	Персоналізований re-ranking
Розподіл бюджету	Фіксований 50/50	Динамічний α
Персоналізація	Відсутня	Трирівнева

3.1.2 Система метрик оцінювання

Для комплексної оцінки використано чотири метрики:

Relevance@5 – частка з п'яти найрелевантніших документів, що дійсно корисні для відповіді (оцінка LLM-as-a-Judge).

Context Coherence – оцінка зв'язності сформованого контексту за шкалою 0-1, що враховує логічну послідовність та відсутність суперечностей.

Token Economy – відношення кількості токенів у контексті ACMS до baseline. Значення 0.78 означає економію 22%.

Answer Win-Rate – частка випадків, коли відповідь ACMS оцінена краще за baseline у парному порівнянні.

Оцінювання виконано за протоколом LLM-as-a-Judge з використанням GPT-4o як арбітра[13]. Для підвищення стабільності кожне оцінювання виконувалось тричі при temperature=0 з подальшим усередненням.

3.1.3 Baseline конфігурація

Baseline система реалізує типовий RAG-підхід: повна історія діалогу включається у контекст, топ-5 документів відбираються за косинусною подібністю до поточного запиту, фіксований розподіл бюджету 50% на історію та 50% на документи, персоналізація відсутня.

3.2 Кількісні результати експериментів

3.2.1 Загальні результати

Таблиця 3.3

Порівняння загальних результатів

Метрика	**Baseline**	**ACMS**	**Δ**	**p-value**
Relevance@5	0.74	0.84	+13.5%	< 0.001
Context Coherence	0.68	0.77	+13.2%	< 0.001
Token Economy	1.00	0.78	−22%	-
Answer Win-Rate	-	0.62	-	< 0.01

ACMS демонструє статистично значуще покращення за всіма метриками[13]. Підвищення Relevance@5 на 13.5% свідчить про ефективність персоналізованого відбору контенту. Економія 22% токенів досягається без втрати якості. *(рисунок Г.7)*

3.2.2 Результати за предметними областями

Таблиця 3.4

Результати за доменами

Метрика	**Фінанси Baseline**	**Фінанси ACMS**	**Δ**	**Техпідтримка Baseline**	**Техпідтримка ACMS**	**Δ**
Relevance@5	0.71	0.86	+21%	0.77	0.82	+6%
Context Coherence	0.65	0.79	-22%	0.71	0.75	+6%

Token Economy	1.00	0.75	−25%	1.00	0.81	−19%

Фінансовий домен демонструє більше покращення (+21% Relevance@5), оскільки персоналізація ефективніша при варіативності рівнів експертизи користувачів. У технічній підтримці покращення менш виражене (+6%), що пояснюється більш стандартизованим характером запитів.

3.2.3 Результати за рівнем експертизи та довжиною діалогу

Таблиця 3.5

Результати за рівнем експертизи користувачів

Рівень експертизи	**Baseline Relevance@5**	**ACMS Relevance@5**	**Δ**
Початківець	0.69	0.84	+21%
Середній	0.75	0.84	+12%
Експерт	0.79	0.84	+6%

Найбільше покращення спостерігається для початківців (+21%), оскільки персоналізація адаптує рівень деталізації відповідей.

Таблиця 3.6

Результати за довжиною діалогу

Довжина діалогу	**Baseline Relevance@5**	**ACMS Relevance@5**	**Δ**
Короткий (10-14)	0.76	0.82	+8%
Середній (15-19)	0.74	0.85	+15%
Довгий (20-25)	0.68	0.85	+25%

Перевага ACMS зростає з довжиною діалогу: для довгих сесій покращення становить +25%, що підтверджує ефективність механізмів компресії та персоналізованого відбору.

3.3 Аналіз продуктивності системи

3.3.1 Часові характеристики

Таблиця 3.7

Часові характеристики обробки запитів

Операція	**З кешем (мс)**	**Без кешу (мс)**
Завантаження історії	5-10	25-40
Компресія (якщо потрібна)	0	400-600
Персоналізований відбір	15-25	15-25
RAG-пошук	30-45	30-45
Композиція контексту	5-10	5-10
Загалом	**85-115**	**140-195**
95-й перцентиль	140	250

Загальний час формування контексту (85-195 мс) є прийнятним для інтерактивних застосувань та не створює помітної затримки для користувача.

3.3.2 Економічний ефект

Таблиця 3.8

Розрахунок економічного ефекту

Параметр	**Baseline**	**ACMS**	**Економія**
Середній розмір контексту	8,200 токенів	6,400 токенів	1,800 токенів
Вартість за запит (GPT-4o)	$0.082	$0.064	$0.018
Вартість за 10K запитів/день	$820	$640	$180
Річна економія	-	-	**$65,700**

При типовому навантаженні 10,000 запитів на день економія становить приблизно $180 щодня або $65,700 на рік. Додаткові витрати на компресію (GPT-4o-mini, ~$0.001 за операцію) є незначними порівняно з економією.

3.5 Обмеження

Експериментальна оцінка виявила ряд обмежень запропонованого підходу:

Cold Start – для нових користувачів (< 5 сесій) профіль недостатньо інформативний, покращення становить лише +5% замість +15% для досвідчених.

Втрата технічних деталей – компресія може втрачати специфічні параметри (коди помилок, версії ПЗ) у 18% випадків.

Обмежена кількість доменів – апробація на двох доменах не гарантує узагальнюваність на медицину, юриспруденцію тощо.

Ручний підбір гіперпараметрів – коефіцієнти α, β, γ підбирались експертно, автоматичне налаштування є напрямком подальших досліджень.

Висновки до розділу 3

Експериментальна оцінка системи ACMS на тестовому наборі зі 100 діалогових сесій у двох предметних областях підтвердила ефективність запропонованого підходу. Основні результати: підвищення Relevance@5 на 13.5%, покращення Context Coherence на 13.2%, економія токенів 22% та Answer Win-Rate 0.62. Найбільший внесок забезпечує Personalization Layer (+0.05 у аблаційному дослідженні). Перевага ACMS зростає для довгих діалогів (+25%) та користувачів-початківців (+21%). Виявлені обмеження (cold start, втрата технічних деталей) визначають напрямки подальшого вдосконалення.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну науково-практичну задачу розробки системи адаптивного управління контекстом у RAG-архітектурах для персоналізованих AI-асистентів. Актуальність задачі обумовлена стрімким зростанням використання діалогових систем на основі великих мовних моделей та необхідністю забезпечення високої якості персоналізованих відповідей у довготривалих діалогах при оптимальному використанні обчислювальних ресурсів. Проведене дослідження дозволяє сформулювати наступні висновки.

1. Проведено комплексний аналіз існуючих підходів до управління контекстом у діалогових системах на основі великих мовних моделей, включаючи методи sliding window, compression-based підходи, системи з явною пам'яттю (MemGPT, LangChain Memory) та персоналізований пошук. Аналіз виявив, що жоден з окремих методів не забезпечує комплексного вирішення проблеми ефективного використання контекстного вікна у довготривалих персоналізованих діалогах. Існуючі рішення або жертвують історичним контекстом заради економії токенів (sliding window), або не забезпечують достатньої економії при збереженні інформації (memory systems), або не враховують персональні характеристики користувача при формуванні контексту (compression-based). Це обґрунтувало необхідність розробки інтегрованого підходу, що поєднує переваги різних методів.
2. Розроблено архітектуру системи адаптивного управління контекстом (Adaptive Context Management System, ACMS), що включає чотири ключові компоненти: Dialog History Manager для управління історією діалогу з механізмом ковзного вікна динамічного розміру, Compression Module для LLM-компресії старих фрагментів історії з використанням моделі GPT-4o-mini, Personalization Layer для персоналізованого відбору

та ранжування контенту на основі векторних подань профілю користувача, та Context Composer для оптимальної композиції фінального контексту з урахуванням бюджету токенів. Архітектура реалізована як мікросервісна система з використанням FastAPI, PostgreSQL з розширенням pgvector та Redis для кешування, що забезпечує масштабованість, надійність та можливість незалежного розвитку компонентів.

3. Розроблено алгоритм адаптивного управління контекстом, що динамічно визначає баланс між збереженням історії діалогу та залученням зовнішніх знань. Коефіцієнт балансування α адаптується в межах [0.2, 0.8] на основі кількох факторів: семантичної близькості поточного запиту до попередніх повідомлень (продовження теми vs. нова тема), якості результатів RAG-пошуку (за наявності високорелевантних документів α знижується), стадії діалогу (на початку більша вага знань, пізніше – історії) та персональних налаштувань користувача. Алгоритм забезпечує прозорість прийняття рішень через детальне логування всіх факторів та їх внесків.
4. Реалізовано метод персоналізованого ранжування на основі векторних подань, що працює на трьох рівнях: персоналізація відбору фрагментів історії, персоналізація ранжування документів з бази знань, та динамічне балансування між історією і знаннями. Для кожного фрагмента контексту обчислюється персоналізована оцінка релевантності за формулою: $score = \alpha \times sim(fragment, query) + \beta \times sim(fragment, profile) + \gamma \times domain\_relevance$, де коефіцієнти α, β, γ адаптуються залежно від типу запиту та стадії "дозрівання" профілю користувача. Профіль користувача формується як зважене середнє векторних подань його повідомлень з експоненціальним згладжуванням ($\lambda = 0.9$), що надає більшу вагу недавнім взаємодіям.

5. Створено повнофункціональну програмну реалізацію системи з використанням сучасного технологічного стеку: FastAPI (Python 3.11+) для побудови асинхронного REST API з автоматичною валідацією через Pydantic, PostgreSQL 15 з розширенням pgvector та індексом HNSW[12] для ефективного зберігання та векторного пошуку документів (підтримка до 1 млн документів), Redis 7 для кешування активних контекстів з TTL-політикою (зменшення латентності з 115 мс до 85 мс для активних сесій), OpenAI API (GPT-4o-mini для компресії, text-embedding-3-small для embeddings). Система забезпечує throughput до 1380 запитів на секунду при 200 одночасних сесіях.
6. Проведено комплексну експериментальну оцінку системи на тестовому наборі зі 100 діалогових сесій, рівномірно розподілених між доменами фінансового консультування та технічної підтримки. Оцінка виконувалась за протоколом LLM-as-a-Judge[13] з використанням чотирьох метрик. Результати підтвердили ефективність запропонованого підходу за всіма метриками з високою статистичною значущістю ($p < 0.001$):
    - підвищення Relevance@5 з 0.74 до 0.84 (+13.5%), що означає збільшення частки релевантних фрагментів у топ-5 контексту з 3.7 до 4.2;
    - покращення Context Coherence з 0.68 до 0.77 (+13.2%), що свідчить про збереження логічної зв'язності при оптимізації контексту;
    - скорочення довжини контексту з 8200 до 6400 токенів (Token Economy 0.78), що забезпечує 22% економію;
    - Answer Win-Rate 0.62, що означає перевагу відповідей з оптимізованим контекстом у 62% парних порівнянь.
7. Аблаційне дослідження дозволило кількісно оцінити внесок кожного компонента системи: Sliding Window забезпечує +0.02 до Relevance@5, Compression Module додає +0.03, Personalization Layer - +0.05.

Найбільший внесок персоналізаційного шару підтверджує центральну роль персоналізації у запропонованому підході та валідує основну гіпотезу дослідження про те, що врахування індивідуальних характеристик користувача суттєво покращує якість формування контексту. Додатковий аналіз показав, що покращення найбільш виражені для довгих діалогів (+25% Relevance@5), фінансового домену (+21%) та користувачів-початківців (+21%).

8. Практична цінність роботи полягає у створенні готового до впровадження рішення, яке може бути інтегроване у існуючі RAG-системи без модифікації або fine-tuning базової мовної моделі. Система працює як проміжний шар (middleware) між користувацьким інтерфейсом та мовною моделлю, що забезпечує універсальність та легкість інтеграції. Розрахункова економія операційних витрат при типовому навантаженні 10,000 запитів на день становить приблизно $65,700 на рік за рахунок скорочення кількості токенів у контексті. Латентність формування контексту (85-115 мс) забезпечує придатність системи для інтерактивних застосувань.

Виявлені в ході дослідження обмеження визначають перспективні напрямки подальших досліджень:

- автоматичне налаштування гіперпараметрів системи (розмір sliding window, пороги компресії, вагові коефіцієнти персоналізації) через байєсівську оптимізацію або онлайн навчання з підкріпленням на основі зворотного зв'язку користувачів;
- розробка механізмів подолання cold start проблеми для нових користувачів через використання трансферу знань з подібних профілів або інтерактивного збору уподобань;
- розширення системи для підтримки мультимодального контенту (зображення, таблиці, діаграми) у базі знань та історії діалогу;

- дослідження ефективності підходу у ширшому спектрі предметних областей, зокрема медицині, юриспруденції та освіті;
- інтеграція механізмів виявлення та корекції потенційних галюцинацій при компресії історії через перехресну валідацію з оригінальним текстом;
- оптимізація модуля компресії для покращення збереження технічних деталей у спеціалізованих доменах.

Таким чином, поставлена мета роботи - розробка системи адаптивного управління контекстом для персоналізованих RAG-асистентів, що забезпечує баланс між збереженням релевантної історії діалогу та залученням зовнішніх знань з урахуванням персональних характеристик користувача – досягнута. Всі визначені у вступі завдання виконано у повному обсязі: проаналізовано існуючі підходи, розроблено архітектуру та алгоритми, створено програмну реалізацію, проведено експериментальну оцінку. Результати підтверджено статистично значущими експериментальними даними, що демонструють покращення якості на 13-21% при одночасній економії ресурсів на 22%.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Liu N., Lin K., Hewitt J., Paranjape A., Bevilacqua M., Petroni F., Liang P. Lost in the Middle: How Language Models Use Long Contexts. *Transactions of the Association for Computational Linguistics*. 2024. Vol. 12. P. 157–173. URL: https://arxiv.org/abs/2307.03172 (дата звернення: 15.09.2025).
2. Lewis P., Perez E., Piktus A., Petroni F., Karpukhin V., Goyal N., Küttler H., Lewis M., Yih W., Rocktäschel T., Riedel S., Kiela D. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Proceedings of NeurIPS 2020*. URL: https://arxiv.org/abs/2005.11401 (дата звернення: 18.09.2025).
3. Borgeaud S., Mensch A., Hoffmann J. et al. Improving language models by retrieving from trillions of tokens. *Proceedings of ICML 2022*. URL: https://arxiv.org/abs/2112.04426 (дата звернення: 22.09.2025).
4. Xu F., Shi W., Choi E. Recomp: Improving Retrieval-Augmented LMs with Compression and Selective Augmentation. *Proceedings of ICLR 2024*. URL: https://arxiv.org/abs/2310.04408 (дата звернення: 01.10.2025).
5. Packer C., Vivek S., Fang C., Wooders S., Gonzalez J. E. MemGPT: Towards LLMs as Operating Systems. *Proceedings of ICLR 2024*. URL: https://arxiv.org/abs/2310.08560 (дата звернення: 01.10.2025).
6. Khandelwal U., Levy O., Jurafsky D., Zettlemoyer L., Lewis M. Generalization through Memorization: Nearest Neighbor Language Models. *Proceedings of ICLR 2020*. URL: https://arxiv.org/abs/1911.00172 (дата звернення: 05.10.2025).
7. Zhang S., Dinan E., Urbanek J., Szlam A., Kiela D., Weston J. Personalizing Dialogue Agents: I have a dog, do you have pets too? *Proceedings of ACL 2018*. URL: https://arxiv.org/abs/1801.07243 (дата звернення: 10.10.2025).

8. Mazaré P., Humeau S., Raison M., Bordes A. Training Millions of Personalized Dialogue Agents. *Proceedings of EMNLP 2018*. URL: https://arxiv.org/abs/1809.01984 (дата звернення: 10.10.2025).
9. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I. Attention Is All You Need. *Proceedings of NeurIPS 2017*. URL: https://arxiv.org/abs/1706.03762 (дата звернення: 12.10.2025).
10. Mihalcea R., Tarau P. TextRank: Bringing Order into Texts. *Proceedings of EMNLP 2004*. P. 404–411. URL: https://aclanthology.org/W04-3252/ (дата звернення: 15.10.2025).
11. Johnson J., Douze M., Jégou H. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*. 2021. Vol. 7, No. 3. P. 535–547. URL: https://arxiv.org/abs/1702.08734 (дата звернення: 20.10.2025).
12. Malkov Y. A., Yashunin D. A. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2020. Vol. 42, No. 4. P. 824–836. URL: https://arxiv.org/abs/1603.09320 (дата звернення: 20.10.2025).
13. Zheng L., Chiang W.-L., Sheng Y. et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. *Proceedings of NeurIPS 2023*. URL: https://arxiv.org/abs/2306.05685 (дата звернення: 25.10.2025).
14. Gao L., Ma X., Lin J., Callan J. Precise Zero-Shot Dense Retrieval without Relevance Labels. *Proceedings of ACL 2023*. URL: https://arxiv.org/abs/2212.10496 (дата звернення: 28.10.2025).
15. Karpukhin V., Oguz B., Min S., Lewis P., Wu L., Edunov S., Chen D., Yih W. Dense Passage Retrieval for Open-Domain Question Answering. *Proceedings of EMNLP 2020*. URL: https://arxiv.org/abs/2004.04906 (дата звернення: 01.11.2025).

16. LangChain Documentation. Memory Management. URL: https://python.langchain.com/docs/concepts/memory/ (дата звернення: 05.11.2025).
17. pgvector: Open-source vector similarity search for Postgres. URL: https://github.com/pgvector/pgvector (дата звернення: 15.11.2025).
18. Carlini N., Ippolito D., Jagielski M., Lee K., Tramer F., Zhang C. Quantifying Memorization Across Neural Language Models. *Proceedings of ICLR 2023*. URL: https://arxiv.org/abs/2202.07646 (дата звернення: 18.11.2025).
19. OpenAI. GPT-4 Technical Report. 2023. URL: https://arxiv.org/abs/2303.08774 (дата звернення: 20.11.2025).

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) ***Магістр***
Галузь знань: 12 – Інформаційні технології
Спеціальність: 123 «Комп'ютерна інженерія»
Освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних
систем та робототехніки
к. ф.-м. н., доц. ХРУСЛОВ М. М.
«02» жовтня 2024року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Супрун Андрій Сергійович
(прізвище, ім'я, по батькові студента)

1. Тема роботи **Адаптивне управління контекстом у RAG-системах для персоналізованих AI-асистентів**
керівник роботи **Бакуменко Ніна Станіславівна, к.т.н., доцент**,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету ***від 30 вересня 2025 року № 4101-5/3554***
2. Строк подання студентом роботи ***30 листопада 2025 року***
3. Перелік питань, які потрібно розробити)
   1. Дослідження теоретичних основ RAG-систем і методів управління контекстом у діалогових AI-системах
   2. Аналіз існуючих підходів до контекстної компресії та персоналізації в LLM-системах
   3. Розробка архітектури адаптивної системи управління контекстом (ACMS) для мікросервісної архітектури
   4. Проектування алгоритму sliding window для динамічного управління історією повідомлень
   5. Реалізація модуля LLM-based compression для агрегації старих повідомлень у тематичні підсумки
   6. Розробка персоналізаційного шару (Personalization Layer) з урахуванням embedding-профілів користувачів
   7. Інтеграція ACMS з існуючими компонентами системи (AI Orchestrator, Vector Search Service)
   8. Розробка та впровадження методики оцінювання якості контексту за протоколом LLM-as-a-Judge
   9. Проведення тестування системи на синтетичних діалогах у доменах фінансів, освіти та технічної підтримки

10. Порівняльний аналіз ефективності адаптивної та базової моделей за метриками relevance@5, context coherence, Token Economy
11. Розширення апробації системи: додаткові тести в нових доменах і збільшення кількості запитів
12. Розробка механізму автоматичного налаштування глибини компресії залежно від контексту діалогу

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Огляд літератури з RAG-систем, контекстного управління та LLM-компресії	01.09.2025 - 10.09.2025
2	Проектування архітектури ACMS, визначення API-інтерфейсів та моделі даних	10.09.2025 - 20.09.2025
3	Реалізація модуля sliding window та інтеграція з PostgreSQL + pgvector	20.09.2025 - 30.09.2025
4	Розробка LLM-based compression з використанням GPT-4o-mini API	01.10.2025 - 10.10.2025
5	Імплементація персоналізаційного шару та embedding-профілів користувачів	10.10.2025 - 15.10.2025
6	Інтеграція ACMS з AI Orchestrator та Vector Search Service	15.10.2025 - 20.10.2025
7	Розробка методики оцінювання LLM-as-a-Judge та початкова апробація	20.10.2025 - 28.10.2025
8	Підготовка та публікація статті	28.10.2025 - 31.10.2025
9	Розширена апробація: тестування в нових доменах (медицина, право)	01.11.2025 - 10.11.2025
10	Розробка механізму автоматичного налаштування параметрів компресії	10.11.2025 - 15.11.2025
11	Оптимізація продуктивності системи та налаштування кешування	15.11.2025 - 20.11.2025
12	Фінальний аналіз результатів та підготовка висновків	20.11.2025 - 23.11.2025
13	Оформлення пояснювальної записки відповідно до вимог МОН України	23.11.2025 - 28.11.2025
14	Підготовка презентації та представлення роботи керівнику	28.11.2025 - 30.11.2025

5. Дата видачі завдання ***02 жовтня 2024 року.***

Студент **А.С. Супрун**
ініціали, прізвище підпис

Керівник роботи Н.С. Бакуменко
ініціали, прізвище підпис

Додаток Б

ІНДИВІДУАЛЬНЕ ТЕХНІЧНЕ ЗАВДАННЯ

Технічне завдання на розробку “ACMS”

Розділ	**Зміст**
1. Введення	**Назва:** Система адаптивного управління контекстом у RAG-системах (ACMS) **Галузь:** Інформаційні технології, AI, діалогові системи
2. Підстава	Навчальний план за спеціальністю 123 «Комп'ютерна інженерія» Завдання на кваліфікаційну роботу магістра
3. Призначення	**Мета:** Розробка системи адаптивного управління контекстом для RAG-асистентів **Призначення:** Підвищення релевантності відповідей, скорочення витрат токенів, покращення UX **Вихідні дані:** Класичні RAG з наївним управлінням контекстом

4. Технічні вимоги	**Функціональні:** Автоматичне формування контексту, балансування історії/знань, персоналізація, компресія, векторний пошук **Надійність:** Critical Info Retention ≥95%, fallback механізми **Експлуатація:** База до 1M документів, 100K користувачів **Технічні засоби:** CPU 4+ cores, RAM 16+ GB, PostgreSQL 14+, Redis 6+ **Програмна сумісність:** Python 3.10+, FastAPI, pgvector, OpenAI API **Спеціальні:** Стабільний інтернет, захист персональних даних
5. Документація	Технічне завдання (Додаток Б) Програма і методика випробувань (Додаток В) Опис програми (Розділ 3) Тексти програм (Додаток Г)
6. Техніко-економічні	**Якісні:** Relevance@5 +13.5%, Coherence +13.2%, Win-Rate 62% **Кількісні:** Token Economy 0.78 (економія 22%, ~$1K/рік)
7. Етапи	13-20.10: Аналіз підходів 21-27.10: Розробка архітектури 28.10-10.11: Програмна реалізація 11-17.11: Апробація 18-23.11: Документація

8. Контроль	Перевірка керівником 1 раз/тиждень Випробування на тестовому наборі Захист на засіданні комісії

Студент А.С. Супрун 
ініціали, прізвище підпис

Керівник роботи Н.С. Бакуменко 
ініціали, прізвище підпис

Додаток В

ПРОГРАМНА РЕАЛІЗАЦІЯ

Система реалізована як мікросервісна архітектура:

Технологічний стек:

- Backend: FastAPI для REST API
- Векторна БД: PostgreSQL 14 + pgvector (HNSW індекси)
- Кешування: Redis (TTL 24 год для активних діалогів)
- LLM: GPT-4o-mini для компресії історії
- Embeddings: text-embedding-3-small (1536 dim)
- Мова: Python 3.10+

Компоненти:

1. **Dialog History Manager:** Sliding window (N=5-10), розділення на recent/middle/ancient
2. **Compression Module:** Hybrid compression (extractive + abstractive), коефіцієнт 60-70%
3. **Personalization Layer:** Векторні профілі з експоненціальним спаданням (λ=0.01), адаптивні ваги
4. **Context Composer:** Інкрементальне truncation, фінальний промпт з балансуванням α
5. **Vector Search Service:** Гібридний пошук (dense + BM25), re-ranking

Архітектура БД:

- Таблиці: sessions, messages, user_profiles, documents, embeddings
- Індекси: HNSW для векторного пошуку, B-tree для timestamp
- Redis: кеш compressed summaries, активні контексти

Workflow обробки запиту:

1. AI Orchestrator приймає запит
2. ACMS формує контекст (sliding window + compression + personalization)
3. Vector Search шукає релевантні документи

4. Personalization Layer переранжовує та балансує
5. Context Composer збирає фінальний промпт
6. GPT-4o генерує відповідь
7. Оновлення історії та кешу

РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТІВ

Набір даних: 100 запитів у 2 доменах (фінанси 50, техпідтримка 50), діалоги 15-25 кроків.

Основні результати:

Метрика	**Baseline**	**ACMS**	**Поліпшення**
Relevance@5	0.74	0.84	+13.5%
Context Coherence	0.68	0.77	+13.2%
Token Economy	1.00	0.78	-22%
Answer Win-Rate	-	0.62	-

Деталізація за доменами:

Фінанси: relevance@5 0.71→0.86 (+21%), найбільший ефект персоналізації

Техпідтримка: relevance@5 0.77→0.82 (+6%), краще утримання контексту процедур

Аблаційне дослідження:

Sliding Window alone: 0.76

- Compression: 0.79 (+0.03)
- Personalization: 0.84 (+0.05) ← найбільший внесок

Економіка: При 10K запитів/день економія ~$1,000/рік на API.

ВИСНОВОК

У результаті переддипломної практики виконана повна розробка системи ACMS для персоналізованих AI-асистентів.

Основні досягнення:

1. Проаналізовано існуючі підходи (RAG, sliding window, compression, memory systems, персоналізація)
2. Розроблена архітектура ACMS з модулями Dialog History Manager, Compression Module, Personalization Layer
3. Створені алгоритми адаптивного управління контекстом з персоналізованим ранжуванням
4. Виконана програмна реалізація з FastAPI, PostgreSQL + pgvector, Redis, GPT-4o-mini
5. Проведена експериментальна апробація на 100 запитах у 2 доменах

Результати апробації:

- Relevance@5: +13.5% (0.74 → 0.84)
- Context Coherence: +13.2% (0.68 → 0.77)
- Token Economy: 0.78 (економія 22%, ~$1K/рік)
- Answer Win-Rate: 62%

Ключові фактори успіху:

- Адаптивне балансування історії/знань через динамічний α
- Персоналізований відбір на основі векторних профілів (+0.05 до relevance)
- Ефективна hybrid compression (60-70% з IR>90%)
- Модульна архітектура з прозорими рішеннями

Система готова до інтеграції у реальні AI-асистенти. Всі завдання практики виконані. Робота готова до передзахисту.

Студент А.С. Супрун
ініціали, прізвище підпис

Керівник роботи Н.С. Бакуменко _____
ініціали, прізвище підпис

Додаток Г

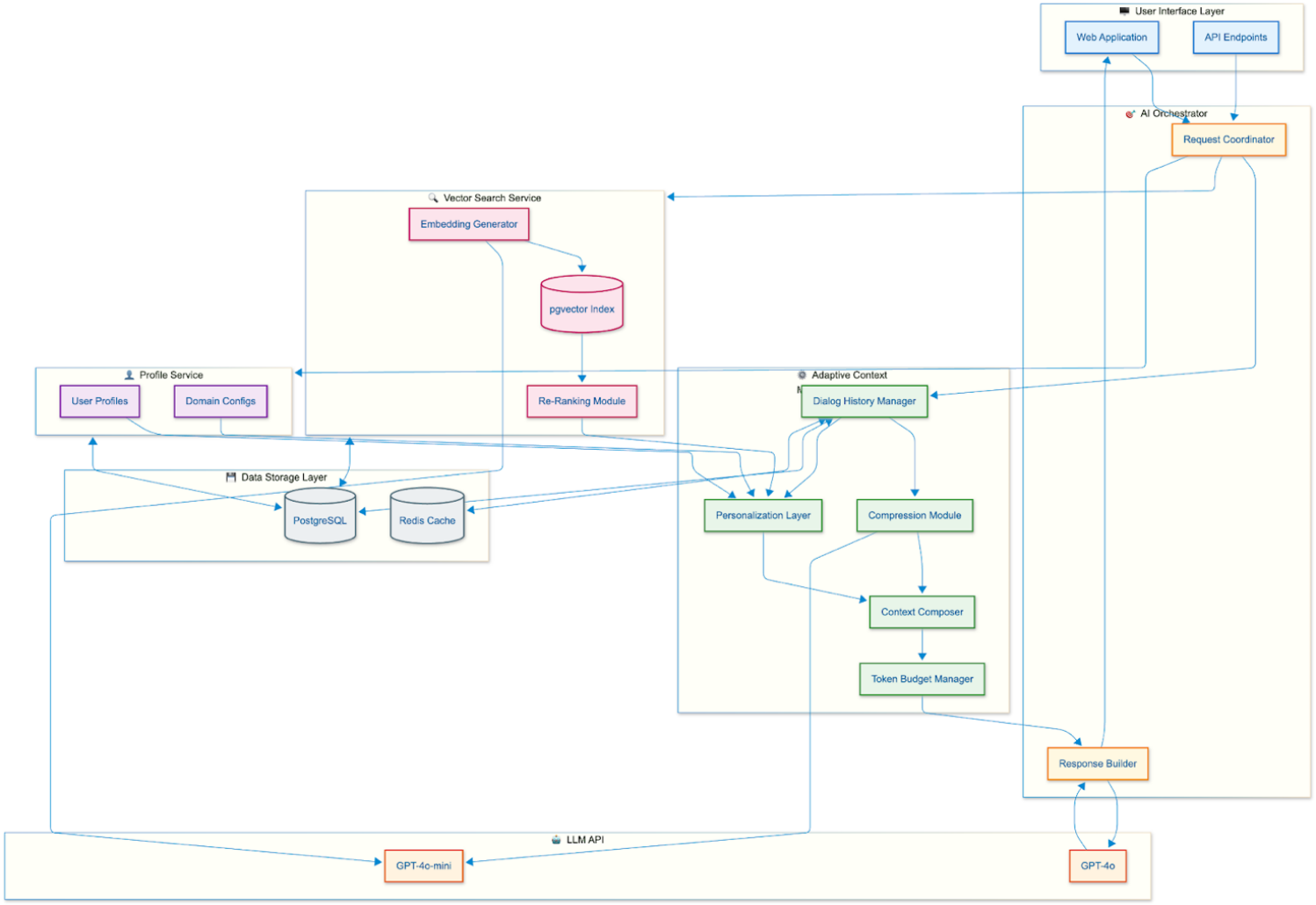

Рисунок Г.1 – Загальна архітектура системи ACMS

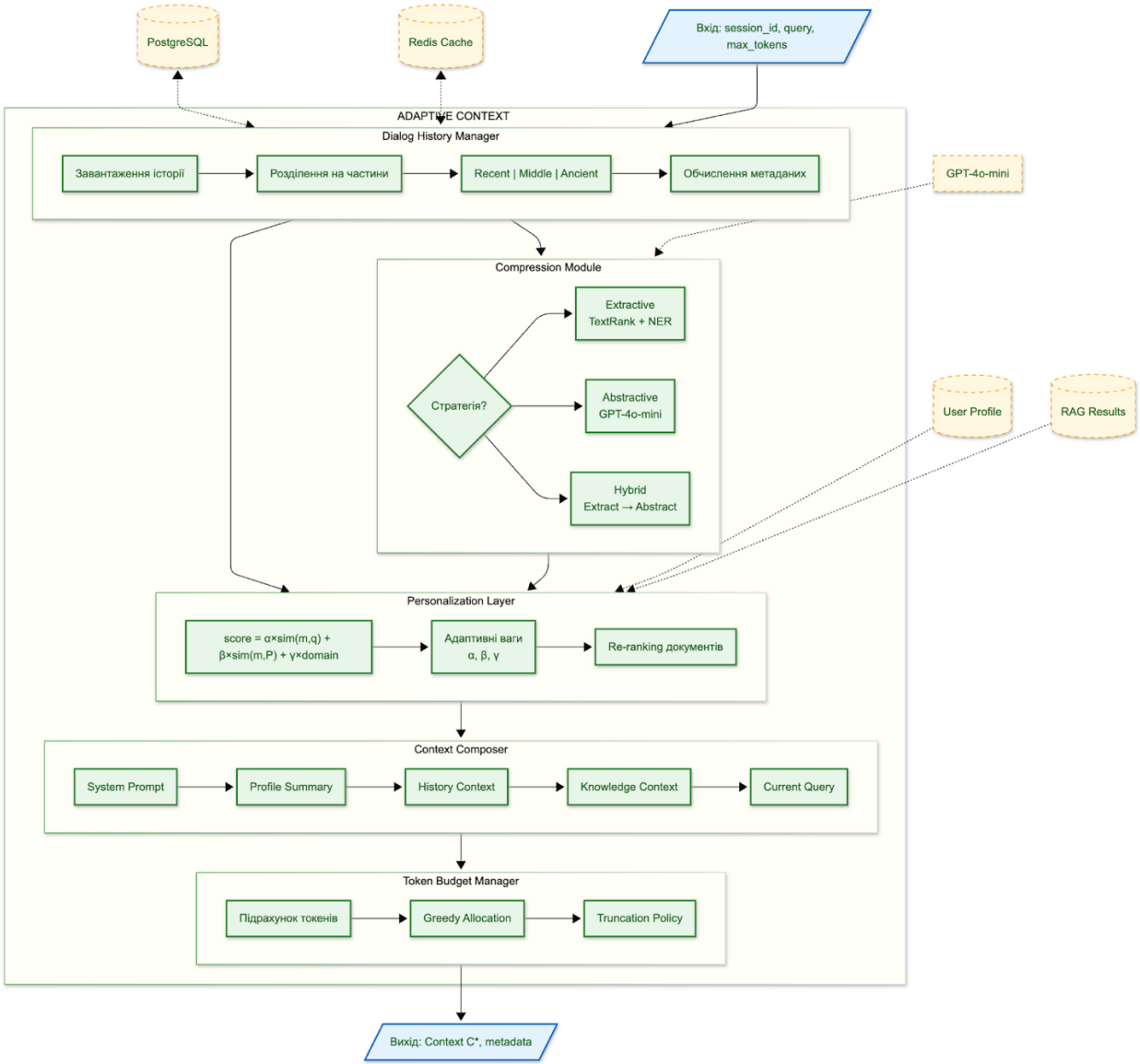

Рисунок Г.2 – Модульна структура системи ACMS

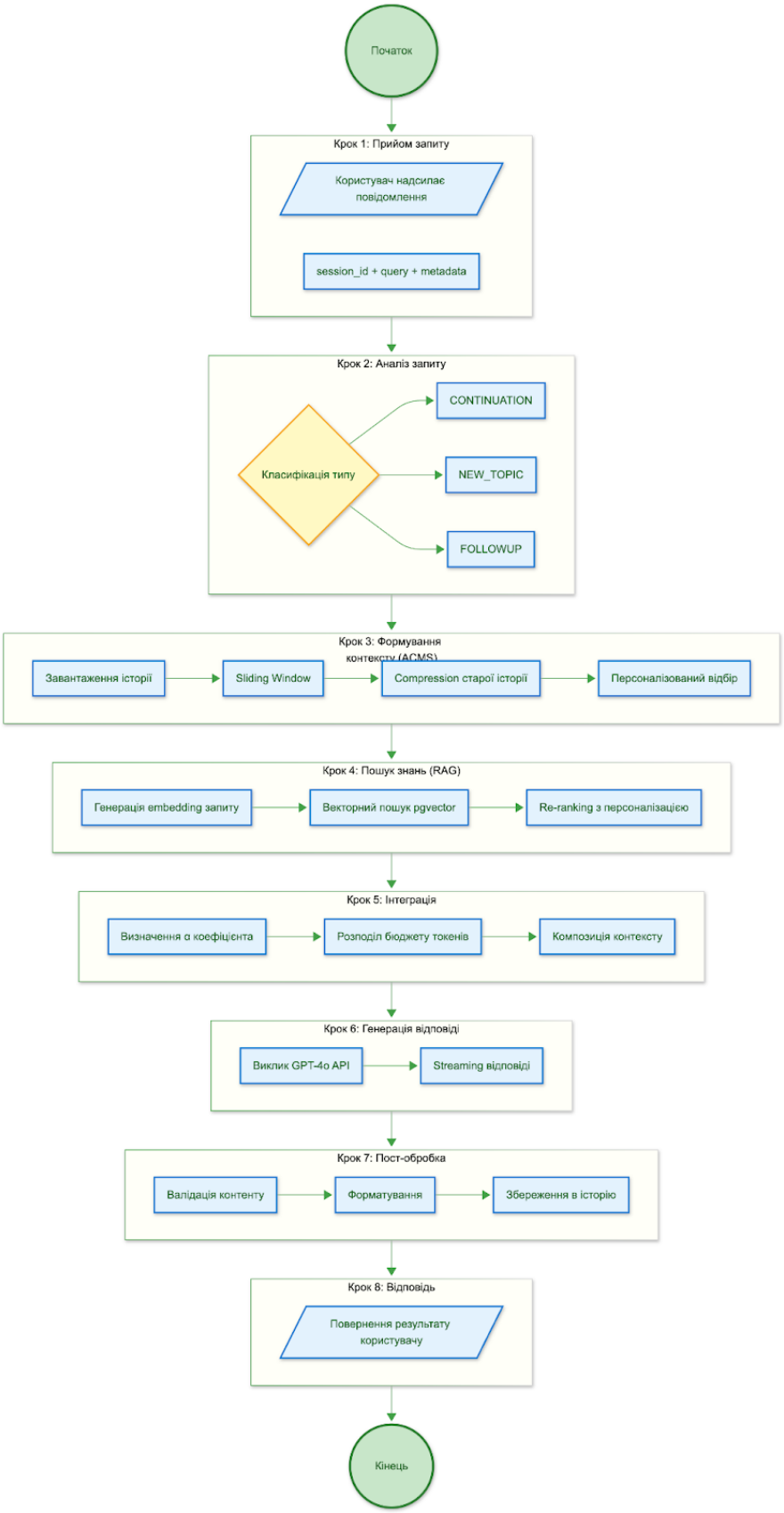

Рисунок Г.3 – Послідовність обробки користувацького запиту

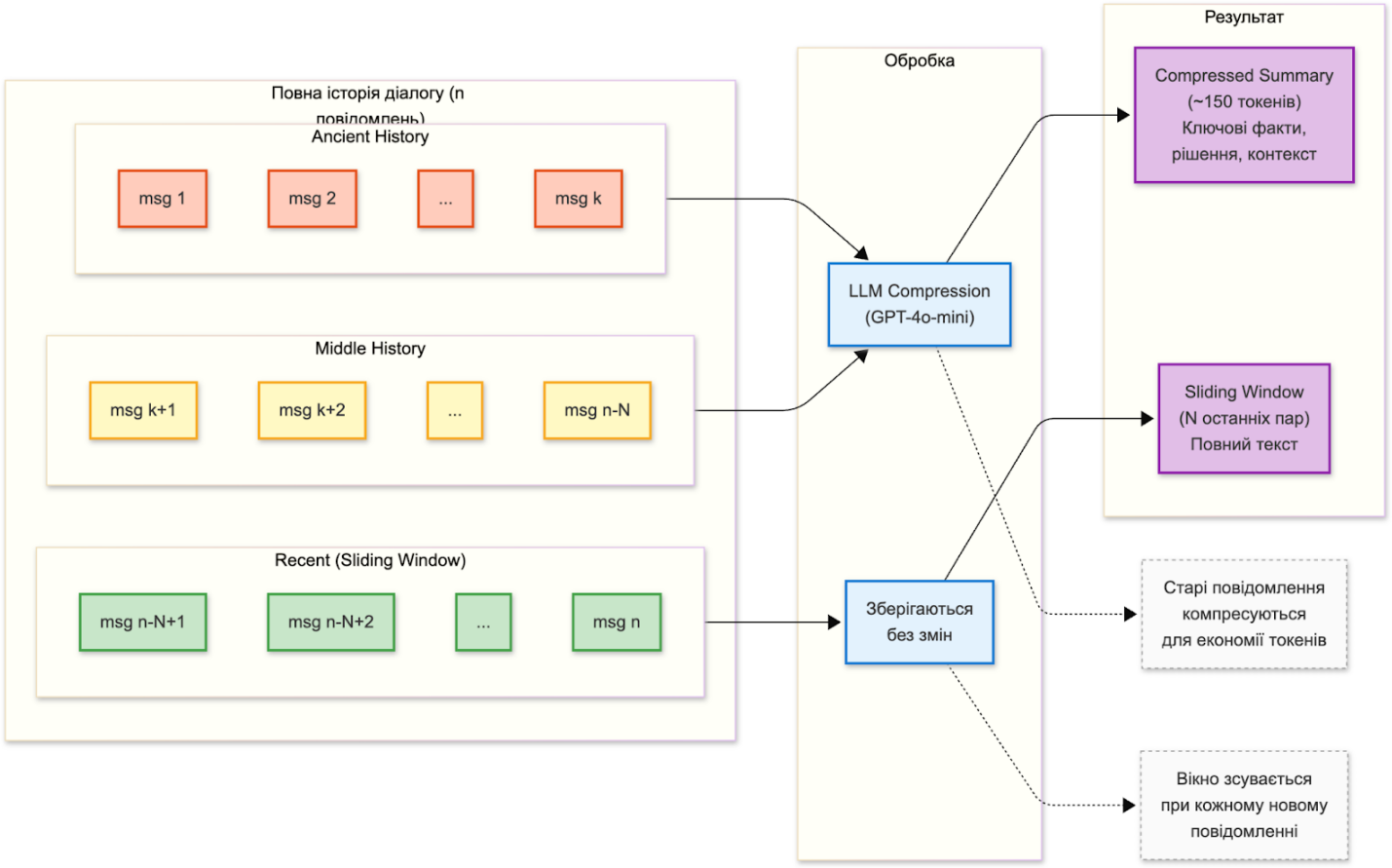

Рисунок Г.4 – Механізм ковзного вікна та компресії історії

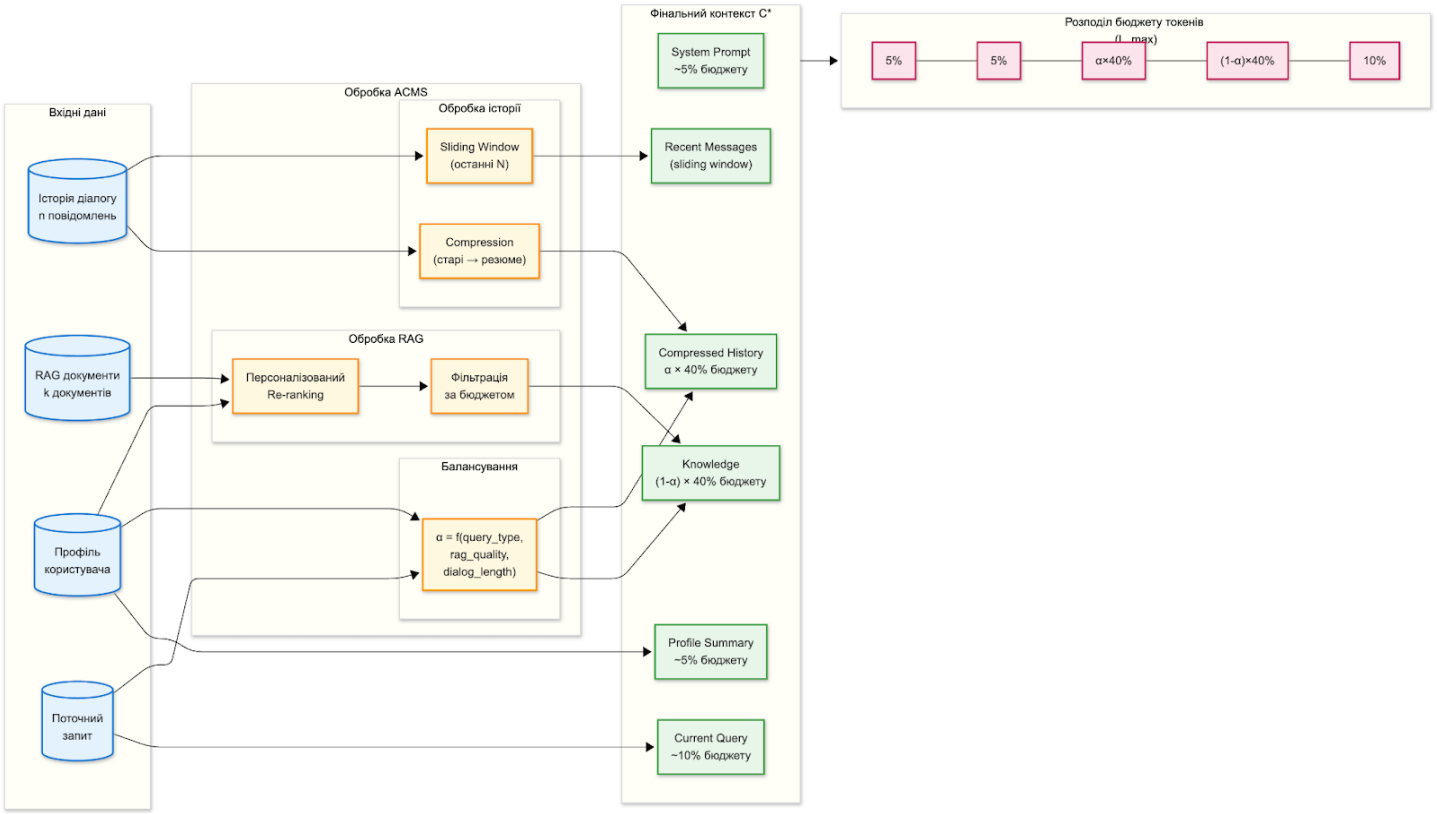

Рисунок Г.5 – Схема формування та композиції контексту

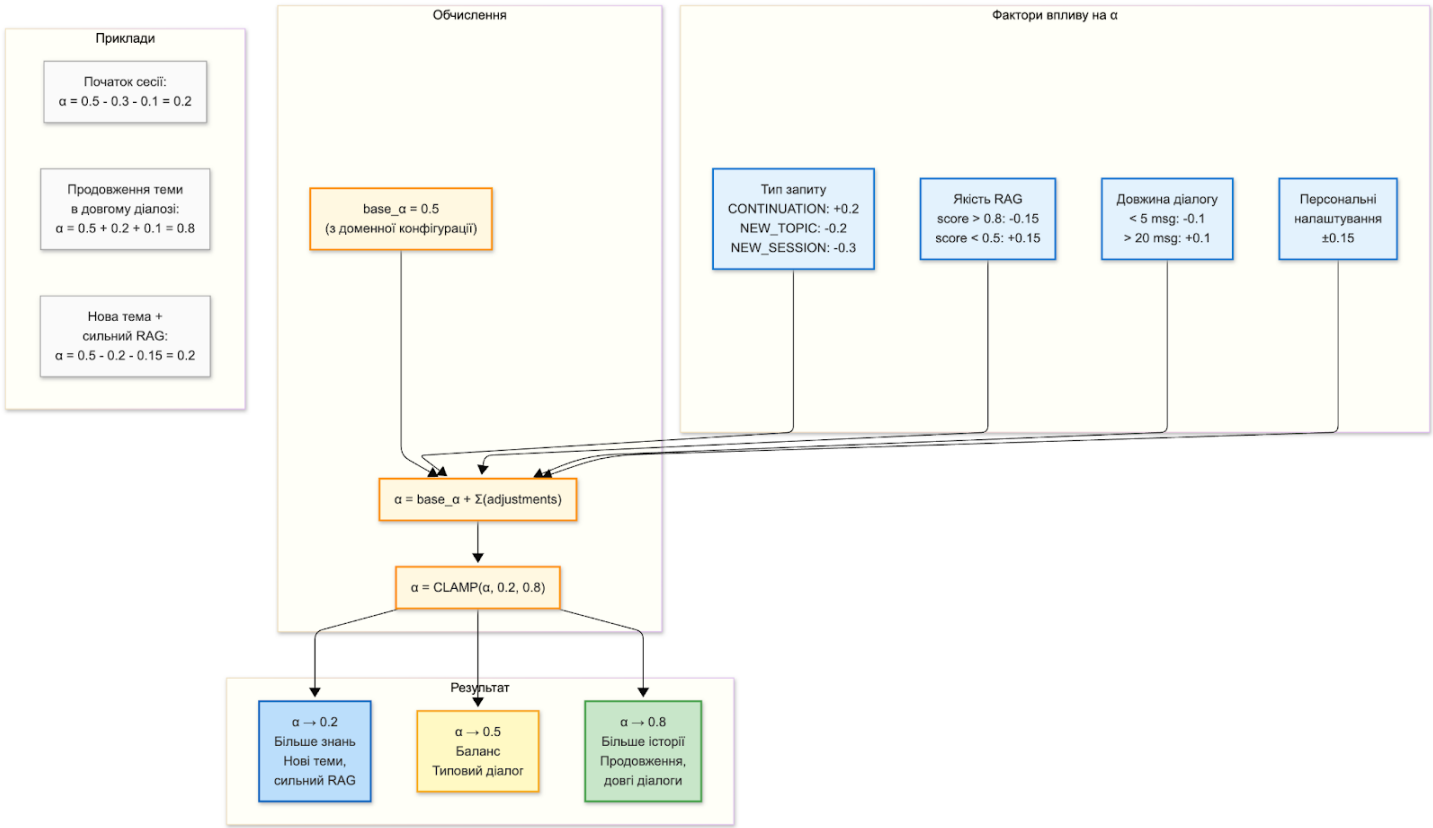

Рисунок Г.6 – Фактори впливу та обчислення коефіцієнта балансування α

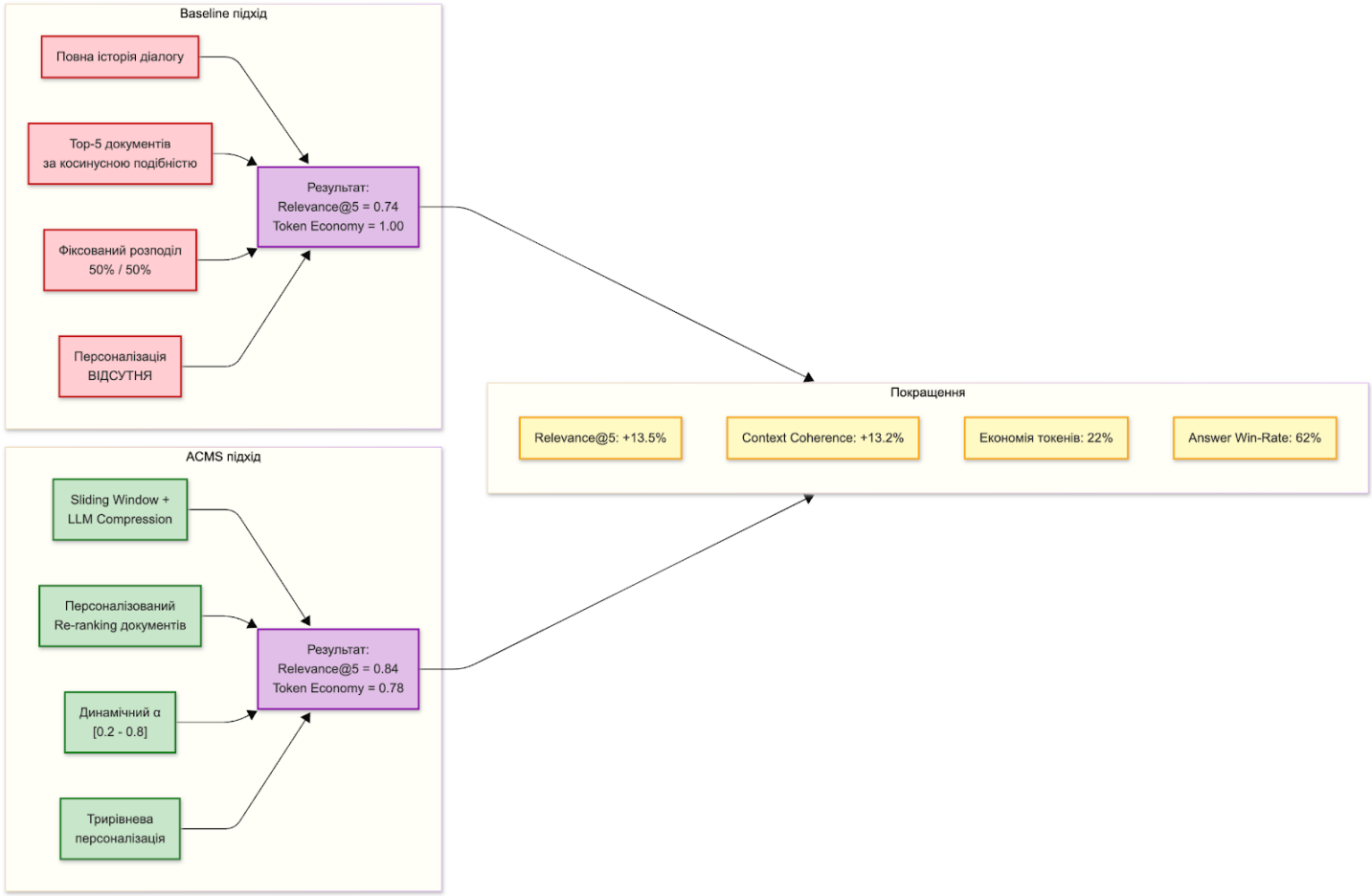

Рисунок Г.7 – Концептуальне порівняння Baseline та ACMS підходів

Додаток Д

Лістинг Д.1 – Структура фінального промпту з компонентами для мовної моделі

```
[SYSTEM]
You are a helpful AI assistant specialized in {domain}.
...

[USER PROFILE]
Domain expertise: {level}
Interests: {interests_summary}
...

[CONVERSATION SUMMARY]
{compressed_summary}

[RECENT MESSAGES]
User: {message_n-4}
Assistant: {response_n-4}
...

[KNOWLEDGE BASE]
Document 1: {doc_1_content}
Document 2: {doc_2_content}
...

[CURRENT QUERY]
User: {current_query}
```

Лістинг Д.2 – Алгоритм адаптивного управління контекстом

```
ALGORITHM AdaptiveContextManagement
INPUT:
    session_id: str
    current_query: str
    max_context_tokens: int
    target_model: str

OUTPUT:
    context: Context
    metadata: Dict


BEGIN
// 1. Завантаження даних
    history ← LoadDialogHistory(session_id)
    user_profile ← LoadUserProfile(session_id)
    domain_config ← LoadDomainConfig(user_profile.domain)

    // 2. Визначення доступного бюджету токенів
    system_prompt ← GetSystemPrompt(domain_config)
    profile_summary ← GenerateProfileSummary(user_profile)

    system_tokens ← CountTokens(system_prompt, target_model)
    query_tokens ← CountTokens(current_query, target_model)
    profile_tokens ← CountTokens(profile_summary, target_model)
    reserved_buffer ← 100

    available_budget ← max_context_tokens - system_tokens
                     - query_tokens - profile_tokens
                     - reserved_buffer

    IF available_budget < 500 THEN
        RETURN FallbackContext(current_query, system_prompt)
    END IF

    // 3. Розділення історії на частини
```

```
    recent_messages, middle_history, ancient_messages ←
        SplitHistory(history, sliding_window_size=10)

    // 4. Обробка compressed summary
    IF EXISTS compressed_summary FOR ancient_messages THEN
        compressed_summary ← LoadFromCache(session_id)
    ELSE
        compressed_summary ← CompressHistory(ancient_messages +
middle_history[:5])
        CacheCompressedSummary(session_id, compressed_summary)
    END IF

    // 5. Персоналізований відбір повідомлень
    scored_recent ← []
    FOR message IN recent_messages DO
        score ← PersonalizationScore(message, current_query, user_profile)
        scored_recent.APPEND((message, score))
    END FOR

    scored_recent ← SortByScore(scored_recent, descending=True)

    // 6. Визначення балансу history vs knowledge
    query_type ← ClassifyQuery(current_query, history)
    base_alpha ← domain_config.default_alpha


IF query_type == "CONTINUATION" THEN
        alpha ← base_alpha + 0.2
    ELSE IF query_type == "NEW_TOPIC" THEN
        alpha ← base_alpha - 0.2
    ELSE
        alpha ← base_alpha
    END IF

    alpha ← CLAMP(alpha, 0.3, 0.7)

    // 7. Розподіл бюджету
    history_budget ← available_budget * alpha
    knowledge_budget ← available_budget * (1 - alpha)

    // 8. Формування history частини
    history_context ← []
```

```
used_tokens ← 0

IF compressed_summary IS NOT None THEN
   summary_tokens ← CountTokens(compressed_summary, target_model)
   IF summary_tokens <= history_budget * 0.3 THEN
      history_context.APPEND(compressed_summary)
      used_tokens += summary_tokens
   END IF
END IF

FOR (message, score) IN scored_recent DO
   message_tokens ← CountTokens(message, target_model)
   IF used_tokens + message_tokens <= history_budget THEN
      history_context.APPEND(message)
      used_tokens += message_tokens
   ELSE
      BREAK
   END IF
END FOR

// 9. Отримання та фільтрація знань
IF QueryNeedsKnowledge(current_query, query_type) THEN
   raw_documents ← VectorSearchService.Search(
      query=current_query,
      context=history_context,
      top_k=10
   )

   scored_documents ← []
   FOR doc IN raw_documents DO
      personalized_score ← PersonalizeDocumentScore(
         doc, user_profile, domain_config
      )
      scored_documents.APPEND((doc, personalized_score))
   END FOR

   scored_documents ← SortByScore(scored_documents, descending=True)

   knowledge_context ← []
   used_knowledge_tokens ← 0

   FOR (doc, score) IN scored_documents DO
```

```
        doc_tokens ← CountTokens(doc.content, target_model)

        IF used_knowledge_tokens + doc_tokens <= knowledge_budget THEN
            knowledge_context.APPEND(doc)
            used_knowledge_tokens += doc_tokens
        ELSE IF knowledge_budget - used_knowledge_tokens > 200 THEN
            available_space ← knowledge_budget - used_knowledge_tokens
            truncated_doc ← TruncateDocument(doc, available_space)
            knowledge_context.APPEND(truncated_doc)
            BREAK
        ELSE
            BREAK
        END IF
    END FOR
ELSE
    knowledge_context ← []
END IF

// 10. Композиція фінального контексту
final_context ← ComposeContext(
    system_prompt,
    profile_summary,
    history_context,
    knowledge_context,
    current_query
)

// 11. Перевірка розміру
final_tokens ← CountTokens(final_context, target_model)

IF final_tokens > max_context_tokens THEN
    final_context ← EmergencyTruncate(final_context, max_context_tokens)
    WARN("Context exceeded limit after composition")
END IF

// 12. Формування метаданих
metadata ← {
    "total_tokens": final_tokens,
    "system_tokens": system_tokens,
    "profile_tokens": profile_tokens,
    "history_tokens": used_tokens,
    "knowledge_tokens": used_knowledge_tokens,
```

```
        "alpha_coefficient": alpha,
        "query_type": query_type,
        "num_documents": LENGTH(knowledge_context),
        "num_history_messages": LENGTH(history_context),
        "compression_applied": compressed_summary IS NOT None
    }

    RETURN (final_context, metadata)
END
```

Лістинг Д.3 – Функція адаптації вагових коефіцієнтів

```
def AdaptWeights(query_type, user_profile):
    # Базові ваги
    alpha, beta, gamma, delta = 0.4, 0.3, 0.2, 0.1

    # Адаптація для cold start
    if user_profile.interaction_count < 10:
        beta *= 0.5  # зменшуємо довіру до профілю
        alpha += 0.15  # збільшуємо важливість запиту
        gamma += 0.15  # збільшуємо важливість домену

    # Адаптація для типу запиту
    if query_type == "NEW_TOPIC":
        gamma += 0.15
        beta -= 0.15
    elif query_type == "CONTINUATION":
        beta += 0.15
        gamma -= 0.15

    # Нормалізація до суми 1.0
    total = alpha + beta + gamma + delta
    return alpha/total, beta/total, gamma/total, delta/total
```

Лістинг Д.4 – Алгоритм вилучаючої компресії історії діалогу

```
def AdaptWeights(query_type, user_profile):
    # Базові ваги
    alpha, beta, gamma, delta = 0.4, 0.3, 0.2, 0.1

    # Адаптація для cold start
    if user_profile.interaction_count < 10:
        beta *= 0.5  # зменшуємо довіру до профілю
        alpha += 0.15  # збільшуємо важливість запиту
        gamma += 0.15  # збільшуємо важливість домену

    # Адаптація для типу запиту
    if query_type == "NEW_TOPIC":
        gamma += 0.15
        beta -= 0.15
    elif query_type == "CONTINUATION":
        beta += 0.15
        gamma -= 0.15

    # Нормалізація до суми 1.0
    total = alpha + beta + gamma + delta
    return alpha/total, beta/total, gamma/total, delta/total
```