КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра технологій управління

Спеціальність 122 – Комп'ютерні науки,
освітня програма «Інформаційна аналітика та впливи»

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему:

«Розробка інтелектуальної системи аналізу європейської корпоративної фінансової звітності на основі технології RAG»

Студента 2-го курсу групи ІАВ-21
Юрковського Нікіти Олександровича
(прізвище, ім'я, по батькові)

_____
(підпис студента)

Науковий керівник:
кандидат економічних наук, доцент
(науковий ступінь, вчене звання)
Мірошниченко Ігор Вікторович
(прізвище, ім'я, по батькові)

_____ _____
(дата) *(підпис)*

Попередній захист:

_____

(Висновок: «До захисту в Екзаменаційній комісії»)

Завідувач кафедри
технологій управління _____ Віктор МОРОЗОВ _____
(підпис) *(прізвище, ініціали)* *(дата)*

Київ – 2026

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра технологій управління

Освітньо-кваліфікаційний рівень Магістр

Спеціальність 122 – Комп'ютерні науки

Освітня програма Інформаційна аналітика та впливи

ЗАТВЕРДЖУЮ

Завідувач кафедри

професор Віктор МОРОЗОВ

_____

«___» _____ 20__ року

ЗАВДАННЯ НА ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Студент Юрковський Нікіта Олександрович

Група ІАВ-21

1. Тема кваліфікаційної роботи

«Розробка інтелектуальної системи аналізу європейської корпоративної фінансової звітності на основі технології RAG»

Затверджена наказом по університету від «____» _____ 2026 р. № _____.

2. Строк подання студентом готової роботи – «____» _____ 2026 р.

3. Цільова установка та вихідні дані до роботи

Цільова установка роботи полягає у розробці прототипу інтелектуальної системи аналізу європейської корпоративної фінансової звітності на основі технології Retrieval-Augmented Generation (RAG) та у проведенні порівняльного оцінювання різних конфігурацій. Основна мета – створити працюючий прототип, здатний обробляти мультимовний корпус фінансових звітів європейських

компаній та надавати аналітичні відповіді на запити користувачів. Реалізація системи передбачає вибір технологічного стеку (Python, LangChain, ChromaDB, sentence-transformers, Ollama), формування корпусу фінансових звітів з платформи financialreports.eu та інтеграцію як відкритих так і комерційних мовних моделей. Вихідні дані з сайту включають 4 528 Markdown-документів від 66 компаній із 14 країн ЄС за період 2019–2025 років.

4. Зміст роботи

Зміст роботи полягає у аналізі регуляторної інфраструктури європейської корпоративної звітності, огляд існуючих рішень європейскього, американскього та інших фінансових NLP, дослідження теоретичних основ RAG-систем, формування корпусу саме європейських фінансових звітів, реалізацію прототипу з трьома embedding-моделями та чотирма локальними і комерційними LLM, проведення порівняльного тестування 12 конфігурацій за критеріями якості, швидкості, повноти та вартості, а також розробку рекомендацій щодо впровадження системи та техніко-економічного обґрунтування різних варіантів розгортання для різних користувачів.

5. Перелік графічного матеріалу (слайдів)

Дана кваліфікаційна робота магістра налічує ____ таблиць та ____ рисунків. Перелік слайдів презентації та відповідного їм змісту: актуальність обраної теми (1 слайд), мета та завдання дослідження (1 слайд), об’єкт та предмет (1 слайд), наукова новизна та практичне значення (1 слайд), архітектура прототипу (1 слайд), технологічний стек (1 слайд), результати порівняльного тестування (3 слайди), мультимовне тестування (1 слайд), техніко-економічне обґрунтування (1 слайд), висновки (1 слайд).

6. Календарний план виконання роботи

№ з/п	Назва частин роботи	%	Виконання роботи	
			За планом	Фактично
1.	Вибір теми кваліфікаційної роботи	3	01.10.25	01.10.25
2.	Затвердження тем кваліфікаційних робіт та призначення наукових керівників	2	27.12.25	27.12.25
3.	Формування переліку матеріалів, літератури з проблематики кваліфікаційної роботи	10	08.01.26	08.01.26
4.	Складання розгорнутого плану кваліфікаційної роботи	5	19.01.26	19.01.26
5.	Ознайомлення наукового керівника з розгорнутим планом кваліфікаційної роботи. Внесення змін.	5	25.01.26	25.01.26
6.	Підготовка розділу 1 «Аналіз проблематики автоматизованого аналізу європейської корпоративної звітності»	10	12.02.26	12.02.26
7.	Підготовка розділу 2 «Теоретичні основи RAG-систем та обґрунтування технологічного стеку»	14	08.03.26	08.03.26
8.	Підготовка розділу 3 «Реалізація прототипу та експериментальне дослідження»	14	01.04.26	01.04.26

№ з/п	Назва частин роботи	%	Виконання роботи	
			За планом	Фактично
9.	Підготовка розділу 4 «Технологія застосування розробленої системи»	13	20.04.26	20.04.26
10.	Оформлення кваліфікаційної роботи. Підготовка висновків і пропозицій	15	05.05.26	05.05.26
11.	Передача кваліфікаційної роботи науковому керівникові	2	_____	_____
12.	Передача кваліфікаційної роботи рецензенту для рецензування	2	_____	_____
13.	Попередній захист кваліфікаційної роботи	5	_____	_____

Дата видачі завдання «___» _____ 2026 р.

Керівник роботи: кандидат економічних наук, доцент Ігор МІРОШНИЧЕНКО

(посада, ім'я, прізвище) _____

(підпис)

Завдання прийняв до виконання студент групи ІАВ-21:

Юрковський Нікіта Олександрович _____

(підпис)

ЗМІСТ

АНОТАЦІЯ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра технологій управління
Спеціальність 122 – Комп'ютерні науки,
освітня програма «Інформаційна аналітика та впливи»

Дипломна робота магістра Юрковського Нікіти Олександровича.

Тема роботи – «Розробка інтелектуальної системи аналізу європейської корпоративної фінансової звітності на основі технології RAG».

Мета дипломної роботи магістра – розробка прототипу інтелектуальної системи аналізу європейської корпоративної фінансової звітності на основі технології Retrieval-Augmented Generation (RAG) та проведення порівняльного оцінювання її різних конфігурацій.

Об'єкт дослідження – процеси автоматизованого аналізу корпоративної фінансової звітності європейських компаній.

Предмет дослідження – методи та технології побудови RAG-систем для інтелектуального аналізу фінансових документів.

Наукова новизна роботи полягає у тому, що вперше запропоновано та експериментально досліджено RAG-систему, спеціалізовану на корпусі європейської фінансової звітності у форматі ESEF/Markdown, із порівняльним аналізом впливу вибору embedding-моделі (multilingual-e5-large, all-mpnet-base-v2, bge-large-en-v1.5) та генеративної LLM (llama3.1:8b, mistral, Claude Sonnet, GPT-4o) на якість аналітичних відповідей по багатьом різних параметрів. Практична цінність роботи полягає у розробленому працюючому прототипі, який може бути адаптований для використання, наприклад: фінансовими аналітиками, інвестиційними компаніями, аудиторськими фірмами та регуляторними органами як інструмент зручного, точного інтелектуального пошуку та аналізу європейської корпоративної звітності.

У роботі досліджуються існуючі підходи до автоматизованого аналізу європейської корпоративної звітності та технології RAG, проводиться

порівняльний аналіз 12 конфігурацій системи (3 embedding-моделі × 4 LLM) на корпусі з 4 528 Markdown-документів від 66 компаній із 14 країн ЄС за період 2019–2025 років. Розробляються рекомендації щодо впровадження системи для п’яти категорій цільових користувачів та виконується техніко-економічне обґрунтування.

Дипломна робота складається зі вступу, основної частини, яка включає чотири розділи, висновків та списку використаних джерел. Всього налічує 100 сторінок, перелік посилань з 82 джерел на 5 сторінках.

Ключові слова: Retrieval-Augmented Generation, RAG, фінансова звітність, Європейський Союз, ESEF, embedding, великі мовні моделі, LLM, ChromaDB, мультимовний аналіз, NLP.

ПЕРЕЛІК ВИКОРИСТАНИХ СКОРОЧЕНЬ

ANN – Approximate Nearest Neighbor, алгоритм наближеного пошуку найближчих сусідів

API – Application Programming Interface, програмний інтерфейс застосунку

BM25 – Best Matching 25, алгоритм ранжування у пошукових системах

ESEF – European Single Electronic Format, єдиний європейський електронний формат звітності

ESAP – European Single Access Point, єдина європейська точка доступу

ESMA – European Securities and Markets Authority, Європейське управління з цінних паперів і ринків

ESG – Environmental, Social, and Governance, екологічні, соціальні та управлінські критерії

GAAP – Generally Accepted Accounting Principles, загальноприйняті принципи бухгалтерського обліку

GPU – Graphics Processing Unit, графічний процесор

HNSW – Hierarchical Navigable Small World, ієрархічна навігація у малих світах

IFRS – International Financial Reporting Standards, Міжнародні стандарти фінансової звітності

iXBRL – Inline eXtensible Business Reporting Language, вбудована мова ділової звітності

LLM – Large Language Model, велика мовна модель

MTEB – Massive Text Embedding Benchmark, масштабний бенчмарк текстових вкладень

NLP – Natural Language Processing, обробка природної мови

OAM – Officially Appointed Mechanism, офіційно призначений механізм зберігання регульованої інформації

RAG – Retrieval-Augmented Generation, генерація з доповненням пошуком

RAGAS – Retrieval Augmented Generation Assessment, фреймворк оцінки якості RAG-систем

SEC – Securities and Exchange Commission, Комісія з цінних паперів і бірж США

TCO – Total Cost of Ownership, загальна вартість володіння

VRAM – Video Random Access Memory, відеопам’ять графічного процесора

XBRL – eXtensible Business Reporting Language, розширювана мова ділової звітності

XHTML – eXtensible HyperText Markup Language, розширювана мова гіпертекстової розмітки

ВСТУП

Ринок капіталів Європейського Союзу обслуговує сотні і тисячі публічних компаній із 27 країн-членів, кожна з яких зобов'язана розкривати фінансову інформацію відповідно до Директиви про прозорість 2004/109/ЄС та Делегованого регламенту (ЄС) 2019/815, що встановив вимоги до єдиного електронного формату звітності (European Single Electronic Format, ESEF). За останні п'ять років активного впровадження ESEF, за допомогою стандартизації на основі тегування розширюваною мовою ділової звітності (XBRL) та створення національних механізмів зберігання (OAM) було сформовано масштабну інфраструктуру машиночитних фінансових документів. Прийняття Регламенту (ЄС) 2023/2859 про створення Єдиної європейської точки доступу (ESAP), яку Європейське управління з цінних паперів і ринків (ESMA) має, в теорії, запустити до 2027 року, підтверджує те, що ЄС обрали і поступово йдуть курсом орієнтованим на цифровізацію фінансової інформації.

Але, навіть зважаючи на ці не малі досягнення у цьому напрямку, автоматизований аналіз європейської корпоративної звітності залишається нерозв'язаною задачею на даний момент, особливо у порівнянні з американським ринком. Масштаб корпусу, десятки тисяч документів щорічно – робить ручний аналіз такого великого обсягу даних фізично неможливим. Мультимовність створює додаткові складнощі: фінансові звіти публікуються понад 24 офіційними мовами ЄС, а мультимовні моделі обробки природної мови (NLP) демонструють помітне зниження ефективності на мовах з обмеженим обсягом тренувальних даних. Структурна різнорідність між юрисдикціями та стандартами обліку (IFRS проти національних GAAP) ускладнює порівняльний аналіз. Перший досвід застосування ESEF виявив системні помилки в XBRL-тегуванні – від 2 до 44 помилок на компанію навіть серед великих емітентів, що ставить під сумнів надійність структурованих даних.

Існуючі інструменти фінансового NLP – спеціалізовані мовні моделі (BloombergGPT, FinGPT, TAT-LLM), бенчмарки (FinQA, FinanceBench, FinBen), RAG-системи (HybridRAG, MultiFinRAG) – практично повністю орієнтовані на документи Комісії з цінних паперів і бірж США (SEC) та англомовний контент. У випадку з європейським ринком, європейський аналітик, котрий, у свою чергу, працює з мультимовним корпусом звітів – залишається з менш зручним інструментарієм, або взагалі без нього, порівнянного з американським контекстом.

Технологія Retrieval-Augmented Generation (RAG), яка поєднує семантичний пошук у зовнішній базі знань із генеративними можливостями великих мовних моделей (LLM), є перспективним підходом до розв'язання цієї проблеми. RAG-система працює виключно з повним текстом фінансових документів, які подаються їй на вхід, не покладаючись на потенційно некоректне структуроване тегування. Проте застосування RAG до європейського контексту потребує дослідження впливу ключових компонентів – вибору embedding-моделі (мультимовної чи англомовної), генеративної LLM (відкритої чи комерційної) та їх вплив на якість аналітичних відповідей, а також оцінки здатності системи працювати з мультимовним корпусом.

Метою роботи є розробка прототипу інтелектуальної системи аналізу європейської корпоративної фінансової звітності на основі технології RAG та проведення порівняльного оцінювання її конфігурацій.

Відповідно до мети визначено такі завдання дослідження:

1. Проаналізувати проблематику автоматизованого аналізу європейської корпоративної звітності та провести огляд існуючих підходів і рішень.
2. Дослідити теоретичні основи та компоненти RAG-систем, виконати порівняльний аналіз стратегій поділу документів, моделей векторних представлень, генеративних мовних моделей і методик оцінювання якості.

3. Сформувати корпус європейських фінансових звітів на основі даних financialreports.eu та реалізувати прототип RAG-системи з кількома альтернативними конфігураціями.
4. Провести порівняльне тестування конфігурацій прототипу за критеріями якості відповідей, швидкості генерації та вартості.
5. Розробити рекомендації щодо впровадження системи та оцінити техніко-економічну доцільність варіантів розгортання.

Об’єкт дослідження – процеси автоматизованого аналізу корпоративної фінансової звітності європейських компаній.

Предмет дослідження – методи та технології побудови RAG-систем для інтелектуального аналізу фінансових документів.

У роботі використано аналіз і синтез при дослідженні регуляторної інфраструктури та існуючих підходів до фінансового NLP. Порівняльний аналіз при оцінюванні компонентів RAG-систем та конфігурацій прототипу. Моделювання при проектуванні архітектури та пайплайнів. Експерименти при тестуванні прототипу на реальному корпусі з фіксацією кількісних метрик (час відповіді, кількість токенів, вартість) та якісною оцінкою відповідей.

Наукова новизна одержаних результатів полягає у тому, що вперше запропоновано та експериментально досліджено підхід до побудови RAG-системи, спеціалізованої на корпусі європейської корпоративної звітності у форматі ESEF/Markdown, з порівняльним аналізом впливу вибору embedding-моделі (multilingual-e5-large, all-mpnet-base-v2, bge-large-en-v1.5) та генеративної LLM (llama3.1:8b, mistral, Claude Sonnet, GPT-4o) на якість аналітичних відповідей. На відміну від існуючих досліджень, орієнтованих на документи SEC та англомовний контент, ця робота – одна з перших адресує мультимовний європейський контекст і включає порівняльний аналіз відкритих та комерційних компонентів RAG-системи на даних з 14 країн ЄС та Європи.

Практичне значення одержаних результатів визначається тим, що розроблений прототип може бути адаптований для використання фінансовими

аналітиками, інвестиційними компаніями, аудиторськими фірмами та регуляторними органами як інструмент інтелектуального пошуку та аналізу європейської корпоративної звітності. Сформульовані рекомендації щодо оптимальних конфігурацій (вибір embedding-моделі та LLM) для різних сценаріїв мають практичну цінність, доведену при експериметнах, яка знадобиться при впровадженні подібних систем. Виявлені обмеження прототипу та визначені напрямки розвитку (гібридний пошук, re-ranking, інтеграція з ESAP) формують основу для подальшої інженерної роботи.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, переліку використаних інформаційних джерел та додатків. Перший розділ присвячений аналізу проблематики та постановці задачі. Другий розділ містить теоретичні основи RAG-систем та обґрунтування технологічного стеку. У третьому розділі описано реалізацію прототипу та результати експериментального дослідження. Четвертий розділ охоплює технологію застосування системи, якісний аналіз, техніко-економічне обґрунтування та перспективи розвитку.

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ АВТОМАТИЗОВАНОГО АНАЛІЗУ ЄВРОПЕЙСЬКОЇ КОРПОРАТИВНОЇ ЗВІТНОСТІ

1.1. Європейська система корпоративної фінансової звітності

Ринок капіталів ЄС обслуговує тисячі публічних компаній із 27 країн-членів, кожна з яких зобов'язана регулярно розкривати фінансову інформацію. Ця система побудована на низці нормативних актів, що сформували єдину інфраструктуру збору, зберігання та поширення корпоративної звітності. Розуміння цієї інфраструктури є передумовою проектування системи автоматизованого аналізу європейських фінансових документів.

Основу регуляторної рамки становить Директива 2004/109/ЄС від 15 грудня 2004 року – Директива про прозорість (Transparency Directive) [1]. Вона встановила вимоги до періодичного та поточного розкриття інформації емітентами цінних паперів на регульованих ринках ЄС: річні фінансові звіти – протягом чотирьох місяців після закінчення фінансового року (Стаття 4) [1]. Початкова редакція Статті 5 встановлювала термін публікації піврічних звітів у два місяці. Чинний термін у три місяці запроваджено поправкою Директиви 2013/50/EU, яка одночасно доручила ESMA розробити регуляторні стандарти для єдиного електронного формату звітності. Стаття 21 зобов'язує кожну країну-члена забезпечити функціонування принаймні одного офіційно призначеного механізму (Officially Appointed Mechanism, OAM) для централізованого зберігання регульованої інформації [1] – національного репозиторію, аналогічного системі EDGAR у США.

Делегований регламент (ЄС) 2019/815 від 17 грудня 2018 року конкретизував вимоги до єдиного електронного формату (European Single Electronic Format, ESEF) [2]. Регламент встановив, що повні річні фінансові звіти готуються у форматі XHTML, який не потребує спеціального ПЗ для відображення та є вільно доступним непропрієтарним форматом [2]. Консолідовані звіти за IFRS додатково розмічуються мовою XBRL –

машиночитаною мовою розмітки, що реалізується через Inline XBRL (iXBRL): один файл одночасно читається людиною у браузері та обробляється програмно.

Таксономія для XBRL-розмітки базується на IFRS Taxonomy із доповненнями ESMA. Стаття 4 Регламенту визначає два рівні обов'язковості розмітки [2]: первинні фінансові звіти (баланс, P&L, грошові потоки, зміни у капіталі) підлягають детальному тегуванню, де кожен елемент отримує тег із таксономії. Примітки – блочному тегуванню, де цілі розділи позначаються одним тегом (вимога чинна для періодів з 1 січня 2022 року). Якщо існуючі елементи таксономії не відповідають специфіці компанії, емітент створює розширення, які мають бути "заякорені" до найближчого базового елемента [2]. Початково ESEF мав стати обов'язковим з 1 січня 2020 року, але через COVID-19 термін перенесено на рік [5].

Sran, Tuijn та Vollon (2023) у дослідженні впливу OAM на ринки капіталу за 2001–2015 роки по 19 країнах ЄС підтвердили, що централізація фінансової інформації покращує ліквідність ринку у середньому на вісім відсоткових пунктів [3]. Ефект найсильніший для малих фірм із низьким медіа-покриттям та з високою часткою роздрібних інвесторів – тобто там, де витрати на обробку інформації найвищі. Дослідники також виявили механізм інформаційних перетоків: ліквідність компанії покращується завдяки кращому доступу до звітності компаній-конкурентів у тій самій галузі [3]. Підходи національних OAM різняться – від спеціалізованих державних платформ (Франція, Німеччина) до рішень на базі фондових бірж – і ця фрагментованість стала аргументом на користь подальшої інтеграції через ESAP.

Регламент (ЄС) 2023/2859 передбачає створення Єдиної європейської точки доступу (ESAP), яку ESMA зобов'язана запустити до 10 липня 2027 року [4]. Платформа забезпечить централізований електронний доступ до інформації, оприлюдненої відповідно до переліку законодавчих актів ЄС. На відміну від національних OAM, ESAP агрегує інформацію з усіх країн-членів в одному місці, забезпечуючи доступ через програмний інтерфейс (API) для машинної

обробки та зручний пошуковий інтерфейс [4]. Інформація має подаватися у форматах, придатних для машинної обробки та витягування даних – це принципово важливо для побудови інтелектуальних систем аналізу.

Перший досвід впровадження виявив значні проблеми з якістю розмітки. Kobiela-Pionnier та Karwowski (2024) проаналізували всі XBRL-теги у консолідованих звітах 50 польських компаній на Варшавській біржі за 2020 рік [5]. З 5 980 тегів базової таксономії та 510 розширень виявлено помилки у всіх без винятку компаніях. Кількість помилок коливалась від двох до 44. Серед основних типів – вибір тегу із занадто широким значенням, невиправдане створення розширень (коли підходящий тег уже існує) та помилки закріплення розширень. Розширення становили 9,1% від усіх тегів корпусу. Найбільше помилок зафіксовано у звітах про рух грошових коштів, де частка розширень сягала 14,9% від основних тегів цього розділу, з яких 67,3% містили помилки [5]. Аналогічні проблеми задокументовано в Італії та Фінляндії. Головний висновок авторів – основним джерелом помилок є недостатня обізнаність із таксономією ESEF, а не технічні обмеження. Це має пряме значення для проектування RAG-систем: якість структурованих даних не можна вважати гарантованою – текстові версії документів часто є надійнішим джерелом для семантичного аналізу.

Прямий доступ до ESEF-звітів через ОАМ різних країн ускладнений розрізненістю інтерфейсів та відсутністю єдиного API (до запуску ESAP). Одним із рішень є агрегатори, серед яких – платформа financialreports.eu [6]. Її ключовою перевагою є наявність полегшених текстових версій документів у форматі Markdown, оптимізованих для обробки мовними моделями. Типи доступних документів наведено у таблиці 1.1.

Таблиця 1.1 – Типи документів на платформі financialreports.eu

Тип документа	**Опис**	**Частота**
Annual Report (ESEF)	Річний звіт у форматі ESEF (XHTML з iXBRL-розміткою)	Щорічно
Annual Report	Повні річні звіти, включаючи звіт керівництва та примітки	Щорічно
Annual/Quarterly Financial Statement	Окрема фінансова звітність (баланс, P&L)	Щоквартально/щорічно
Earnings Release	Прес-релізи з ключовими фінансовими результатами	Щоквартально
Business and Financial Review	Огляд бізнес-діяльності та фінансових показників	Щоквартально/щорічно
Quarterly Report	Квартальні звіти з деталізованою фінансовою інформацією	Щоквартально
Interim/Quarterly Report	Проміжні та піврічні звіти за Директивою про прозорість	Щоквартально/щопівроку
Environmental & Social Information	Звітність ESG	Щорічно
Management Reports	Звіти керівництва з аналізом діяльності та ризиків	Щорічно

Доступ до повних можливостей з використання API на цій платформи потребує окремої домовленості з розробниками сайту, тому у роботі дані отримувались ручним завантаженням ZIP-архівів із веб-інтерфейсу, доступ до яких вдалося отримати за допомогою місячнох підписки. Процес збору, характеристики корпусу та його підготовку детально описано у розділі 3.1.

Європейська система корпоративної звітності – це багаторівнева інфраструктура, що еволюціонує від все ще розрізнених на даний момент національних вимог, до стандартизованого, машиночитаного формату з єдиною точкою доступу. Директива про прозорість [1] створила регуляторну основу, ESEF [2] забезпечив технічну стандартизацію, OAM сформували національні точки зберігання з доведеним позитивним впливом на ліквідність [3], ESAP [4] стане загальноєвропейським агрегатором з відкритим API. Проте проблеми

якості тегування [5] та фрагментованість доступу залишаються актуальними викликами, що обґрунтовує потребу в інтелектуальних системах аналізу текстових версій звітів.

1.2. Ключові проблеми автоматизованого аналізу європейської звітності

Регуляторна інфраструктура створює масштабний неоднорідний масив фінансових даних, автоматизована обробка якого пов'язана з низкою специфічних проблем. Їх можна структурувати за чотирма напрямами: масштаб даних, мультимовність, різнорідність форматів і обмеженість існуючих інструментів.

Платформа financialreports.eu агрегує документи від десятків тисяч європейських емітентів. Навіть обмежившись річними фінансовими звітами, обсяг вимірюється тисячами документів щорічно по 50 – 300 сторінок кожен. Часті оновлення версій (зі змінами на кілька відсотків тексту) пропорційно збільшують обсяг бази. Аналітик суто фізично не спроможний прочитати навіть невелику частину цього масиву. Традиційний підхід вимагає або розподілу роботи між великою кількістю спеціалістів, або свідомого обмеження охоплення вузьким переліком компаній – що, у свою чергу, створює потребу в інтелектуальних інструментах пошуку та аналізу.

На відміну від американської звітності, що подається лише англійською, європейські звіти публікуються офіційними мовами країн-членів – потенційно понад 24 мовами. Більшість великих компаній публікують звіти рідною мовою та англійською паралельно, але менші емітенти часто обмежуються лише національною мовою, що є досить великою проблемою, так як малих/середніх компаній значно більше. Датасет MultiFin [7] містить фінансові заголовки 15 мовами з п'яти мовних сімей. Результати оцінки показали, що мультимовні моделі (mBERT, XLM-R, mT5) при достатній якості на високоресурсних мовах демонструють помітне зниження ефективності для мов з обмеженим обсягом тренувальних даних [7]. Змагання FNS 2023 [8] із сумаризації фінансових

наративів засвідчило: найкращі системи досягали ROUGE-2 на рівні 0,32 для текстів англійською проти 0,12 – 0,14 для грецьких та іспанських [8]. Це означає, що перенесення NLP-методів з англійської на інші європейські мови у фінансовому полі супроводжується значною втратою якості. Мультимовність також впливає на векторизацію текстів: моделі embedding, навчені переважно на англомовних корпусах, неадекватно представляють семантику фінансових текстів іншими мовами, що безпосередньо впливає на якість пошуку.

Незважаючи на стандартизацію через ESEF, структура звітів помітно відрізняється між компаніями, галузями та юрисдикціями. Банківський звіт з Німеччини за наповненням докорінно відрізняється від звіту технологічної компанії з Естонії або промислового холдингу з Італії. Проблему ускладнює співіснування IFRS з національними GAAP країн, які окремі компанії все ще продовжують використовувати, так як перехід до єдиної системи ще не завершено. Дослідження Kobiela-Pionnier та Karwowski [5] на 50 польських компаніях за 2020 рік виявило, що BCI емітенти так чи інакше допустили помилки у тегуванні: 211 помилок у тегах основної таксономії (3,8%) і 354 у розширеннях (69,4% від усіх створених). Найпоширеніша помилка – вибір занадто загального тегу (45,2% помилок у розширеннях), а 19,8% розширень виявились непотрібними [5]. Якщо структуровані дані містять систематичні помилки навіть на рівні окремої країни, покладатися виключно на тегування для крос-юрисдикційного аналізу неможливо – це обґрунтовує підхід на основі аналізу саме неструктурованого тексту.

Більшість комерційних та дослідницьких рішень орієнтована на документи SEC (форми 10-K, 10-Q, 8-K) та англійську мову. Бенчмарки FinQA, TAT-QA, ConvFinQA, FinanceBench (детальніше буде у розділі 1.3) побудовані на американських документах. Комерційні платформи Bloomberg Terminal, FactSet, Refinitiv первинно орієнтовані на ринки США. Аналогічна ситуація у фінансових мовних моделях: BloombergGPT, FinGPT та інші тестуються на англомовних бенчмарках. Поодинокі європейські адаптації – як German FinBERT

– фрагментарні та не покривають навіть основних/найрозповсюдженіших мов ЄС. Європейський аналітик позбавлений усього цього нового інструментарію, у порівнянні з американським контекстом.

Узагальнення проблематики наведено у таблиці 1.2.

Таблиця 1.2 – Ключові проблеми автоматизованого аналізу європейської корпоративної звітності

Проблема	**Характеристика**	**Вплив на автоматизований аналіз**
Масштаб даних	Десятки тисяч емітентів, тисячі документів щорічно по 50–300 сторінок	Неможливість ручного аналізу та потреба в інтелектуальних системах
Мультимовність	24+ мови ЄС, різні системи письма, нерівномірна NLP-підтримка [7]	ROUGE-2 для грецької/іспанської у 2–3 рази нижчий, ніж для англійської [8]
Різнорідність форматів	Різна структура між країнами, IFRS vs національні GAAP, 69,4% розширень XBRL з помилками [5]	Неможливість покладатися на тегування, потреба в аналізі тексту
Обмеженість інструментів	Бенчмарки, LLM, платформи орієнтовані на SEC та англійську	Відсутність готових рішень для європейського ринку

Сукупність цих проблем напряму формує запит на інтелектуальну систему, здатну працювати з великими обсягами мультимовних фінансових документів різної структури. Технологія RAG є перспективним підходом, оскільки поєднує пошук по корпусу з генеративними можливостями LLM.

1.3. Огляд існуючих підходів та рішень

1.3.1. Спеціалізовані фінансові мовні моделі

Загальні LLM не завжди ефективні з фінансовою термінологією та числовими даними, що спонукало створення спеціалізованих FinLLMs. Огляд Lee et al. [13] виділяє три стратегії побудови таких моделей: zero-shot/few-shot prompting (без додаткового навчання, лише вибір прикладів у запиті), fine-tuning на доменних даних (дотренування заздалегідь навченої LLM на фінансових текстах) та pre-training з нуля на спеціалізованому корпусі (найдорожчий, але найгнучкіший варіант). Nie et al. [14] каталогізують понад 40 спеціалізованих

фінансових LLM, що з'явилися протягом 2022–2024 років. Однак Lopez-Lira et al. [74] у свіжому огляді 2025 року фіксують значний розрив між швидким розвитком LLM-технологій та повільним впровадженням у фінансовій індустрії: реальні системи вимагають обережної валідації, аудиту відповідей та regulatory compliance – ці фактори стримують перенесення академічних результатів у виробничі рішення.

BloombergGPT [9] став першою масштабною фінансовою LLM, що отримала широкий резонанс – пропрієтарна модель з 50 мільярдами параметрів, навчена з нуля на унікальному корпусі: 363 мільярди токенів фінансових текстів (внутрішні дані Bloomberg за десятиліття) та 345 мільярдів токенів із загального корпусу для збереження загальних мовних здібностей. На внутрішніх фінансових бенчмарках модель перевершила базові LLM того самого розміру, водночас зберігши конкурентну ефективність на загальних NLP-бенчмарках. Однак закритий характер моделі – відсутність публічного доступу до ваг та значна вартість навчання – обмежили її використання академічною спільнотою, що стало стимулом до розвитку відкритих, open-source альтернатив [9].

Цією відповіддю на закритість BloombergGPT став **FinGPT** [10] – повністю відкритий проєкт, що ілюструє альтернативну стратегію: замість дорогого pre-training з нуля використовується LoRA-адаптація – техніка дотренування лише невеликих низькорангових матриць-адаптерів (зазвичай 0,1–1% від загальної кількості параметрів моделі), що істотно знижує обчислювальні витрати без особливо помітної втрати якості. Орієнтовна вартість одного циклу адаптації – близько 300 доларів США, на два-три порядки менше за повне дотренування великої моделі. На задачі класифікації фінансової тональності FinGPT досягає Macro-F1 80,9% проти 54,4% базової LLaMA 3 без адаптації [10] – наочна демонстрація ефективності доменного дотренування навіть малими ресурсами.

InvestLM [11] використовує іншу стратегію: instruction tuning моделі LLaMA-65B на колекції інструкцій, отриманих з матеріалів іспитів CFA (Chartered Financial Analyst – триступенева міжнародна сертифікація фінансових

аналітиків, що охоплює economics, accounting, fixed income, equities, derivatives, alternative investments, ethics). CFA-екзамени є валідним бенчмарком саме тому, що вимагають інтеграції фактологічних знань, кількісного міркування та інтерпретації фінансової звітності – спектр вимог, близький до реальної аналітичної роботи. **DISC-FinLLM** [12] адаптує модель Baichuan (13B) на китайському фінансовому корпусі через MEFF (Multi-Expert Fine-tuning Framework) – навчання чотирьох незалежних LoRA-адаптерів для різних типів задач (фінансові консультації, аналіз ризиків, числове міркування, обчислення на основі знань) із подальшим маршрутизатором запитів між ними.

Найпереконливіший приклад того, що архітектурні рішення можуть переважати простий розмір моделі, – **TAT-LLM** [15]. Замість монолітної LLM автори побудували триетапний пайплайн: Extractor (виокремлює релевантні числові та текстові елементи з гібридних таблично-текстових даних), Reasoner (формулює послідовність кроків міркування у вигляді псевдокоду), Executor (виконує арифметичні операції детермінованим інтерпретатором – це гарантує відсутність помилок у простих обчисленнях, що є слабким місцем LLM). Навіть 7-мільярдна версія TAT-LLM перевершила GPT-4 (на момент публікації – найпотужнішу комерційну модель) на трьох суміжних бенчмарках: FinQA 64,60% vs 63,91%, TAT-QA 74,56% vs 71,92%, TAT-DQA 69,45% vs 64,46% [15]. Цей результат демонструє: для фінансового домену модульна декомпозиція з явною формалізацією reasoning-кроків може бути ефективнішою за просте збільшення розміру моделі.

Таблиця 1.3 – Порівняння спеціалізованих фінансових мовних моделей

Модель	**Рік**	**Параметри**	**Підхід**	**Доступ**	**Особливість**
BloombergGPT [9]	2023	50B	Навчання з нуля (708B токенів)	Закритий	Перша масштабна фінансова LLM
FinGPT [10]	2023	LoRA	LoRA-адаптація	Відкритий	$300 за цикл адаптації

Продовження на наступній сторінці

Продовження таблиці 1.3

Модель	**Рік**	**Параметри**	**Підхід**	**Доступ**	**Особливість**
InvestLM [11]	2023	65B (LLaMA)	Instruction tuning	Відкритий	Порівнянний з GPT-4
DISC-FinLLM [12]	2023	13B (Baichuan)	MEFF (4 LoRA)	Відкритий	Китайський ринок
TAT-LLM [15]	2024	7B–70B (LLaMA 2)	Step-wise Pipeline	Відкритий	Перевершує GPT-4 на FinQA/TAT-QA

Жодна з моделей взагалі не орієнтована на європейську корпоративну звітність, це навіть не планувалось – усі на американському або китайському ринках, що підтверджує актуальність дослідження.

1.3.2. Бенчмарки та датасети

Дослідники створили низку спеціалізованих датасетів для оцінки фінансового NLP – від простого вилучення числових значень до багатокрокового міркування з гібридними таблично-текстовими даними. Розглянемо ключові бенчмарки за хронологією появи та задачами, які вони адресують.

Ранні бенчмарки (2021–2022) заклали основу для оцінювання числового міркування. **FinQA** [16] – найвпливовіший датасет цього періоду: 8 281 пара “питання-відповідь”, вибраних експертами зі звітів 500 найбільших публічних компаній США (S&P 500). Особливість датасету – вимога числового міркування: відповіді часто потребують арифметичних операцій над таблицями (відсотки, різниці, відношення). Експертний бенчмарк (фінансові аналітики-люди) показав 91,16% точності, тоді як спеціалізована модель FinQANet, навчена авторами датасету, досягла лише 61,24% – розрив у понад 30 п.п. демонструє, що навіть простіші фінансові обчислення складні для моделей без явного reasoning-механізму [16]. **TAT-QA** [17] (16 552 гібридних питань) розширює FinQA до одночасної роботи з таблицями та довколишнім текстом: експерт 90,8% F1 проти 58,0% у моделі. **ConvFinQA** [18] (3 892 діалоги) додає діалоговий вимір –

наступне питання може посилатися на попереднє через локальний контекст: GPT-3 175B досягав менш ніж 50% точності, експерт – 89,4%.

Великі open-book бенчмарки (2023–2024). FinanceBench [19] – open-book QA з 10 231 питанням до реальних 10-K і 10-Q звітів: GPT-4-Turbo з повноцінним retrieval помилявся на 81% питань – приклад масштабної невдачі базового RAG, що зумовило інтенсивну подальшу роботу над якістю pipeline. **BizBench** [71] оцінює кількісне міркування через генерацію Python-коду на основі фінансової документації – підхід, що поєднує гнучкість LLM з точністю детермінованих обчислень. **FinBen** [20] – комплексна оцінка з 36 датасетів та 24 задач; ключовий висновок огляду – LLM відносно сильні в Information Extraction (IE – задачі вилучення структурованих фактів з тексту) та класифікації, але слабкі у складному числовому міркуванні [20].

Нові напрямки (2024). FinDABench [70] оцінює здатність LLM до фінансового аналізу даних: GPT-4 у zero-shot режимі досягає лише 32,4% – наочний приклад прірви між generic-моделями та реальною аналітичною роботою. **FAMMA** [21] (1 945 питань) – мультимодальний і мультимовний бенчмарк: GPT-o1 досягає лише 30% на складних задачах. **FinTextQA** [72] адресує long-form QA з 1 262 парами – формат, у якому коротка відповідь неприйнятна, потрібен розгорнутий аналітичний текст. **OmniEval** [22] – перший спеціалізований бенчмарк для RAG-систем у фінансах (5 типів задач × 16 тем), що дозволяє оцінювати окремо якість retrieval та генерації.

Таблиця 1.4 – Порівняння основних бенчмарків для оцінки NLP-моделей у фінансовій сфері

Бенчмарк	**Рік**	**Фокус**	**Масштаб**	**Ключовий результат**
FinQA [16]	2021	Числове міркування	8 281 QA	Модель 61,2%, експерт 91,2%
TAT-QA [17]	2021	Гібридне QA	16 552	Модель 58,0%, експерт 90,8%

Продовження на наступній сторінці

Продовження таблиці 1.4

Бенчмарк	**Рік**	**Фокус**	**Масштаб**	**Ключовий результат**
ConvFinQA [18]	2022	Діалогове міркування	14 115	GPT-3 <50%, експерт 89,4%
FinanceBench [19]	2023	Open-book QA	10 231	GPT-4-Turbo: 81% помилок
BizBench [71]	2023	Через код	8 задач	Проблема – фінансові знання
FinBen [20]	2024	Комплексна оцінка	36 датасетів	GPT-4 в IE, Gemini у генерації
FinDABench [70]	2024	Аналіз даних	6 задач	GPT-4 zero-shot: 32,4%
FAMMA [21]	2024	Мультимодальне	1 945	GPT-o1: 30% на складних
FinTextQA [72]	2024	Long-form QA	1 262	RAG ≈ GPT-3.5
OmniEval [22]	2024	RAG у фінансах	11,4K	5 задач × 16 тем

Найпотужніші комерційні LLM далекі від надійного розв'язання фінансових задач – рівень помилок (як-от 81% у FinanceBench для GPT-4-Turbo) неприйнятний для промислу. Мультимовний вимір залишається слабко покритим – переважна більшість бенчмарків англомовні. Власний benchmark, побудований у межах поточного дослідження, частково заповнює цю прогалину: він орієнтований на ESEF-звіти європейських компаній, охоплює мікс структурованих XBRL-тегованих даних та narrative-секцій, тестує здатність системи відповідати рідною мовою користувача на запит до документа іншої мови – сценарій, не покритий жодним із оглянутих бенчмарків.

1.3.3. RAG-системи для фінансового аналізу

Здатність моделей точно працювати з документами поза навчальними даними залишається проблемою, що стимулює розвиток RAG у фінансовому домені. Розглянемо ключові системи, які сформували практичний стандарт станом на 2024–2025 роки.

HybridRAG від BlackRock та NVIDIA [23] поєднує дві комплементарні стратегії пошуку. VectorRAG використовує семантичний пошук за щільними

векторними представленнями фрагментів – швидкий і ефективний для фактологічних запитів. GraphRAG будує граф знань над сутностями (компанії, продукти, події, керівники) та відношеннями між ними, екстрагованими з документів за допомогою LLM, що дозволяє виконувати запити з інтеграцією інформації з різних частин корпусу. Об'єднання результатів обох стратегій із ваговими коефіцієнтами та подача релевантних фрагментів у генеративну модель дозволили на корпусі стенограм Earnings Calls індійської біржі Nifty 50 досягти показників Faithfulness 0,96 та Answer Relevancy 0,96 (метрики RAGAS), проти 0,94 і 0,91 у класичного VectorRAG [23]. Перевага особливо помітна на запитах із перехресними посиланнями – типовий приклад фінансових документів, де обговорення однієї події (наприклад, M&A-угоди) розкидане по різних розділах звіту.

Setty et al. [26] провели систематичний аналіз джерел невдач RAG на корпусі звітів SEC 10-K та довели: головною причиною неточних відповідей є саме таки неякісний пошук (retrieval), а не помилки генеративної моделі, як могло здатися. Типовий приклад провалу – фіксоване фрагментування розриває таблицю посередині рядка, через що числовий контекст втрачається у векторному поданні фрагмента; LLM при цьому генерує впевнену, але неправильну відповідь, оскільки оперує лише отриманими шматками. Висновок авторів – інженерія chunking та re-ranking дає більший приріст якості, ніж заміна LLM на потужнішу модель за такої самої pipeline.

Kim et al. [25] (ICLR 2025 Workshop) пропонують трифазний пайплайн для довгих фінансових звітів формату 10-K. Перша фаза – fine-tuned embedding-модель, дотренована на парах "фінансовий запит – релевантний фрагмент" для подолання domain shift від загальних embedding-моделей. Друга фаза – Multi-Vector Retrieval з агрегацією за різними рівнями деталізації документа (розділ, абзац, речення). Третя фаза – генеративна модель, додатково вирівняна через Direct Preference Optimization (DPO) на фінансових preference-pairs для зменшення галюцинацій. Архітектура демонструє виразні переваги саме на тих

типах запитів, де базовий RAG провалюється – питання, що вимагають інтеграції числових даних із різних розділів звіту, що зустрічається досить часто.

MultiFinRAG від S&P Global Ratings [24] адресує мультимодальну природу 10-K і 10-Q звітів: текст, таблиці та діаграми. Система використовує квантовану мультимодальну LLM Gemma3:12B (квантизація Q4_K_M зменшує розмір моделі приблизно з 24 ГБ до 7 ГБ без помітної втрати якості – критично для on-premise розгортання у фінансових установах із регуляторними обмеженнями на передачу даних у хмарні API). Таблиці обробляються спеціалізованим pipeline-структурного аналізу, діаграми – візуальним енкодером із подальшим текстовим описом. На внутрішньому бенчмарку S&P система досягла +19 відсоткових пунктів точності проти ChatGPT-4o [24].

Альтернативним напрямом є **мультиагентні архітектури**. Fatemi та Hu [73] реалізують схему “аналітик + критик”: перший LLM-агент генерує попередню відповідь, другий LLM-агент перевіряє її обґрунтованість витягнутими фрагментами і вказує на потенційні помилки; перший агент уточнює відповідь з урахуванням критики в ітеративному циклі. На FinQA з LLaMA3-8B такий підхід підвищує Exact Match з 54,67% до 72,48% – це доводить, що навіть без переходу на потужнішу LLM додавання шару рефлексії дає істотний приріст точності [73]. **Multi-Reranker** [75] виборов 2-ге місце на ACM-ICAIF ’24 на семи фінансових датасетах завдяки двоступеневому re-ranking: перший етап ранжує топ-200 кандидатів швидким bi-encoder, другий – переранжовує топ-10 крос-енкодером, що моделює повну взаємодію між запитом і фрагментом за рахунок дорожчого, але точнішого обчислення.

Таблиця 1.5 – Порівняння RAG-систем для фінансового аналізу

Система	**Рік**	**Тип даних**	**Інновація**	**Результати**
HybridRAG [23]	2024	Earnings calls	VectorRAG + GraphRAG	F=0.96 (RAGAS)

Продовження на наступній сторінці

Продовження таблиці 1.5

Система	**Рік**	**Тип даних**	**Інновація**	**Результати**
Setty et al. [26]	2024	SEC 10-K	Chunking, re-ranking	Якісний аналіз
Kim et al. [25]	2025	10-K	Трифазний з DPO	ICLR 2025 Workshop
MultiFinRAG [24]	2025	10-K/10-Q	Мультимодальна	+19 в.п. vs ChatGPT-4o
Multi-Agent [73]	2024	FinQA	Мультиагентна рефлексія	+15% EM (LLaMA3-8B)
Multi-Reranker [75]	2024	7 датасетів	Двоступеневий re-ranking	2-ге місце ACM-ICAIF ’24

Усі розглянуті системи працюють з американськими (SEC) або індійськими (Nifty 50) документами. Жодна не адресує специфіку європейської звітності: ESEF/iXBRL, мультимовність ЄС, OAM. Ця суттєва прогалина для європейського ринку – ключовий мотив цього дослідження.

1.3.4. Відсутність рішень для європейського ринку

Аналіз з розділів 1.3.1 – 1.3.3 демонструє чітку тенденцію: переважна більшість досліджень фінансового NLP та RAG-систем орієнтована на американський або китайський ринки. Бенчмарки FinQA, TAT-QA, ConvFinQA, FinanceBench побудовані на звітності компаній SEC. Спеціалізовані моделі (BloombergGPT, FinGPT) навчалися переважно на англомовних даних США. RAG-системи тестувалися на корпусах американських 10-K та 10-Q.

Серед нечисленних винятків європейського контексту виокремлюються лише два проєкти, обидва зосереджені на німецькомовних текстах. German FinBERT [27] – попередньо навчена модель архітектури BERT, адаптована під німецькомовні фінансові тексти. Корпус навчання – понад 10 млрд токенів (53,19 ГБ): річні звіти з Bundesanzeiger, фінансові новини з Handelsblatt та MarketScreener, ad-hoc повідомлення, Вікіпедія. Версія further pre-trained показала найкращі результати: macro F1 = 86,08% (класифікація тем), F1 = 74,61% (QA), точність 95,41% (тональність) [27]. Автор статті зазначає обмеження – єдиний вручну розмічений датасет (ad-hoc повідомлення), два інші згенеровані ChatGPT [27].

KPI-BERT – система спільного NER та RE від Fraunhofer IAIS і PricewaterhouseCoopers для автоматичної ідентифікації KPI у німецьких фінансових документах [28]. Базується на BERT з GRU-шаром та умовним маскуванням міток для послідовного тегування “kpi”, “поточне значення”, “значення попереднього року” [28]. На тестовому наборі: F1 = 81,08% на NER, F1 = 70,88% на класифікації зв’язків, перевершує SpERT [28]. Інтегрована в аудиторський процес. Працює лише з німецькою мовою та вузьким завданням – вилучення числових KPI, а не комплексний аналіз звітності, особливо різними мовами.

Обидва проєкти ілюструють одну проблему: європейські інструменти обмежені однією мовою (німецькою), одним типом завдання та не пропонують комплексного рішення для аналізу звітності. Жоден не реалізує повноцінну RAG-архітектуру для довільних аналітичних запитів до корпусу європейських звітів різними мовами.

Ця прогалина формує основну дослідницьку мотивацію роботи: створення RAG-системи, здатної працювати з мультимовним корпусом європейської фінансової звітності у форматі ESEF – завдання, яке досі не було адресовано в жодному з проаналізованих досліджень.

1.4. Постановка задачі

Проведений у попередніх підрозділах аналіз дозволяє чітко окреслити дослідницьку проблему та сформулювати задачу роботи.

У розділі 1.1 показано, що ЄС створив розгалужену регуляторну інфраструктуру корпоративної звітності – від Директиви про прозорість 2004/109/ЄС до єдиного електронного формату ESEF та перспективної платформи ESAP. Ця інфраструктура генерує значний обсяг машиночитних фінансових документів: лише на платформі financialreports.eu доступні тисячі звітів від компаній із усіх країх ЄС та інших країн Європи. Проте, як показано у розділі 1.2, автоматизований аналіз цих документів наштовхується на низку

бар'єрів: масштаб корпусу, мультимовність (24+ офіційних мов ЄС), структурна різнорідність між країнами та галузями, проблеми якості XBRL-тегування.

Огляд існуючих систем (розділ 1.3) засвідчив активний розвиток інструментів фінансового NLP: спеціалізовані мовні моделі (BloombergGPT [9], FinGPT [10], TAT-LLM [15]), бенчмарки (FinQA [16], FinanceBench [19], FinBen [20]), RAG-системи (HybridRAG [23], MultiFinRAG [24], Kim et al. [25]). Однак практично всі ці розробки орієнтовані на документи SEC та англомовний контент. Навіть поодинокі європейські ініціативи – German FinBERT [27] та KPI-BERT [28] – обмежені окремими мовами і вузькими задачами.

Існує великий розрив між зростаючим обсягом структурованої європейської фінансової звітності та відсутністю інтелектуальних інструментів для її аналізу. Технологія RAG, яка поєднує семантичний пошук із генеративними можливостями LLM, є перспективним підходом, оскільки дозволяє працювати безпосередньо з повним текстом документів, не покладаючись на, у більшості випадків, потенційно некоректне XBRL-тегування.

Таким чином, проведений огляд виявив чітку прогалину: інструментів, які дозволяли б непрофесійним користувачам ставити природномовні запити до мультимовного корпусу європейської звітності у форматі ESEF із гарантією верифікованості, на момент дослідження не існує. Цей розрив підтверджується відомими бенчмарками: FinQA фіксує перевагу експертів (91,16%) над FinQANet (61,24%) у понад 30 п.п., а FinanceBench показав 81% помилок GPT-4-Turbo з retrieval. Розв'язання цієї прогалини є предметом наступних розділів.

РОЗДІЛ 2. ТЕОРЕТИЧНІ ОСНОВИ RAG-СИСТЕМ ТА ОБҐРУНТУВАННЯ ТЕХНОЛОГІЧНОГО СТЕКУ

2.1. Архітектура Retrieval-Augmented Generation

Підхід Retrieval-Augmented Generation (RAG) запропоновано Lewis et al. у 2020 році [29] як спосіб подолати фундаментальне обмеження великих мовних моделей – залежність від параметричної пам'яті. Основна ідея – поєднання двох типів пам'яті: параметричної (мовна модель seq2seq на базі BART) та непараметричної (щільний векторний індекс документів через Dense Passage Retriever). Перед генерацією система знаходить у зовнішньому сховищі релевантні фрагменти тексту і подає їх як додатковий контекст генеративній моделі [29]. Lewis et al. запропонували два формулювання: RAG-Sequence (один набір документів для всієї вихідної послідовності) та RAG-Token (окремий документ для кожного токена). RAG досяг найкращих результатів на задачах відкритого QA, перевершивши параметричні моделі та спеціалізовані архітектури при значно меншій кількості параметрів (626M проти T5-11B) [29].

Gao et al. [30] виокремлюють три послідовні парадигми RAG-архітектур, кожна з яких відображає зростаючий рівень технологічної зрілості та складності реалізації.

Naive RAG – базова реалізація схеми "витягни-і-прочитай" (retrieve-then-read), що складається з трьох послідовних кроків: документи розбиваються на фрагменти, кодуються embedding-моделлю та зберігаються у векторній базі; під час запиту витягуються top-K найбільш семантично близьких фрагментів; релевантні фрагменти передаються генеративній моделі разом із оригінальним запитом для побудови відповіді [30]. Перевагою цього підходу є простота, передбачуваність та можливість прозорого аналізу впливу кожного компонента ізольовано. Обмеження – низька точність пошуку при семантично складних або багатозначних запитах, ризик галюцинацій при потраплянні нерелевантних

фрагментів у контекст, а також дублювання інформації, коли близькі за семантикою фрагменти однаково ранжуються [30].

Advanced RAG впроваджує оптимізації на двох ключових фазах. Pre-retrieval-етап включає розширення запитів через генерацію перефразувань або гіпотетичних відповідей (HyDE), збагачення метаданими та query routing для вибору цільового джерела пошуку. Post-retrieval-етап реалізує ре-ранкінг через крос-енкодери (зокрема bge-reranker або Cohere Rerank), стиснення контексту через суммаризацію та фільтрацію за релевантністю. Ці оптимізації значно покращують якість пошуку, але ускладнюють відлагодження та підвищують час відповіді [30].

Modular RAG передбачає модульну архітектуру з можливістю заміни компонентів та новими патернами координації: Rewrite-Retrieve-Read (запит попередньо перефразовується для покращення семантичного зіставлення), Generate-Read (генеративна модель видає попередню відповідь як query expansion), ітеративні цикли ITER-RETGEN із багаторазовим уточненням, адаптивний пошук через FLARE (модель самостійно вирішує, коли потрібен пошук, на основі впевненості у наступному токені) та Self-RAG [30].

Huang та Huang [33] систематизують RAG з перспективи інформаційного пошуку у чотири фази: pre-retrieval (індексація, маніпуляція з запитами), retrieval (пошук та ранжування), post-retrieval (ре-ранкінг та фільтрація) і generation (збагачення, верифікація відповіді). Цей фреймворк однаково застосовний до однокрокових і мультикрокових циклів “пошук-генерація” [33].

Self-RAG (Asai et al., 2023) реалізує адаптивний підхід до пошуку через чотири типи спеціальних токенів рефлексії, які модель навчена вставляти у вихідну послідовність: [Retrieve] (вирішує, чи потрібен пошук для поточного фрагмента), [IsRel] (оцінює релевантність витягнутого фрагмента), [IsSup] (підтверджує, чи відповідь обґрунтована джерелом), [IsUse] (оцінює корисність кандидата відповіді для запиту) [31]. Підхід дозволяє моделі самостійно регулювати інтенсивність пошуку та надає прозорий механізм самоверифікації

– критично важливий для застосувань із високими вимогами до достовірності, як-от фінансовий аналіз. RAPTOR (Sarthi et al., 2024) вирішує задачу пошуку у довгих документах через рекурсивну кластеризацію фрагментів і генерацію резюме для кожного кластера через LLM, формуючи дерево абстракцій з декількох рівнів узагальнення [31]. Під час пошуку обираються фрагменти з різних рівнів дерева – від конкретних фактологічних деталей до високорівневих узагальнень розділів. У поєднанні з GPT-4 RAPTOR покращив найкращий попередній результат на QuALITY (бенчмарк QA на довгих текстах) на 20% абсолютної точності. RAFT (Zhang et al., 2024) тренує модель явно ігнорувати "документи-відволікачі" та цитувати релевантні джерела через supervised fine-tuning із синтетичними даними [31]. Fan et al. [32] на KDD 2024 систематизують RA-LLMs з перспектив архітектури, навчання та застосувань. Zhao et al. [64] пропонують таксономію RAG у контексті AIGC, охоплюючи текст, зображення, аудіо та відео. Ni et al. [65] вводять поняття "довірчого RAG" з п'ятьма напрямами: надійність, конфіденційність, безпека, справедливість, інтерпретованість.

Архітектура RAG-системи складається з двох фаз. Офлайн-індексація: збір та очищення документів, поділ на фрагменти, перетворення у вектори, збереження у векторній базі. Онлайн-обробка: векторизація запиту, пошук top-K семантично близьких фрагментів, опційна post-retrieval обробка, генерація відповіді. Кожен компонент – стратегія chunking, embedding-модель, векторна база, метод пошуку, генеративна модель – впливає на якість відповіді. У межах поточної роботи прототип реалізує змішану парадигму **Naive and Advanced RAG**: цей вибір зумовлений трьома міркуваннями. По-перше, прозорою декомпозицією компонентів, що дозволяє ізольовано оцінювати вплив embedding-моделі та LLM на якість відповідей – це є ключовою метою порівняльного дослідження. По-друге, відтворюваністю результатів та зменшенням ризику конфаундингу між технічними рішеннями: будь-яка оптимізація Advanced або Modular рівня може приховати різницю у поведінці

базових компонентів. По-третє, залучення деяких з практик **Advanced RAG**: збагачення метаданими та мітками, допомагає значно покращити результат та швидкість обробки запиту без повного занурення у інші ключові фази, залишаючись у рамках простоти та відтворюваності. Детальний аналіз кожного компонента наведено у наступних підрозділах.

2.2. Стратегії поділу документів на фрагменти

Поділ документів на фрагменти (chunking) – один із критичних етапів RAG, оскільки якість фрагментації прямо впливає на ефективність пошуку та генерації. Barnett та співавтори у дослідженні семи типових точок відмови RAG-систем встановили, що якість фрагментації визначає векторні представлення фрагментів та їх відповідність запитам [34]. Малі фрагменти втрачають контекст, великі – додають шум [34]. Виокремлюють два фундаментальні підходи: евристичний (на основі пунктуації, кінця абзацу) та семантичний (з урахуванням семантики тексту для визначення меж фрагментів) [34]. Практичний ландшафт стратегій можна впорядкувати за зростанням обчислювальної складності: фіксований і рекурсивний поділ – швидкі і не потребують додаткових моделей; структурний – використовує метадані документа без додаткового навчання; семантичний та ієрархічний (RAPTOR-подібний) – потребують embedding-моделей або LLM на етапі індексації; логіт-керовані та мультигранулярні підходи – найдорожчі, але дають найвищу якість на складних доменах.

Фіксований поділ розбиває текст на фрагменти заданої довжини в символах або токенах [39]. Перевагами є простота та передбачуваність, проте підхід ігнорує структуру та семантику. Jimeno Yepes та співавтори порівняли фіксований поділ із 128, 256 та 512 токенами на фінансових звітах SEC [35]: фрагменти 128 токенів забезпечують найвищу точність пошуку на рівні сторінок (72,34%), але найнижчу за ROUGE (0,383) та BLEU (0,181). 512 токенів дають нижчу сторінкову точність (68,09%), але вищі ROUGE 0,455 та BLEU 0,250 [35].

Агрегація трьох конфігурацій дає 83,69% сторінкової точності ціною потрійного збільшення фрагментів [35].

Рекурсивний поділ ієрархічно застосовує набір роздільників: спочатку за розділами/абзацами, потім – реченнями та словами, доки фрагмент не стане меншим за поріг [39]. Реалізований у `RecursiveCharacterTextSplitter` LangChain з послідовністю `\n\n`, `\n`, пробілом. Краще зберігає структуру тексту, але залежить від наперед визначених роздільників. Liu et al. [81] на бенчмарку GutenQA отримали DCG@1 лише 47,22%, що поступалось більш просунутим методам.

Семантичний поділ визначає межі за косинусною подібністю векторних представлень речень [39]. **ChunkRAG** (Singh et al.) поєднує семантичне фрагментування з багаторівневою фільтрацією – кожен витягнутий фрагмент проходить кілька оцінювачів релевантності з різними критеріями (LLM-based, словниковий, метаданий) – та з гібридом BM25+dense, де лексичний пошук BM25 повертає кандидатів за збігом ключових слів, а dense-пошук – за семантичною близькістю; результати об'єднуються через нормалізоване ранжування. Підхід досяг 64,9% на PopQA, 77,3% на PubHealth, 86,4% FactScore на Biography, перевершивши Self-RAG та CRAG [36]. **SAGE** (Zhang et al.) тренує легку модель сегментації замість використання GPT-4 (що коштувало б >$90 на корпус 10000000 токенів) та використовує градієнтний вибір фрагментів – алгоритм оцінює зміну ймовірності правильної відповіді при додаванні/видаленні кожного кандидата і обирає підмножину з максимальним внеском, аналогічно до feature selection у класичному ML. Покращив якість відповідей у середньому на 61,25% та ефективність витрат на 49,41% [37].

Sarthi et al. (Stanford) рекурсивно кластеризують фрагменти, формують резюме для кожного кластера через LLM та повторюють процес – будуючи дерево абстракцій [38]. Типовий приклад для фінансового звіту: рівень 0 містить оригінальні фрагменти ≈500 токенів, рівень 1 – резюме розділів (наприклад, MD&A, Risk Factors), рівень 2 – резюме всього звіту. Під час пошуку обираються

фрагменти з різних рівнів – це дозволяє відповідати як на конкретні (рівень 0), так і на узагальнюючі (рівень 2) питання. У поєднанні з GPT-4 RAPTOR покращив найкращий попередній результат на QuALITY на 20% абсолютної точності [38].

Структурний поділ використовує розмітку документа: заголовки, параграфи, таблиці, списки. Jimeno Yepes et al. реалізували через модель Chipper з розпізнаванням типів елементів. Забезпечив 53,19% точності проти 48,23% для Base 512 при зіставній кількості фрагментів [35]. S^2 Chunking (Verma) поєднує просторовий та семантичний аналіз через спектральну кластеризацію зваженого графа елементів [39].

Liu et al. (BNP Paribas, Ecole Polytechnique) поєднують логіт-керований поділ за ймовірністю токена [EOS] (через Llama3-8b) та мультигранулярне фрагментування на “дочірні” різної деталізації [81]. На GutenQA досяг DCG@1 = 63,00% проти 47,22% рекурсивного та 41,33% семантичного [81].

Порівняння стратегій узагальнено в таблиці 2.1.

Таблиця 2.1 – Порівняння стратегій поділу документів на фрагменти

Стратегія	**Принцип**	**Переваги**	**Обмеження**	**Джерело**
Фіксований поділ	Розбиття на фрагменти заданої довжини з перекриттям	Простота, передбачуваність	Ігнорує семантику, розриває логічні зв’язки	[35], [39]
Рекурсивний поділ	Ієрархічне застосування роздільників	Зберігає структуру абзаців	Залежить від роздільників	[39], [80]
Семантичний поділ	Межі за косинусною подібністю ембедингів	Когерентні фрагменти	Обчислювально затратний	[36], [37]
Ієрархічний (RAPTOR)	Рекурсивне кластерування та реферування	Багаторівнева абстракція	Висока складність, потребує LLM	[38]
Структурний (Chipper)	Типи елементів документа	Зберігає структуру	Потребує модель розпізнавання макета	[35], [39]
Мультигранулярний (LGMGC)	Логіт-керований + дочірні фрагменти	Стабільність при різних розмірах	Потребує LLM для [EOS]	[80]

У прототипі обрано рекурсивний поділ через `RecursiveCharacterTextSplitter` LangChain з `chunk_size=2500` символів та `chunk_overlap=250`. Документи корпусу financialreports.eu вже у форматі Markdown зі структурними маркерами – заголовками, розділами – які сплітер використовує як роздільники. Розмір 2500 символів – компроміс між збереженням контексту фінансових таблиць/наративних описів та ефективністю пошуку: як показали Jimeno Yepes et al., надто малі фрагменти втрачають контекст, тоді як ≈2000 символів забезпечують баланс [35]. Перекриття 250 символів (10%) зменшує ризик втрати інформації на межах – проблему типу FP2 за Barnett et al. [34]. Рекурсивний підхід практичний: не потребує додаткових моделей (на відміну від LGMGC, Chipper), не залежить від API (ChunkRAG) та не вимагає тривалої побудови дерев (RAPTOR). Більш складні стратегії – напрям подальшого вдосконалення.

2.3. Моделі векторних представлень

Якість пошуку у RAG-системі напряму залежить від моделей векторних представлень (embedding models) – нейронних мереж, що перетворюють текст у числові вектори фіксованої розмірності. Семантично близькі тексти отримують геометрично близькі вектори, що дає змогу використовувати косинусну подібність як міру релевантності між запитом і документом.

Еволюція embedding-моделей пройшла кілька етапів. Ранні підходи (GloVe) формували статичні вектори без урахування контексту [40]. З появою BERT стали можливими контекстно-залежні представлення, а Sentence-BERT продемонстрував переваги додаткового дотренування трансформера з контрастивною функцією втрат [40]. Сучасний ландшафт розширився від компактних енкодерів до підходів з LLM-backbone у мільярди параметрів.

2.3.1. Бенчмарки для оцінки embedding-моделей

MTEB (Massive Text Embedding Benchmark) [40] охоплює 8 типів задач (пошук, кластеризація, класифікація, семантична подібність, ранжування, парна

класифікація, bitext mining, сумаризація) на 58 датасетах та 112 мовах. Жодна модель не домінує одночасно на всіх типах задач: SimCSE сильна на STS, але слабка на пошуку та кластеризації [40]. MTEB з відкритим кодом та публічним лідербордом на Hugging Face став де-факто стандартом.

BEIR (Benchmarking IR) [42] зосереджений на zero-shot оцінці систем інформаційного пошуку: 18 датасетів з 9 типів задач у різних доменах. Ключовий висновок: BM25 залишається надійним baseline, а dense моделі, що перевершують BM25 на навчальних даних, часто поступаються йому у zero-shot сценаріях [42] – проблема узагальнення, особливо актуальна для фінансової звітності з різних країн.

MMTEB (Massive Multilingual Text Embedding Benchmark) [41] – мультимовне розширення MTEB з понад 500 задачами, 10 категоріями та 250+ мовами. Нові типи: слідування інструкціям, пошук у довгих документах, code retrieval. Найкращою публічно доступною моделлю на мультимовних бенчмарках виявилась `multilingual-e5-large-instruct` із лише 560M параметрів, що перевершила значно більші LLM-based моделі [41] у тестах.

2.3.2. Покоління embedding-моделей та ключові підходи

Огляд Cao (2024) [46] виділяє дві групи: моделі з фокусом на даних (E5, BGE, GTE) та LLM-based (E5-Mistral, Gecko, NV-Embed).

E5 (Wang et al., 2022) [69] продемонструвала ефективність масштабного контрастивного попереднього навчання на CCPairs – 270M текстових пар з consistency-фільтра з початкових 1,3 млрд. Перша модель, що перевершила BM25 на BEIR у zero-shot без розмічених даних. E5-base з дотренуванням конкурувала з моделями у 40 разів більшими [69]. BGE (BAAI General Embedding) використовує багатоетапний рецепт навчання з пакетом C-Pack та інструкціями до запитів [46]. GTE навчалась на ≈800M текстових пар з різноманітних джерел, включно з кодом [46].

E5-Mistral (Wang et al., 2024) [67] – перехідна модель: дотренування Mistral-7B виключно на синтетичних даних від іншої LLM менш ніж за 1000 кроків. Досягла state-of-the-art на BEIR та MTEB. Ключовий механізм – текстові інструкції, що додаються до запиту [46]. Навчання через LoRA (rank 16, 42M тренованих параметрів) – лише 576 годин на графічних процесорах (GPU) V100 [46]. Gecko (Google) [68] використовує дистиляцію знань з LLM: спочатку LLM генерує синтетичні пари "задача+запит", потім переранжовує кандидати, визначаючи позитивні та складні негативні приклади. З 1,2B параметрів та 256-вимірними embeddings перевершила всі моделі з 768-вимірними embeddings на MTEB [68]. NV-Embed (NVIDIA) [45] використовує latent attention замість усереднення. V2 встановила рекорд із середнім балом 72.31 на 56 задачах MTEB у серпні 2024.

2.3.3. Відкриті embedding-моделі: паритет з комерційними

У 2024 році відкриті embedding-моделі досягли паритету з OpenAI та Cohere. Arctic-Embed (Snowflake) [43] – сімейство з п'яти моделей (22–334M параметрів) під Apache-2. Найбільша `arctic-embed-l` перевершила Cohere embed-v3 та OpenAI text-embedding-3-large на MTEB Retrieval. Nomic Embed [44] – перша повністю відтворювана embedding-модель з відкритими вагами, даними та кодом під Apache-2. З контекстом 8192 токени перевершила OpenAI Ada-002 та text-embedding-3-small. Jina Embeddings v3 [49] – 570M параметрів, архітектурно інноваційний підхід з п'ятьма Task LoRA-адаптерами (retrieval.query, retrieval.passage, separation, classification, text-matching) на основі XLM-RoBERTa. Перевершила комерційні моделі OpenAI/Cohere на MTEB та `multilingual-e5-large-instruct` на мультимовних задачах.

2.3.4. Доменна специфічність та складні сценарії

Висока продуктивність на загальних бенчмарках не гарантує результату на спеціалізованих текстах. FinMTEB (Tang та Yang, 2024) [47] – фінансовий

аналог MTEB з 64 датасетами із семи типів задач: банківські корпуси, фінансові звіти, новини, earnings calls. Тестування семи провідних embedding-моделей виявило значне падіння продуктивності та слабку кореляцію між MTEB та FinMTEB – це обґрунтовує потребу у доменно-специфічних бенчмарках та моделях [47]. BRIGHT (2024) [48] фокусується на reasoning-intensive retrieval – 1384 запити з економіки, психології, математики, програмування, де потрібні логічні міркування. Найкраща MTEB-модель SFR-Embedding-Mistral (nDCG@10 = 59,0) показала лише 18,3 на BRIGHT – падіння втричі [48]. Додавання LLM-кроків міркування покращило на 12,2 пункти, але загальний результат все таки залишався нижче 30.

2.3.5. Порівняння embedding-моделей

Таблиця 2.2 – Порівняльна характеристика embedding-моделей

Модель	**Параметри**	**Розмір-ність**	**Контекст**	**Архітектура**	**Ліцензія**	**Особливість**
E5-base/large [69]	110M / 335M	768 / 1024	512	Encoder (BERT)	MIT	CCPairs: 270M пар з consistency
multilingual-e5-large-instruct [41]	560M	1024	514	Encoder (XLM-R)	MIT	Мультимовне instruction tuning
E5-Mistral-7B [67]	7B (42M LoRA)	4096	32K	Decoder (Mistral)	MIT	Синтетичні дані + <1K кроків
BGE-large-en-v1.5 [46]	326M	1024	512	Encoder (BERT)	MIT	C-Pack + instruction prefix
Arctic-Embed-L [43]	334M	1024	512	Encoder	Apache-2	Stratified batches + tuned negatives
Nomic Embed v1 [44]	137M	768	8192	Encoder	Apache-2	Повна відтворюваність
NV-Embed-v2 [45]	≈7B	4096	32K	Decoder (Mistral)	CC-BY-NC	Latent attention + mixed tuning
Gecko [68]	1.2B	256–768	2048	Encoder (T5)	Закрита	LLM-дистиляція
Jina v3 [49]	570M	32–1024	8192	Encoder (XLM-R)	CC-BY-NC	Task LoRA (5 адаптерів)

Продовження на наступній сторінці

Продовження таблиці 2.2

Модель	**Параметри**	**Розмір-ність**	**Контекст**	**Архітектура**	**Ліцензія**	**Особливість**
all-mpnet-base-v2 [40]	109M	768	384	Encoder (MPNet)	Apache-2	Баланс швидкості та якості

Аналіз виявляє кілька закономірностей. Існує дихотомія між компактними encoder-моделями (100–600M параметрів) та LLM-based (7B+) – перші практичніші для локального розгортання при прийнятній якості. Підходи з синтетичними даними від LLM (E5-Mistral, Gecko, NV-Embed) послідовно дають вищі результати – підтверджує ефективність дистиляції. Моделі з довгим контекстом (Nomic Embed, Jina v3) особливо цінні для фінансових документів.

У прототипі використовуються три embedding-моделі: `multilingual-e5-large` (560M), `bge-large-en-v1.5` (326M) та `all-mpnet-base-v2` (109M). Вибір `multilingual-e5-large` зумовлений потребою обробки мультимовних європейських звітів – за MMTEB це найкраща публічно доступна мультимовна модель [41]. `bge-large-en-v1.5` додано як англомовну модель з приблизно тієї ж розмірної групи, що дозволяє ізолювати ефект мультимовності від ефекту розміру моделі. `all-mpnet-base-v2` – компактний baseline для оцінки виправданості більших моделей. Детальне порівняння їх ефективності – у розділі 3.7.

2.4. Векторні бази даних

Центральна операція векторної БД – пошук k найближчих сусідів (k-NN) для вектора запиту. На колекціях у сотні тисяч векторів точний пошук стає надто повільним через “прокляття розмірності” [50], що зумовлює потребу в наближеному пошуку (ANN). Основні алгоритми ANN: IVF (k-means кластеризація) [50], Product Quantization (PQ – стиснення векторів через підвектори, IVFADC ефективний на мільярді векторів) [50], HNSW (багатошаровий навігаційний граф) [64] – домінуючий алгоритм у сучасних векторних БД завдяки балансу recall/throughput [30]. Інші підходи (KD-дерева,

LSH) поступаються графовим методам. Wu et al. [63] детально описують способи кодування фрагментів та алгоритми індексування – від BM25 до HNSW і деревоподібних структур.

Таблиця 2.3 – Порівняння векторних баз даних для RAG-систем

Система	**Тип**	**Алгоритми ANN**	**Метадані**	**Гібрид. пошук**	**Ліцензія**
FAISS [50]	Бібліотека	IVF, PQ, HNSW, Flat	Ні	Ні	MIT
ChromaDB	Вбудована БД	HNSW	Так	Ні	Apache 2.0
Milvus	Розподілена БД	IVF, HNSW, DiskANN	Так	Так	Apache 2.0
Pinecone	Керований сервіс	Власний	Так	Так	Комерційна
Weaviate	Open-source	HNSW	Так	Так (нативно)	BSD-3
Qdrant	Open-source	HNSW	Так (розширена)	Так	Apache 2.0

FAISS (Meta) [50] забезпечує побудову графа k-NN на мільярді векторів за <12 годин на 4 GPU. ChromaDB [30] – легка вбудовувана БД з Python API, оптимальна для прототипів. Milvus, Weaviate, Qdrant – розподілені рішення для мільярдних колекцій. Pinecone – керований хмарний сервіс.

Корпус – 4 528 документів, ≈822 тис. фрагментів – не вимагає розподілених рішень. ChromaDB забезпечує найпростішу інтеграцію з LangChain, вбудоване зберігання тексту та метаданих спрощує фільтрацію за країною/компанією/роком/типом звіту. Для промислових систем з мільйонами документів доцільні Milvus або Qdrant. Деталі конфігурації – у розділі 3.4.

2.5. Гібридний пошук

Жоден окремий підхід не є універсальним. Sparse-методи (BM25) обчислюють релевантність за TF/IDF – точне зіставлення ключових слів і доменної термінології (EBITDA, 10-K), але страждають від лексичного розриву [42]. Dense-пошук захоплює семантичну подібність без спільних термінів, але потребує обчислювальних ресурсів і може втрачати ефективність на

out-of-distribution даних. Бенчмарк BEIR [42] показав: BM25 залишається надійним baseline у zero-shot крос-доменних сценаріях, тоді як dense-моделі (топ на MS MARCO) часто програють при переносі. Це мотивує гібридний пошук.

Cormack et al. [53] запропонували Reciprocal Rank Fusion (RRF): $RRFscore(d) = \sum_{r \in R} 1/(k + r(d))$, де $k = 60$ емпірично. RRF стабільно перевершує альтернативи на TREC у середньому на 4–5% MAP, працює лише з рангами – стійкий до відмінностей у шкалах [53]. Blended RAG (IBM) [51] поєднує BM25, Dense Vector та Sparse Encoder (ELSER). Досяг 88,77% top-10 на NQ та F1=68,4 на SQuAD без дофайнтюнінгу – на 50% вище за RAG-end2end. DAT (Hsu, Tzeng) [52] динамічно обчислює вагу α для кожного запиту через LLM-оцінку релевантності top-1 результатів. На SQuAD з GPT-4o досяг 92,34% точності α та Precision@1 на ≈2,8% вище за фіксований гібрид. GeAR (Shen et al.) [54] додає графовий вимір: LLM витягує семантичні трійки і розширює мультихоп-контексти – на MuSiQue покращення >10% за R@15. Mozolevskyi та AlShikh [82] порівняли вісім промислових RAG: спеціалізовані підходи (Writer Retrieval – 86,31 RobustQA) перевершують “коробкові” платформні рішення (Amazon SageMaker 32,74, Google VertexAI 51,08).

Sparse-етап (BM25) → dense-етап для семантичного ре-ранжування → cross-encoder для точного оцінювання пар “запит–документ”. Cross-encoder перевершують BM25 на 16 із 18 датасетів BEIR [42]. У прототипі гібридний пошук не реалізовано – визначено як напрям подальшого розвитку (розділ 4.6).

2.6. Генеративні мовні моделі у контексті RAG-систем

Генеративна модель – центральний компонент RAG: вона отримує фрагменти і формує відповідь. Якість залежить від релевантності пошуку та здатності моделі інтерпретувати контекст і уникати галюцинацій. Моделі поділяються на комерційні (GPT-4, Claude) та відкриті (Llama, Mistral, Qwen).

Комерційні залишаються орієнтиром: GPT-4 (OpenAI) – числове міркування, табличні дані. Claude (Anthropic) – детальні структуровані

відповіді. Обидві з контекстом 128K+. Проблеми – витрати на API та ризики конфіденційності. Ateia та Kruschwitz [61] наводять інциденти: зупинка ChatGPT через витік повідомлень у 2023, заборона ChatGPT у Samsung через передачу конфіденційних даних. Подібних ситуацій, котрі складали певний ризик, ще багато.

ChatQA NVIDIA [60] – двоетапний instruction tuning (звичайний + context-enhanced). Llama2-70B-ChatQA досягла 54,14 на CHATRAG BENCH проти GPT-4-0613 (53,90) та GPT-4-Turbo (54,03). Llama3-ChatQA-1.5-70B перевершила GPT-4-Turbo на 4,4% [60]. На BioASQ Mixtral 8x7B з 10-shot конкурентоспроможний з GPT-4 при значно нижчій вартості ($1,2 vs $12) [61]. Oketch et al. [62] показали: Qwen2.5-72B та Llama 3-70B у few-shot ≈ GPT-4 за MSE/MAE/QWK при у 37 разів нижчій ціні. Gautam та Purwar [59] на enterprise-даних з BGE та FAISS: Llama3-8B перевершив Mistral 8x7B (56B) за unigram precision (0,737–0,82 vs 0,67–0,73) та answer relevancy (0,93–1,00 vs 0,87–0,95). Обидві відкриті перевершили GPT-3.5 за contextual recall (0,916–0,98 vs 0,86). Збільшення top-k понад поріг не покращує якість.

Таблиця 2.4 – Порівняння генеративних мовних моделей для RAG-систем

Характеристика	**GPT-4/4o**	**Claude**	**Llama 3**	**Mixtral 8x7B**	**Qwen2.5-72B**
Тип	Закрита	Закрита	Відкрита	Відкрита	Відкрита
Якість RAG (CHATRAG)	54,03 [60]	–	57,14 [60]	–	–
vs GPT-4 (few-shot)	Базовий	Порівн.	Порівн./вищий [60,62]	Конкур. [61]	Порівн. [62]
Contextual recall	–	–	0,916–0,98 [59]	0,91–0,98 [59]	–
Вартість	1×	Висока	1/15–1/37× [62]	Низька	1/15–1/17× [62]
Конфіденційність	Ризик	Ризик	Повний контроль	Повний контроль	Повний контроль
Контекстне вікно	128K+	200K+	8K–128K	32K	128K

Відкриті моделі рівня Llama3-8B порівнянні з GPT-3.5. 70B+ конкурують з GPT-4. Двоетапний instruction tuning [60] дає більший ефект, ніж збільшення розміру – це пояснює, чому Llama3-8B конкурує з Mistral 8x7B у RAG [59]. Конфіденційність – реальне обмеження для корпоративної звітності [61]. Відкриті локальні моделі усувають ризик. У прототипі – комбінований підхід: локальні Llama3.1-8B та Mistral (Ollama) + комерційні Claude Sonnet та GPT-4o (API) для можливості прямого порівняння.

2.7. Методики оцінки якості RAG-систем

Оцінка якості RAG є окремою проблемою через модульну природу систем. Традиційні метрики (Exact Match, F1, ROUGE-L) оцінюють лише лексичний збіг, що недостатньо відображає семантичну правильність [79]. За останні два роки з'явилися спеціалізовані фреймворки.

RAGAS (Es et al., 2023) [55] – перший фреймворк reference-free оцінки з трьома метриками: faithfulness (частка тверджень відповіді, підтверджених контекстом), answer relevance (косинусна подібність між запитанням та відповіддю) та context relevance (частка релевантних речень контексту). На WikiEval досяг 95% узгодженості з людьми щодо faithfulness, 78% – answer relevance, 70% – context relevance [55]. Обмеження: фіксовані промпти погано адаптуються до доменів.

ARES (Saad-Falcon et al., 2023) [56] вирішує адаптивність через трьохетапний підхід: генерація синтетичних тренувальних даних, файн-тюн легких суддів (DeBERTa-v3-Large, 304M), prediction-powered inference з ≈150 людськими анотаціями. Kendall's τ для ранжування RAG-систем у середньому на 0.065 вище за RAGAS для context relevance і 0.132 – для answer relevance [56].

RAGChecker (Ru et al., 2024) [58] оцінює на рівні окремих тверджень (claim-level), розраховуючи precision/recall/F1 та модульні метрики окремо для retriever (claim recall, context precision) та generator (context utilization, noise sensitivity,

hallucination, self-knowledge). Кореляція Пірсона з людськими оцінками – 61,93 проти 48,31 у найкращої метрики RAGAS [58].

CoFE-RAG (Liu et al., 2024) [57] заповнює прогалину – оцінка всього пайплайну включно з chunking. Замість анотованих "золотих" фрагментів використовує мультигранулярні ключові слова. Бенчмарк-датасет містить чотири типи запитів (factual, analytical, comparative, tutorial) у форматах PDF, PPT, DOC, XLSX [57].

Інші фреймворки розширюють спектр: RAGEval [78] генерує сценарно-специфічні тести (Completeness, Hallucination, Irrelevance). IRSC [77] оцінює embedding-моделі у RAG-сценаріях через метрики SSCI та RCCI. BERGEN [79] стандартизує експерименти. RAGBench [80] – масштабний крос-доменний датасет із фреймворком TRACe (uTilization, Relevance, Adherence, Completeness).

Таблиця 2.5 – Порівняльна характеристика фреймворків оцінки якості RAG-систем

Фреймворк	**Рік**	**Підхід**	**Основні метрики**	**Еталон**
RAGAS [55]	2023	LLM-промптинг	Faithfulness, Answer/Context Relevance	Ні
ARES [56]	2023	Файн-тюн судді + PPI	Ті самі + довірчі інтервали	≈150 анотацій
CoFE-RAG [57]	2024	Мультигранулярні keywords	Recall/Accuracy на всьому ланцюзі	Так
RAGChecker [58]	2024	Claim-level entailment	Утилізація, шум, галюцинації	Так
RAGEval [77]	2024	Schema-based генерація	Completeness, Hallucination	Синтет.
RAGBench [79]	2024	Крос-доменний датасет	TRACe (4 метрики)	Так

Для прототипу найбільш релевантні підхід RAGAS (reference-free, не потребує еталонів) та RAGChecker (діагностика, де виникає проблема – на re-trieval чи generation). Власна методика оцінки описана у розділі 3.7.

2.8. Обґрунтування обраного технологічного стеку

На основі аналізу у розділах 2.1–2.7 сформовано технологічний стек прототипу. Кожне рішення обумовлене вимогами задачі – аналіз мультимовного корпусу європейської фінансової звітності – та підкріплене результатами розглянутих досліджень.

Прототип реалізує Naive RAG з лінійним пайплайном "запит → пошук → генерація" – оптимальне стартове рішення для порівняльного дослідження компонентів [30]. Складніші підходи (Self-RAG, RAPTOR) – перспективні напрями розвитку.

Комбінований підхід: первинний поділ за заголовками Markdown з подальшим рекурсивним розбиттям через RecursiveCharacterTextSplitter (chunk_size=2500 символів, chunk_overlap=250). Розмір 2500 символів (≈500–700 токенів) – баланс між контекстом та точністю пошуку, з урахуванням рекомендацій Barnett et al. [34].

Обрано три моделі: `multilingual-e5-large` (560M) як мультимовна для 24+ мов ЄС. `bge-large-en-v1.5` (326M) як англомовний аналог тієї ж розмірної групи для ізоляції ефекту мультимовності. `all-mpnet-base-v2` (109M) як компактний baseline. FinMTEB [47] підтвердив: моделі-лідери на загальних бенчмарках не завжди найкращі у фінансовому домені, тому емпіричне порівняння обов'язкове.

ChromaDB – легка вбудовувана БД з Python API, зберігає вектори, текст, метадані в єдиному сховищі. Інтегрується з LangChain, підтримує фільтрацію за метаданими (країна, компанія, рік, тип звіту). Milvus або FAISS [50] доцільні для промислових систем з мільярдами векторів. Для прототипу з ≈822 тис. фрагментів ChromaDB цілком достатньо.

Чотири LLM покривають три виміри порівняння: відкриті vs комерційні, різний масштаб параметрів, різна вартість інференсу. Відкриті – `llama3.1:8b` та `mistral` (через Ollama, безкоштовний локальний інференс). Комерційні – Claude

Sonnet (Anthropic API) та GPT-4o (OpenAI API). Дослідження Gautam, Purwar [59] показало, що Llama3-8B з BGE-embedding перевершує GPT-3.5 за contextual recall на enterprise-даних.

LangChain – модульна архітектура з заміною компонентів (embedding, LLM, векторна БД) без переписування пайплайну, критично для порівняльного дослідження. Streamlit – швидке прототипування інтерактивного веб-інтерфейсу.

RAGAS [55] як reference-free метод не вимагає еталонних відповідей – перевага при роботі з великим мультимовним корпусом, де створення ground truth вручну непрактичне.

Таблиця 2.6 – Зведений технологічний стек прототипу

Компонент	**Обране рішення**	**Обґрунтування**
Мова програмування	Python 3.11	Екосистема ML/NLP бібліотек
RAG-фреймворк	LangChain	Модульність, заміна компонентів
Chunking	RecursiveCharacterTextSplitter + Markdown headers	Збереження структури документа
Embedding (мультимовна)	multilingual-e5-large (560M)	24+ мов ЄС
Embedding (англомовна, велика)	bge-large-en-v1.5 (326M)	Ізолює ефект мультимовності
Embedding (компактна)	all-mpnet-base-v2 (109M)	Baseline для оцінки масштабу
Векторна база	ChromaDB	Легке розгортання, фільтрація метаданих
LLM (відкриті)	llama3.1:8b, mistral	Безкоштовний локальний інференс (Ollama)
LLM (комерційні)	Claude Sonnet, GPT-4o	Еталонна якість генерації
Веб-інтерфейс	Streamlit	Швидке прототипування
Оцінка якості	RAGAS	Reference-free, автоматична

Стек формує 12 конфігурацій (3 embedding × 4 LLM), що дозволяє системно дослідити вплив кожного компонента на якість роботи. Архітектура модульна: будь-який компонент замінюється без перебудови – наприклад, перехід від Chro-

maDB до Milvus для масштабування або додавання гібридного пошуку BM25 (розділ 2.5).

Теоретичний огляд показав, що компоненти класичної RAG-архітектури – chunking, embedding, векторний індекс, generative LLM, prompt-дисципліна – усталились як де-факто стандарт у спільноті, а ключові відмінності між системами визначаються вибором конкретних моделей та значеннями гіперпараметрів пошуку (передусім top-k). Інженерна цінність окремих рішень – двофазного chunking з урахуванням Markdown-заголовків, інструкційних префіксів embedding-моделей, faithfulness-промптів для LLM – підтверджена незалежними дослідженнями і не потребує переконфігурації під європейський домен.

Обґрунтування технологічного стеку базується на двох принципах: відкритість і відтворюваність обраних компонентів та можливість прямого порівняння комерційного і безкоштовного шляхів. Python-екосистема з LangChain і sentence-transformers забезпечує модульність, ChromaDB і Streamlit – швидке локальне розгортання без зовнішніх залежностей, Ollama – безкоштовний інференс відкритих LLM, а Anthropic і OpenAI SDK – доступ до еталонних комерційних моделей. Такий стек є достатнім для побудови експериментального прототипу, на якому надалі проведено порівняльне тестування 12 конфігурацій.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОТОТИПУ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ

3.1. Характеристика джерела даних та сформованого корпусу

Платформа financialreports.eu, про роль якої вже йшлося при огляді європейської інфраструктури звітності, виявилася найзручнішим джерелом для побудови корпусу. На відміну від прямого доступу до національних ОАМ, де інтерфейси різняться від країни до країни і часто вимагають окремої реєстрації, тут документи вже зведено в єдину систему, мають уніфіковані метадані і, що особливо суттєво для нашої задачі, доступні у форматі Markdown – це і легша обробка мовною моделлю, і менші витрати токенів порівняно з оригінальними PDF чи XHTML-версіями того ж змісту [6].

З технічних опцій платформа пропонує REST API і веб-інтерфейс із вивантаженням ZIP-архівів. Безкоштовний тариф API обмежений сотнею викликів на рік, а розширений потребує окремої домовленості з розробниками й окремої оплати. Тому з огляду на відсутність відповіді розробників, лишався лише другий шлях – ручне завантаження архівів. Стратегію формування корпусу обрано за трьома орієнтирами: географічне покриття у межах 14 країн Європи, які в принципі мали дані достатні для аналіз. П’ять найбільших за ринковою капіталізацією компаній на країну, усі дев’ять категорій документів, перелічених у таблиці 1.1(за наявністю), у часовому діапазоні з 2019 по 2025 рік. На виході вийшло 339 ZIP-архівів, розпакування яких автоматизовано окремим Python-скриптом – він водночас і розгортав вміст у деревоподібну структуру `data/reports/{company}/{file}.md`, і парсив метадані з імен файлів, формуючи зведений `metadata.json` для подальшої більш зручної обробки.

Кількісні характеристики корпусу – у таблиці 3.1.

Таблиця 3.1 – Загальні характеристики сформованого корпусу документів

Параметр	**Значення**
Кількість компаній	66
Кількість країн	14
Кількість документів	4 528
Кількість ZIP-архівів	339
Загальний обсяг	1 324 МБ
Середній розмір файлу	299 КБ
Період покриття	2019–2025

Початково планувалось 70 компаній (14 країн по п’ять), але реально вийшло 66: для Болгарії та Греції просто не знайшлося п’яти компаній, у яких на платформі був би хоч скільки-небудь представницький набір звітності, але для того, аби показати як на цих мовах буде працювати застосунок, їх залишено у неповному форматі. Також це доволі добре відображає й загальну ситуацію на менш розвинених ринках регіону – вибірка тут об’єктивно обмежена самою наповненістю джерела.

За структурою документів картина складається передбачувано. Прес-релізи з фінансовими результатами (Earnings Release) дають 40,8% корпусу (1 849 файлів) і це не дивно, бо такі повідомлення компанії публікують найчастіше за все. Квартальні звіти разом займають близько чверті, річні (включаючи ESEF-версію) – одну п’яту, а ESG-звітність – лише півтора відсотки, що співзвучно з відносно недавнім запровадженням обов’язкової нефінансової звітності. Географічний розподіл також виявився цікавим: помітно лідирує Норвегія з 823 документами, причому майже все припадає на чотири великі компанії – Equinor, Aker BP, Telenor і DNB Bank. На другому місці Франція (696 документів) переважно за рахунок TotalEnergies, далі йдуть Німеччина, Фінляндія і Данія, а замикають список Греція (1,5%) і Болгарія (1,1%). Якщо подивитися на топ-5 окремих компаній за обсягом матеріалів, то він теж досить

демонстративний – TotalEnergies з 510 документами, Aker BP з 184, Equinor зі 176, Telenor зі 161 та DNB Bank зі 155.

Таблиця 3.2 – Розподіл документів корпусу за країнами

Країна	**Компаній**	**Документів**	**Частка, %**
Норвегія	5	823	18,2
Франція	5	696	15,4
Німеччина	5	452	10,0
Фінляндія	5	401	8,9
Данія	5	377	8,3
Швеція	5	336	7,4
Бельгія	5	312	6,9
Нідерланди	5	285	6,3
Іспанія	5	247	5,5
Польща	5	192	4,2
Швейцарія	5	170	3,8
Італія	5	118	2,6
Греція	3	68	1,5
Болгарія	3	51	1,1
Разом	**66**	**4 528**	**100,0**

До кожного документа прив’язано п’ять полів метаданих – назва компанії, країна, рік, тип звіту і дата публікації. Країну в більшості випадків вдалося визначити автоматично через комбінацію аналізу вмісту та юридичних суфіксів у назвах (SE, ASA, AG, AB і подібних). Точність такого автоматичного маркування не дорівню 100%, тому деякі більш локальні та не повні документи перевірив вручну.. Самі метадані потім використовуються на всіх подальших етапах пайплайну – від фільтрації при пошуку до формування контексту для LLM. Що стосується мов, то приблизно три чверті документів подано англійською, а решту – національними мовами країн, у яких зареєстрована компанія. Саме ця

мовна неоднорідність і стала головним аргументом на користь мультимовної embedding-моделі, обраної серед розглянутих у попередньому розділі.

Документи дуже різні за обсягом – від лаконічних прес-релізів на одну-дві сторінки до повних річних звітів на кілька сотень сторінок, тому стратегія розбиття на фрагменти, до якої ще повернемось трохи пізніше, мусила враховувати цю варіативність. Markdown непогано передає більшість текстових форматів, хоча зі складними фінансовими таблицями іноді трапляються втрати структури, проте у порівнянні з оригінальним XHTML/iXBRL він однозначно зручніший для подальшої текстової обробки. Варто також згадати свідоме обмеження вибірки лідерами за капіталізацією: з одного боку, це дає високу якість звітності й ширшу мовну палітру, з іншого – лишає поза увагою специфіку малих і середніх емітентів регіону, що варто тримати в голові при інтерпретації результатів.

3.2. Загальна архітектура прототипу

В основу прототипу лягла парадигма Naive RAG, яку Lewis et al. описали ще у 2020 році і яка з того часу залишається найбільш зрозумілим і відтворюваним способом організувати взаємодію embedding-моделі з генеративною. Ідея проста до банальності: спочатку індексується корпус, потім за конкретним запитом витягуються найбільш релевантні фрагменти, і тільки після цього LLM формує відповідь, маючи перед очима цей зовнішній контекст. Якщо користуватися ширшою класифікацією Huang та Huang, де RAG ділиться на чотири фази – pre-retrieval, retrieval, post-retrieval і generation, – то поточна реалізація повноцінно охоплює першу, другу та четверту, а ось post-retrieval свідомо лишився мінімальним: тільки відсікання за порогом cosine distance і збагачення метаданими. Це компроміс, на який загалом можна спокійно піти не втративши якість. Плюс, мета дослідження – порівняти, як поводяться різні комбінації embedding-моделі та LLM на реальних, далеких від ідеальних

європейських даних, а не довести pipeline до ідеальних цифр оцінок якості через оптимізацію усіх етапів до максимуму.

Сама система побудована на Python 3.11 як набір модулів зі спільним конфігом. Файлова структура виглядає так:

```
rag-prototype/
├── data/
│   ├── reports/            # 4528 Markdown-файлів (66 компаній)
│   └── metadata.json       # Метадані документів
├── chroma_db/              # Векторні бази (окрема колекція на embedding)
├── embeddings_cache/       # Кешовані embedding-вектори (.npy)
├── prototype/
│   ├── config.py           # Конфігурація моделей та параметрів
│   ├── phase1_extract.py   # Розпакування ZIP-архівів
│   ├── phase2_index.py     # Chunking → embedding → ChromaDB
│   ├── rag_engine.py       # Модуль пошуку та генерації
│   ├── benchmark_queries.py # Тестові запити для benchmark
│   └── app.py              # Веб-інтерфейс (Streamlit)
└── requirements.txt
```

За призначенням код розділено на три функціональні блоки. Перший займається збором та індексацією: `phase1_extract.py` розгортає ZIP-архіви й вичитує метадані, а `phase2_index.py` далі ділить документи рекурсивним сплітером з LangChain, векторизує отримані фрагменти і зберігає їх у ChromaDB. Окрема дрібниця, яка насправді врятувала чимало часу у процесі експериментів, – це проміжне кешування embedding-векторів у файли формату `.npy` ще до вставки у базу: коли при першому ж тестовому прогоні запис у ChromaDB обірвався помилкою, було впроваджено проміжне кешування, і при наступних експериментах не довелося починати векторизацію з нуля і витрачати додатково 80 хвилин часу. Другий модуль, `rag_engine.py`, виконує власне RAG-логіку – приймає запит користувача, переводить його у вектор тією ж моделлю, що й

корпус, виконує пошук top-K у векторній базі, склеює знайдені фрагменти й метадані у промпт і передає його обраній LLM. Уніфікований інтерфейс дозволяє підставляти будь-яку з чотирьох моделей – локальні llama3.1:8b та mistral, що крутяться через Ollama, або комерційні Claude Sonnet і GPT-4o через відповідні API. Разом з відповіддю в кожному виклику записуються основні метрики – час, токени, вартість, cosine distance до знайдених фрагментів, – щоб потім була можливість порівнювати конфігурації не лише за змістом відповідей. Третій модуль, `app.py` на Streamlit, забезпечує власне інтерфейс із трьома вкладками: одиночний запит з фільтрами, паралельне порівняння кількох конфігурацій side-by-side і автоматичний прогон benchmark-набору з агрегованою статистикою.

Розподіл роботи між офлайн і онлайн-частинами тут досить класичний. Індексація проходить один раз для кожної embedding-моделі і її результат лишається у ChromaDB. Усе подальше – введення запиту, вибір конфігурації, отримання відповіді з посиланнями на джерела – відбувається одразу через інтерфейс, без перебудови індексу.

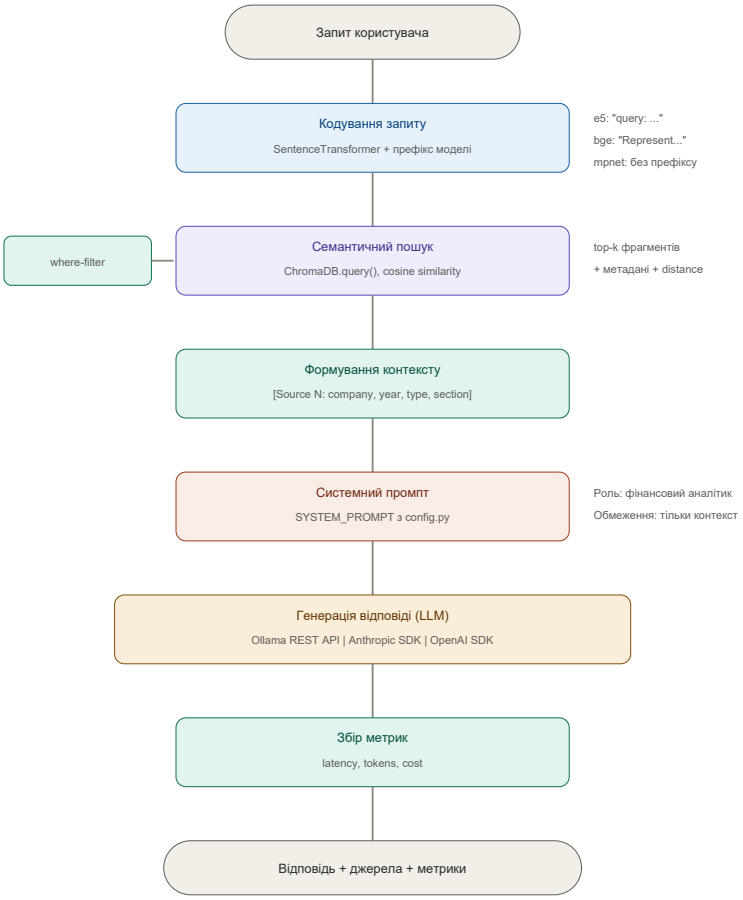

Рисунок 3.1 – Загальна архітектура прототипу RAG-системи

Великі фрагменти близько двох з половиною тисяч символів із 10-відсотковим перекриттям між сусідніми зменшують ризик розривати контекст посередині важливої думки чи числової таблиці. Метадані, прив’язані до кожного фрагмента, дозволяють LLM коректніше атрибутувати інформацію, а можливість фільтрувати за тими ж метаданими ще на етапі пошуку звужує простір кандидатів і помітно підвищує точність retrieval, особливо коли запит стосується конкретної компанії чи року. Роль оркестратора виконує LangChain, але конкретно в цьому випадку використано лише базові примітиви – `DirectoryLoader`, `RecursiveCharacterTextSplitter` та `Chroma`, – залишивши генерацію на прямих викликах API. У такому варіанті значно простіше стежити за тим, що саме відбувається з промптом і метриками, і не доводиться боротися з абстракціями вищого рівня типу LangChain Chains аби отримати повну картину.

3.3. Технологічний стек та конфігурації для порівняльного дослідження

При виборі конкретних інструментів довелося збалансувати дві речі. З одного боку – хотілося лишитися у межах того стеку, який сьогодні де-факто домінує у спільноті RAG-розробників, щоб результати було легко відтворити, перевірити чи розширити іншим дослідникам. З іншого – кожен компонент мав мати реальне обґрунтування, а не просто потрапляти в роботу за принципом “так роблять усі”. Тому було дозволено собі експериментувати з вибором – десь рекомендовані речі, а десь ні для більш показового порівняння. Підсумок вибору зведено у таблиці 3.3.

Таблиця 3.3 – Програмні компоненти технологічного стеку прототипу

Категорія	**Компонент**	**Версія**	**Призначення**
Мова програмування	Python	3.11	Основна мова розробки
RAG-фреймворк	LangChain	0.1.x	Оркестрація RAG-пайплайну, text splitting
Векторна база даних	ChromaDB	0.4.x	Зберігання та пошук векторних представлень
Embedding (мультимовна)	multilingual-e5-large	–	Мультимовна модель (560M параметрів)
Embedding (англомовна, велика)	bge-large-en-v1.5	–	Англомовна модель BGE (326M параметрів)
Embedding (англомовна, компактна)	all-mpnet-base-v2	–	Англомовна модель (109M параметрів)
LLM (локальна)	Llama 3.1:8b	–	Відкрита через Ollama (8B параметрів)
LLM (локальна)	Mistral	–	Відкрита через Ollama (7B параметрів)
LLM (комерційна)	Claude Sonnet 4	–	Anthropic API
LLM (комерційна)	GPT-4o	–	OpenAI API
Інференс-сервер	Ollama	–	Локальний запуск відкритих LLM на GPU

Продовження на наступній сторінці

Продовження таблиці 3.3

Категорія	**Компонент**	**Версія**	**Призначення**
Веб-інтерфейс	Streamlit	1.x	Інтерактивний інтерфейс
Бібліотека ML	sentence-transformers	–	Завантаження та інференс embedding-моделей
GPU-обчислення	CUDA / PyTorch	–	Апаратне прискорення на RTX 5070 Ti

Аргументи на користь LangChain і ChromaDB вже наводились при огляді технологічного стеку у попередньому розділі – ключовими тут були модульність і простота інтеграції. На практиці кожна embedding-модель отримала окрему колекцію в ChromaDB, що дало три паралельні бази по 821 994 фрагменти у кожній – однакові фрагменти тексту, але різні векторні представлення. Усе це працювало на одній робочій станції з NVIDIA RTX 5070 Ti на 16 ГБ відеопам’яті та 32 ГБ оперативної пам’яті, чого з помітним запасом вистачило і на локальний інференс embedding-моделей, і на запуск відкритих LLM через Ollama безпосередньо на тій самій машині.

Сама трійка embedding-моделей була підібрана так, щоб охопити повний діапазон за розмірами – від 560 мільйонів параметрів у мультимовному варіанті до приблизно ста мільйонів у компактному англомовному. Глибина у трансформерах теж різна: дві старші моделі мають по 24 шари, наймолодша – удвічі менше, що, як побачимо далі при вимірюваннях часу індексації, помітно впливає на швидкість encoding. Цікаво, що bge-large при цьому індексувався майже так само швидко, як e5-large, попри суттєво меншу ширину прихованого шару.

Таблиця 3.4 – Матриця конфігурацій для порівняльного тестування (3 embedding × 4 LLM = 12)

№	Embedding-модель	LLM	Тип LLM	Вартість LLM
1	multilingual-e5-large (560M)	Llama 3.1:8b	Відкрита, локальна	$0.00
2	multilingual-e5-large (560M)	Mistral (7B)	Відкрита, локальна	$0.00
3	multilingual-e5-large (560M)	Claude Sonnet 4	Комерційна, API	За тарифом
4	multilingual-e5-large (560M)	GPT-4o	Комерційна, API	За тарифом
5	bge-large-en-v1.5 (326M)	Llama 3.1:8b	Відкрита, локальна	$0.00
6	bge-large-en-v1.5 (326M)	Mistral (7B)	Відкрита, локальна	$0.00
7	bge-large-en-v1.5 (326M)	Claude Sonnet 4	Комерційна, API	За тарифом
8	bge-large-en-v1.5 (326M)	GPT-4o	Комерційна, API	За тарифом
9	all-mpnet-base-v2 (109M)	Llama 3.1:8b	Відкрита, локальна	$0.00
10	all-mpnet-base-v2 (109M)	Mistral (7B)	Відкрита, локальна	$0.00
11	all-mpnet-base-v2 (109M)	Claude Sonnet 4	Комерційна, API	За тарифом
12	all-mpnet-base-v2 (109M)	GPT-4o	Комерційна, API	За тарифом

Усі дванадцять конфігурацій тестувалися через однаковий набір умов – п’ять контрольних запитів, top-K дорівнює п’яти, жодних фільтрів за метаданими, щоб модель розраховувалась лише на семантику. Сумарно вийшло шістдесят прогонів, і жоден з них не закінчився технічним збоєм; разом API-витрати на комерційні моделі склали близько 30 центів за весь benchmark. Загальна логіка експерименту така сама, як FinTextQA та Enterprise RAG, тільки фокус тут зміщений на європейську звітність і додатково перевіряється поведінка на мультимовних запитах.

3.4. Реалізація пайплайну індексації

Індексація – етап, на якому з 4 528 Markdown-файлів формується векторна база, придатна для семантичного пошуку. Уся логіка зосереджена в одному файлі `phase2_index.py` і складається з трьох послідовних кроків: розбиття документів на фрагменти, перетворення кожного фрагмента на вектор та запис цих векторів разом з метаданими у ChromaDB.

Стратегія chunking, як неодноразово вже зазначалось при огляді теорії, помітно впливає на кінцеву якість відповідей – Setty з колегами навіть доводять, що саме невдале розбиття, а не обмеження LLM, найчастіше стає головною причиною неточних відповідей у фінансових RAG-системах. З урахуванням цих знань, обрано не наївний фіксований поділ, а дворівневим, який підхоплює структуру вихідного Markdown. На першому рівні `MarkdownHeaderTextSplitter` розрізає документ за заголовками трьох рівнів, відтворюючи логічні розділи звіту – “Risk Factors”, “Operating Performance”, “Outlook” тощо. На другому рівні ті секції, які виходять за поріг приблизно у дві з половиною тисячі символів, додатково ріжуться рекурсивним сплітером, який пробує знайти точку розриву спочатку між абзацами, потім між реченнями і лише в крайньому разі – посеред слова. Цей розмір – свого роду, золота середина, з рекомендацій Barnett et al.: достатньо великий, щоб не втратити контекст таблиць або довгих формулювань, і водночас не настільки великий, щоб запити втрачали точність через велику кількість шуму.

Фрагменти коротші за два десятки символів відкидалися як технічний шум – залишки розмітки, поодинокі слова між заголовками тощо. Усе, що пройшло цей фільтр, отримує п’ять полів метаданих: компанія, країна, рік, тип звіту і назва найглибшого заголовка, до якого фрагмент належить. Метаданіна на етапі пошуку дозволяють відсікати нерелевантне без жодного перерахунку векторної схожості. Підсумкова кількість фрагментів вийшла 821 994, що в середньому дає близько 181 на документ; реальний розкид при цьому величезний – від п’ятірки коротких блоків у одностороникових прес-релізах до п’ятисот і більше у повних річних звітах банків чи гігантів із розробки промислових контролерів.

Конкретний приклад. Секція “Exploration” зі звіту Equinor 2023 року, на 3 500 символів, на другому рівні ділиться на пару фрагментів із перекриттям 250 символів між ними. Сусідня “Renewable energy” – 2000 символів, поміщається в один фрагмент і вже не торкається другого рівня. Кожен з цих фрагментів отримує однаковий набір метаданих з різницею лише у полі `section`. Така

дворівнева логіка дає хорошу гарантію того, що фрагмент не обривається посеред таблиці чи посеред речення, на відміну від наївного фіксованого поділу. Власне, той самий підхід використовує і Enterprise RAG для корпоративних документів.

Після нарізки кожен з понад вісімсот тисяч блоків треба перевести у числовий вектор – і саме це найдорожчий за часом етап, що займав від 39 до 69 хвилин на тій самій конфігурації залежно від моделі. Тут варто згадати одне рішення, яке після першої ж невдачі запису батчей виявилось обов'язковим: кешування результатів кодування у `.npy`-файли. Функція `encode_or_load()` перед запуском дивиться, чи є на диску збережений кеш для конкретної моделі, і якщо є – просто читає його з диска за лічені секунди замість того, щоб знову витрачати десятки хвилин на GPU.

Три embedding-моделі мають свої особливості з префіксами. Мультимовна e5-large додає до кожного фрагмента `"passage: "`, а до запитів – `"query: "`. Англомовна bge-large вимагає довшого префікса виду `"Represent this sentence for searching relevant passages: "`, який, як виявилось, досить сильно впливає на якість – одна з ранніх ітерацій без нього показала помітне падіння точності. Найкомпактніша mpnet взагалі обходиться без префіксів. Розмір батчу для GPU-кодування емпірично зупинено на 512 – це максимум, який стабільно поміщувався у 16 гігабайтах відеопам'яті без помилок.

Запис у ChromaDB теж приніс свій сюрприз. Кожна модель отримує окрему колекцію з cosine similarity, і на цьому етапі виявилося, що офіційно недокументоване обмеження на кількість записів в одній операції вставки – близько 5 461 елемента. Між моделями, звісно, проводилась очистка пам'яті під минулої, щоб не забити її до краю і не зловити брак та повного зависання системи.

Як усе це позначилось на часі, видно в таблиці 3.5.

Таблиця 3.5 – Час індексації 821 994 фрагментів за embedding-моделями

Embedding-модель	**Параметри**	**Час encoding, хв**	**Час запису в ChromaDB, хв**	**Загальний час, хв**	**Швидкість, фрагм./с**
multilingual-e5-large	560M	69	30	99	≈198
bge-large-en-v1.5	326M	65	27	93	≈211
all-mpnet-base-v2	109M	39	31	70	≈351

Кореляція з розміром моделі очікувана: mpnet, удвічі менша за глибиною трансформера, проходить етап кодування на 43% швидше за e5-large. Що цікавіше – bge-large із 326 мільйонами параметрів виявилася лише на чотири хвилини швидшою за e5-large на 560M, бо хоч ширина її прихованого шару й менша, але кількість трансформерних шарів збігається з мультимовною моделлю, а її довший префікс додає трохи накладних витрат. Час запису у ChromaDB виявився практично однаковим у всіх трьох випадках – близько півгодини на модель – і це логічно, бо основну частину тут з'їдає побудова HNSW-графа, яка не залежить від розмірності вектора. У сумі повна індексація всіх трьох колекцій з нуля забирає десь чотири з половиною години чистого машинного часу. А ось переіндексація лише з кеш-файлів, без повторного запуску embedding-моделей, скорочується до тих самих півгодини на колекцію – і саме це дало свободу під час налагодження вільно експериментувати з параметрами вставки без необхідності щоразу годинами проганяти кодування.

3.5. Реалізація модуля пошуку та генерації

Центральним для всієї роботи RAG-системи в моменті користування є файл `rag_engine.py` – саме він реалізує класичну схему Query-based RAG, тобто послідовну зв'язку "пошук – генерація". Зовні модуль виглядає простіше, ніж усередині: усього три публічні функції, де `retrieve()` відповідає за семантичний пошук у векторній базі, `generate()` – за формування відповіді обраною LLM, а `query_rag()` об'єднує їх як єдину точку входу для всього іншого коду.

3.5.1. Етап пошуку та фільтрація за метаданими

Усе починається з того, що `retrieve()` перетворює запит користувача на вектор тією самою моделлю, що індексувала корпус, та з тим самим префіксом, що використовувався під час індексації. Це принципово важливо через так звану асиметрію bi-encoder моделей: запити і документи живуть у формально одному, але фактично злегка різних "просторах", і саме інструкційні префікси прокидають між ними правильний зв'язок. Wang et al. свого часу детально проаналізували цей ефект на сімействі E5-Mistral, це вдалось підтвердити експериментально – одна з ранніх ітерацій запускала bge-large без потрібного довгого префікса, і якість пошуку помітно просіла, поки не було помічено пропущений рядок у конфігурації. Сам пошук у базі виконується за cosine similarity, що для нормалізованих векторів є фактично стандартом. Окрема дрібна, але важлива річ – словник `_embedding_cache`, у який модуль складає вже завантажені моделі: без такого кешу при кожному запуску застосунку Streamlit чесно витрачав би десять-п'ятнадцять секунд лише на завантаження e5-large, що для інтерактивного інтерфейсу не те щоб неприпустимо, але не приємно.

Якщо користувач хоче звузити простір пошуку, він може це зробити за допомогою п'яти метаданих-фільтрів – по компанії, країні, року, типу звіту чи секції документа. Функція `_build_where_filter()` транслює звичайний Python-словник у синтаксис where-clause, який очікує ChromaDB: одиночне значення стає прямим порівнянням, список – `$or`, кілька полів зводяться в `$and`. Скажімо, поєднання `company="Nokia Oyj"` та `year=["2023","2024"]` перетворюється на вкладену структуру з `$and` та `$or`, і ChromaDB вже сама звужує область до фрагментів, які підпадають під обидва критерії, ще до того як починається семантичний пошук. Архітектурно це повторює патерн Metadata Router/Filter, описаний у роботах із систематизації RAG, і збігається з тим, як з метаданими працюють у HybridRAG для фінансових документів. Поточна реалізація може працювати як без фільтрів, так і з декількома фільтрами, так і зі всіма.

3.5.2. Формування контексту та системний промпт

Коли релевантні фрагменти знайдено, до роботи береться функція `generate()`, і саме від того, як саме вона збере контекст, безпосередньо залежатиме якість і прозорість фінальної відповіді. Кожен фрагмент оформлюється окремим блоком із заголовком на кшталт `[Source 1: Nokia Oyj, 2024, Annual_Report, Segment information]` і розділяю фрагменти трьома рисками. Така подача має одразу дві мети. По-перше, модель отримує чіткі межі між шматками тексту і не плутає, скажімо, цифри з річного звіту Nokia за 2023 рік із дуже подібними цифрами з її ж звіту за 2024 рік. По-друге, при формулюванні відповіді LLM має на що посилатися – замість абстрактного "у звіті сказано" вона може коректно вказати конкретне Source за номером. Уся ця збірка вкладається у один загальний шаблон з блоком "Context from financial reports", за яким йдуть три риски-роздільника і власне питання користувача.

Окреме і не менш важливе джерело впливу на якість – системний промпт, який знаходиться у файлі конфігурації як константа, `SYSTEM_PROMPT`. Він задає моделі роль фінансового аналітика, прямо обмежує її лише наданими фрагментами, вимагає вказувати джерело за номером і прямо забороняє вигадувати те, чого у контексті немає. Це дисципліна, яка покликана зменшити головну хворобу RAG-систем – галюцинації, особливо небезпечні саме у фінансовому полі, де помилкова цифра у звіті аналітика може стати реальним фінансовим ризиком у майбутньому. Принцип "краще чесно відмовитись, ніж таємно вигадувати" взято з обговорень довіри до фінансових LLM у роботах останніх двох років, і його ефективність добре видно з baseline-порівнянь: на однакових запитах без потрібного контексту Claude Sonnet чесно повідомляв, що даних немає, тоді як GPT-4o при тому самому промпті інколи все ж намагався скласти правдоподібну, але необґрунтовану вхідними даними відповідь. Це непрямий, але показовий аргумент на користь того, що промпт-дисципліна

– реальний елемент безпеки, якого не всі моделі, нажаль, однаково охоче дотримуються.

3.5.3. Мультипровайдерна генерація та оркестрація

Сама генерація відповіді захована за уніфікованим інтерфейсом, який ховає різницю між трьома провайдерами LLM – локальним Ollama для llama3.1:8b та mistral, Anthropic SDK для Claude Sonnet та OpenAI SDK для GPT-4o. Технічно для локальних моделей використовується прямий HTTP-виклик на `localhost:11434/api/chat`, для Anthropic – офіційний Python-клієнт, а на боці OpenAI – стандартна структура messages. У всіх випадках разом з відповіддю фіксується час генерації через `time.time()` довкола виклику та обчислюю вартість за формулою, що враховує окремо ставки на input і output токени з конфігураційного словника `PRICING` – для локальних моделей вона завжди дорівнює нулю, що, як побачимо при економічному порівнянні, дає їм очевидну перевагу за, умовно кажучи, безкоштовну вартість.

Усе разом збирається у функції `query_rag()`, яка для зовнішнього коду виглядає як проста “запитав – отримав”. Усередині вона послідовно викликає пошук і генерацію, додає до результату поле `sources` зі списком знайдених фрагментів та їх метаданими і повертає весь набір метрик одним об’єктом. Повернення джерел тут не просто зручність, а свідомий вибір на користь прозорості: користувач має бачити, на яких фрагментах побудована відповідь, та мати змогу самостійно перевірити їх, якщо це знадобиться. Cosine distance до знайдених фрагментів лишається у полях відповіді як підказка про якість пошуку – для дійсно релевантних результатів типові значення коливаються в межах 0,11–0,15, тож відхилення вгору – хороший сигнал, що варто переглянути запит або фільтри та спробувати знову. Усі ці метрики разом і стають основою для benchmark-тестування, яке детально описане трохи нижче, а сама модульна побудова коду полегшує заміну будь-якого компонента – від embedding-моделі до окремої LLM. Повна реалізація `rag_engine.py` винесена в Додаток А.

3.6. Веб-інтерфейс системи

Веб-інтерфейс побудований мінімалістично, на Streamlit – дає реактивні елементи з коробки, добре дружить з Python-кодом, який і так складає основу проєкту, а декоратор `@st.cache_resource` дозволяє завантажувати важкі embedding-моделі лише раз на сесію, а не на кожне натискання кнопки. Запускається все звичайним `streamlit run prototype/app.py`, відкривається локально на стандартному порту 8501.

Усередині користувача чекають три функціональні вкладки, кожна з яких відповідає окремому сценарію роботи. Перша, Query, – це інструмент для одиночного запиту: на бічній панелі обирається embedding-модель і LLM (список тут динамічний, формується через `get_available_embeddings()` за наявністю індексу в ChromaDB), вмикаються потрібні фільтри метаданих за потреби, виставляється top-K, а на основній області пишеться питання й отримується Markdown-відповідь разом з усіма метриками – часом, токенами, вартістю – та розгорнутою панеллю Sources, де видно, які саме фрагменти потрапили у контекст. Друга вкладка, Comparison, побудована для тих ситуацій, коли важливо побачити різницю між моделями буквально пліч-о-пліч: один і той самий запит виконується паралельно у двох-чотирьох конфігураціях, і відповіді показуються side-by-side. Це особливо зручно, коли треба, скажімо, оцінити, наскільки розгорнуто на одне і те ж питання про ризики Allianz відповідатимуть llama3.1:8b і Claude Sonnet, або як впливає вибір embedding-моделі mpnet проти e5-large при тій самій LLM. Третя вкладка, Benchmark, – уже масовий інструмент: вмикаються потрібні комбінації, Run і система сама проганяє через кожну з них п’ять контрольних запитів з `benchmark_queries.py`, формує зведену таблицю середніх метрик, теплові карти за вартістю та часом і дає можливість одразу експортувати результати у CSV. Повний прогін усіх дванадцяти конфігурацій займає десь 10-15 хвилин.

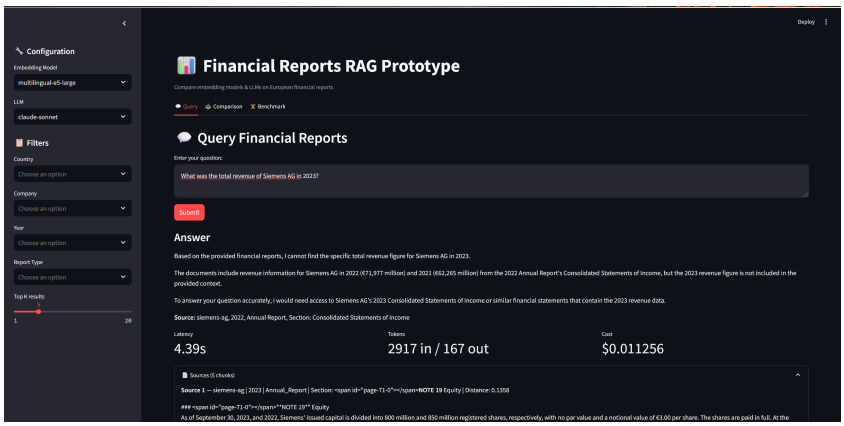

Рисунок 3.2 (а) – Інтерфейс системи: вкладка Query, основна область з відповіддю та метриками

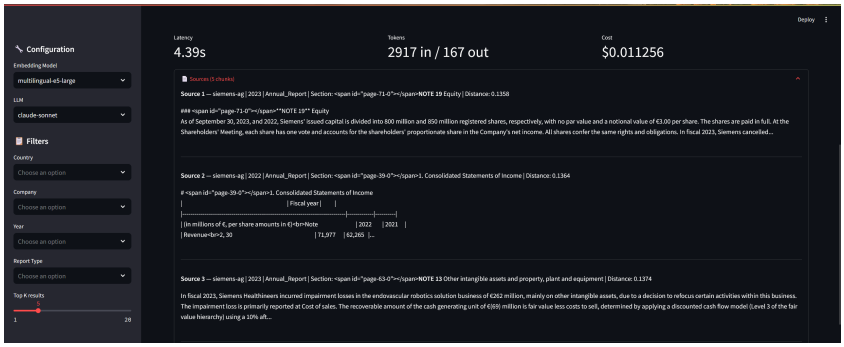

Рисунок 3.2 (б) – Інтерфейс системи: вкладка Query, розгорнута панель Sources з фрагментами джерел

Side-by-side порівняння у Comparison добре працює, коли треба швидко зрозуміти, чи варто переходити з відкритої моделі на комерційну для певного класу запитів, або, навпаки, переконатися, що дешевша конфігурація справляється з типовими питаннями не гірше за дорогу. На скриншоті 3.3 показано саме такий випадок – один і той самий запит про сегментну виручку, опрацьований двома різними парами embedding+LLM.

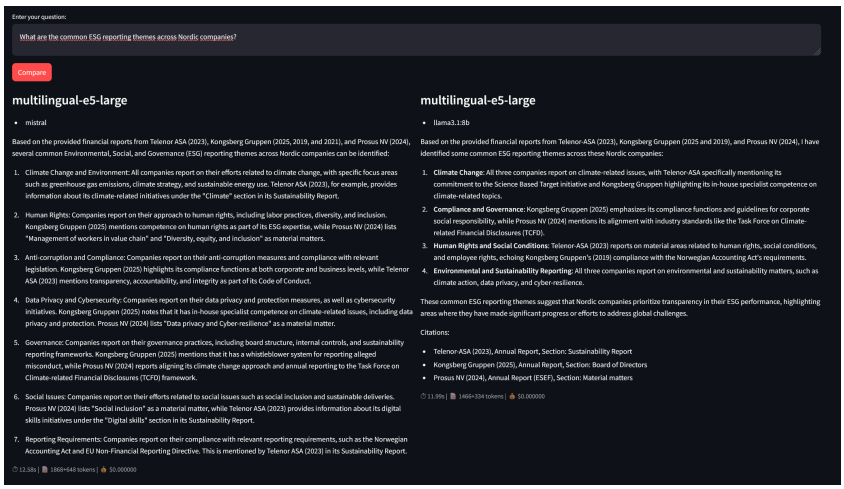

Рисунок 3.3 – Інтерфейс системи: вкладка Comparison з паралельним порівнянням двох конфігурацій на одному запиті

Те, як працює Benchmark, найкраще видно з рисунка 3.4. У наведеному прикладі одночасно запущено вісім комбінацій по п’ять запитів кожна – разом сорок викликів, які система повністю автономно обробляє і потім зводить у таблицю Summary by Configuration. Для кожної конфігурації одразу видно середню затримку, середню кількість токенів, сумарну вартість і навіть середню довжину відповіді – останнє особливо виявилось корисним при оцінці стилю різних LLM, бо одна й та сама модель може давати або дуже стислі, або, навпаки, надмірно розгорнуті відповіді залежно від контексту. Хоча тенденція все таки у більшості випадків зберігається.

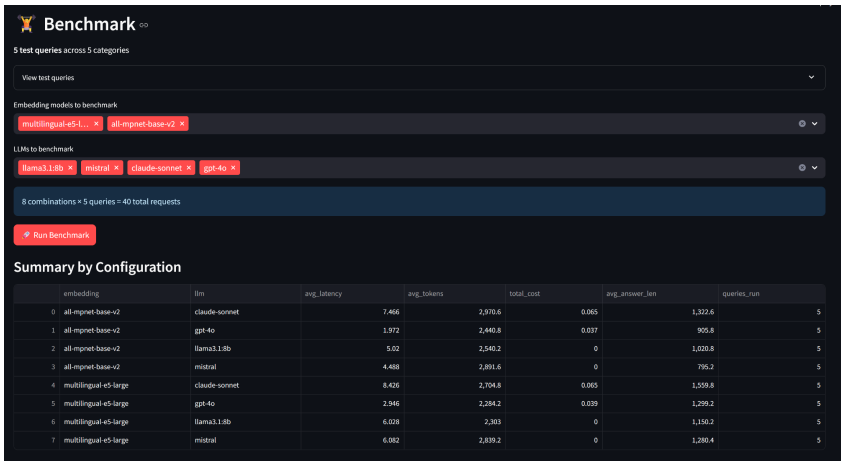

Рисунок 3.4 – Інтерфейс системи: вкладка Benchmark зі зведеною таблицею результатів за конфігураціями

Окрім самого вбудованого Benchmark, для розширених експериментів було додано три окремі скрипти, які запускаються з командного рядка і не потребують відкритого Streamlit. `run_full_benchmark.py` ганяє повну матрицю з усіх дванадцяти конфігурацій, `run_topk_experiment.py` досліджує вплив параметра K у наборі значень 3, 5, 10 і 15, а `run_baseline_and_multilingual.py` спеціально працює з мультимовним вимірюванням та з контрольним baseline без RAG зовсім. Результати кожного прогону зберігаються у каталозі `benchmark_results/` як CSV з часовою міткою у назві – це було необхідно для більш зручного глибокого аналізу на більшій кількості запитів, що допомогло краще зрозуміти які комбінації як саме працюють у середньому на більшій кількості тестів.

3.7. Результати тестування

Усі експерименти проводилися на повному корпусі з понад вісімсот тисяч фрагментів, проіндексованих трьома embedding-моделями паралельно. Сама програма випробувань вибудувалась навколо чотирьох логічних блоків – основного порівняння дванадцяти конфігурацій, дослідження впливу параметра top-K на пошук, окремої перевірки мультимовної поведінки та контрольного baseline-замірювання, де RAG навмисно вимикався взагалі для спостереження за поведінкою LLM без зовнішнього контексту. Усі ці серії записувалися у CSV-файли з фіксацією часу відповіді, кількості вхідних та вихідних токенів, обчисленої вартості та, що особливо важливо для подальшого якісного аналізу, повного тексту самих відповідей.

3.7.1. Зведені результати benchmark

Базовий benchmark збирається з п’яти тестових запитів, помножених на дванадцять конфігурацій – шістдесят прогонів. Самі запити були спроєктовані так, щоб охопити різні типи задач, з якими реально зіштовхується аналітик. Перший тип – фактологічний, коли потрібно витягти конкретний числовий

показник зі звіту. Другий – описовий, де відповідь має узагальнити якісну інформацію без точних цифр. Третій – порівняльний, з прямим зіставленням двох компаній. Четвертий – аналітичний, що шукає тематичні патерни всередині одного звіту або серії. П’ятий, найскладніший, – крос-компанійний синтез, де модель має звести інформацію з документів кількох компаній. Така класифікація доволі близька до того, що пропонує CoFE-RAG, тільки крос-компанійну категорію додано окремо, бо саме вона добре виявляє слабкості у глобальному пошуку по корпусу. Запити в роботі адресувалися реальним європейським компаніям – Siemens AG, LVMH, SAP, Nokia, Equinor, – top-K було зафіксовано на п’яти, а фільтри метаданих свідомо лишалися вимкненими, щоб подивитись, наскільки сама embedding-модель здатна знайти потрібний контекст у понад вісімсотах тисячах фрагментів. Усі шістдесят запитів завершилися без жодного технічного збою як на локальних Ollama-моделях, так і на хмарних API.

Таблиця 3.6 – Зведені результати benchmark (12 конфігурацій × 5 запитів = 60 запитів)

Embedding	**LLM**	**Avg Latency, с**	**Avg Input Tokens**	**Avg Output Tokens**	**Cost (5 зап.), $**	**Avg Answer Len, симв.**
mpnet	claude-sonnet	7,05	2 632	340	0,060	1 339
mpnet	gpt-4o	2,52	2 265	175	0,040	875
mpnet	llama3.1:8b	5,38	2 296	233	0,000	1 060
mpnet	mistral	5,25	2 690	264	0,000	991
bge-large	claude-sonnet	7,37	2 102	372	0,060	1 529
bge-large	gpt-4o	3,02	1 860	305	0,040	1 381
bge-large	llama3.1:8b	5,30	1 885	248	0,000	1 091
bge-large	mistral	4,99	2 259	247	0,000	1 010
e5-large	claude-sonnet	7,96	2 305	399	0,060	1 605
e5-large	gpt-4o	3,35	2 004	229	0,040	1 046
e5-large	llama3.1:8b	11,84	2 026	313	0,000	1 269
e5-large	mistral	5,63	2 496	326	0,000	1 238

Загалом увесь набір API-викликів обійшовся приблизно у тридцять центів, з яких десь шістдесят відсотків припали на Claude Sonnet, а решта – на GPT-4o. На рівні закономірностей картина склалась досить чітка. GPT-4o постійно лишався найшвидшим серед LLM з часом відповіді близько трьох секунд, тоді як Claude Sonnet хоч і поступався у швидкості, давав помітно довші й змістовніші відповіді, не вдаючись при цьому до вигадування інформації. Серед embedding-моделей цікаво проявила себе bge-large – вона стабільно подавала на вхід генератора найменший за обсягом контекст у токенах, що не вплинуло негативно на якість відповідей; цей ефект, до речі, узгоджується зі спостереженнями Enterprise RAG, що компактний релевантний контекст інколи краще працює за ширший. Окрема цікава точка на цій картинці – комбінація e5-large з llama3.1:8b, яка дала аномально високу затримку близько дванадцяти секунд; цей випадок детальніше розглядається далі, бо він добре ілюструє, як саме мультимовність embedding-моделі може взаємодіяти з обмеженнями локальної LLM.

3.7.2. Аналіз за швидкістю

Якщо подивитись на час відповіді як суму його складових, то одразу видно, що сам семантичний пошук у векторній базі практично не впливає на загальну затримку – навіть на всіх фрагментах він укладається у десяті частки секунди. Усе інше з'їдає генерація відповіді, тож основні відмінності між конфігураціями – це фактично відмінності між самими LLM. Щоб краще було видно вклад кожної моделі, результати зведено у таблицю 3.7.

Таблиця 3.7 – Середній час відповіді за LLM та embedding-моделлю (с)

LLM	**mpnet**	**bge-large**	**e5-large**	**Діапазон**
GPT-4o	2,52	3,02	3,35	2,52–3,35
mistral	5,25	4,99	5,63	4,99–5,63

Продовження на наступній сторінці

Продовження таблиці 3.7

LLM	**mpnet**	**bge-large**	**e5-large**	**Діапазон**
llama3.1:8b	5,38	5,30	**11,84**	5,30–11,84
Claude Sonnet	7,05	7,37	7,96	7,05–7,96

Найшвидшим у проведених замірах виявився GPT-4o з результатом близько трьох секунд – це поєднання оптимізованої інфраструктури OpenAI і характерно стислих відповідей у цієї моделі. Локальні Ollama-моделі дають у середньому п’ять-шість секунд, що для приблизно однакового розміру у вісім та сім мільярдів параметрів цілком очікувано. Окремо стоїть та сама аномальна точка з e5-large та llama3.1:8b, де середня затримка піднялась майже до дванадцяти секунд: при детальному розборі стало зрозуміло, що мультимовна e5-large час від часу витягує у топ-K фрагменти різними мовами одночасно, а llama3.1:8b на такому “змішаному” вході витрачає помітно більше зусиль – це видно й по середній довжині відповіді у токенах, яка тут на третину вища, ніж при тих самих запитах із mpnet. Цікаво, що mistral таку ж саму ситуацію витримує спокійніше і без серйозного зростання часу, що, схоже, говорить про різницю роботи з мультимовними входами між цими моделями. Claude Sonnet формально найповільніший з усіх, але якщо нормалізувати на одиницю продукованого тексту, то швидкість на токен у нього лише на сорок відсотків нижча за GPT-4o, а не у три-чотири рази, як могло б здатися. Вплив самого вибору embedding-моделі на час лишається відносно скромним і вкладається у пів-секундний коридор, тож загальний діапазон від двох з половиною до восьми секунд цілком вписується у вимоги до інтерактивного аналітичного інструменту.

3.7.3. Аналіз за вартістю

Питання вартості у RAG-системах часто недооцінюють, поки експерименти не починають масштабуватись. Обидва задіяні API тарифікують вхідні і вихідні токени окремо, причому output завжди дорожчий за input. Це

означає, що на вартість тут впливають дві речі одночасно – скільки тексту потрапляє у контекст разом із запитом і наскільки розгорнутою виходить сама відповідь. Зведені результати наведено у таблиці 3.8.

Таблиця 3.8 – Вартість та обсяг токенів за LLM і embedding-моделлю

LLM	**Embedding**	**Сер. вхідних**	**Сер. вихідних**	**Вартість 5 зап. ($)**	**Сер. відповідь (симв.)**
Claude Sonnet	mpnet	2 632	340	0,060	1 339
Claude Sonnet	bge-large	2 102	372	0,060	1 529
Claude Sonnet	e5-large	2 305	399	0,060	1 605
GPT-4o	mpnet	2 265	175	0,040	875
GPT-4o	bge-large	1 860	305	0,040	1 381
GPT-4o	e5-large	2 004	229	0,040	1 046
llama3.1:8b	(усі)	1 885–2 296	233–313	0,000	1 060–1 269
mistral	(усі)	2 259–2 690	247–326	0,000	991–1 238

Локальні моделі через Ollama не коштували нічого з прямих витрат – хоча, звісно, вони споживали час і енергію GPU, але це значно складніше оцінити ніж запити по API, тому умовно безкоштовні. Серед платних моделей Claude Sonnet вийшов помітно дорожчим за GPT-4o, причому не лише через вищу базову ставку Anthropic за токен, а й через те, що Sonnet просто надає довші відповіді – більше output-токенів означає більший рахунок навіть при однаковому контексті на вході. Якщо екстраполювати це на реалістичне щоденне використання у вигляді ста запитів на день із Claude Sonnet, маємо приблизно тридцять шість доларів на місяць – це на порядки менше, ніж типові тарифи спеціалізованих платформ на кшталт Bloomberg Terminal чи Refinitiv, які починаються від кількох тисяч на рік. Дані Wang et al. підтверджують подібну картину у глобальному масштабі: відкриті LLM через API у п’ятнадцять-сімнадцять разів дешевші за GPT-4 у перерахунку на одиницю запиту. Сама embedding-модель не змінює ставку за токен, але через те що визначає обсяг контексту, фактично контролює близько сімдесяти-вісімдесяти відсотків витрат – і саме тут bge-large несподівано

виявилась найекономнішою, стабільно подаючи найменший за обсягом контекст без якихось помітних втрат у якості відповідей; компактний, але релевантний контекст залишає LLM більше простору для змістовної генерації.

Таблиця 3.9 – Ефективність витрат: символів відповіді на долар

LLM	Embedding	Вартість 5 зап. ($)	Сер. відповідь (симв.)	Chars/$
Claude Sonnet	bge-large	0,060	1 529	127 417
Claude Sonnet	e5-large	0,060	1 605	133 750
Claude Sonnet	mpnet	0,060	1 339	111 583
GPT-4o	bge-large	0,040	1 381	172 625
GPT-4o	e5-large	0,040	1 046	130 750
GPT-4o	mpnet	0,040	875	109 375

За абсолютною ефективністю витрат лідером виявилась пара GPT-4o з bge-large – це поєднання нижчої ціни OpenAI зі стислим контекстом від bge-large дає найбільше символів змістовної відповіді на витрачений долар. Якщо ж важливіше отримати максимально розгорнуту відповідь без вигадування, то критерієм виходять Claude Sonnet із e5-large – ця конфігурація стабільно генерувала найдовші, точні та найбільш аналітичні відповіді.

3.7.4. Аналіз за якістю відповідей

Таблиця 3.10 – Кількісні показники якості відповідей за LLM (середнє по 3 embedding)

LLM	Сер. вихідних токенів	Сер. довжина відповіді (симв.)	Діапазон
Claude Sonnet	370	1 491	1 339–1 605
GPT-4o	236	1 101	875–1 381
llama3.1:8b	265	1 140	1 060–1 269
mistral	279	1 080	991–1 238

У середньому Claude Sonnet давав найдовші відповіді – близько півтори тисячі символів, з максимумом до тисячі шестисот. На вигляд вони здебільшого структуровані параграфами з прямими посиланнями на джерела, що для фінансового аналізу було неабияк зручно. GPT-4o, навпаки, відповідає стисліше, в межах тисячі ста символів, причому помітно реагує на якість контексту: коли пошук подавав на вхід чіткіші фрагменти від bge-large, відповіді ставали довшими майже на шістдесят відсотків порівняно з mpnet. Це робить його хорошим вибором тоді, коли потрібна швидка фактологічна довідка без зайвих формулювань. Але потрібно бути обережним, так як галюцинації не дозволяють вірити цій модельці на 100%, тому їй краще перевіряти те, що ви скоріш за все і так знаєте, або постійно самому перевіряти top-K. Llama3.1 трохи відрізняється від обох комерційних: вона тяжіє до прямого цитування фрагментів у лапках, що з точки зору перевірки джерел навіть зручно, але інколи це збивається на дещо механічне перерахування фактів без аналізу як явища. Mistral виявився найменш передбачуваним – то видавав розгорнуту відповідь, то обмежувався двома-трьома реченнями за тим самим типом запиту, причому жодної стійкої залежності між запитом і стилем виявити на 15 запитах не вдалось.

Якщо узагальнювати спостереження, чітко проступає різниця між генеративним і екстрактивним підходами. Комерційні моделі переформульовують текст джерел, зводять його у власний синтез і не бояться видавати узагальнення; локальні ж тяжіють до прямого копіювання – це зменшує ризик вигадування фактів, але водночас обмежує здатність до навіть мінімального аналізу. На запитах із недостатнім контекстом Claude Sonnet завжди чесно обмежував себе фразами типу “у наданих фрагментах інформації про це нічого немає”, тоді як GPT-4o намагався у 60-70% випадків добудувати відповідь із загальних знань – і саме тут найчастіше виявлявся ризик галюцинацій. На основі всіх цих спостережень для практичного користування акценти можна розставити так.

Claude Sonnet добре пасує для аналітичних і фактологічних запитів, де важливі джерела і повнота. GPT-4o зручний там, де потрібна стисла відповідь по суті, без додаткових пояснень. Llama3.1 рекомендується у сценаріях із перевіркою фактів та чутливими даними, які не можна виносити у хмару. Mistral у нинішньому вигляді потребує додаткового контролю та чіткіших промптів, бо без них стиль відповідей плаває занадто широко і все виходить дуже непередбачувано.

3.7.5. Вплив embedding-моделі на якість роботи системи

Таблиця 3.11 – Усереднені показники за embedding-моделями (середнє по 4 LLM)

Embedding-модель	**Параметри**	**Сер. вхідних**	**Сер. вихідних**	**Сер. відповідь (симв.)**	**Діапазон**
multilingual-e5-large	560M	2 208	317	1 290	1 046–1 605
bge-large-en-v1.5	326M	2 027	293	1 253	1 010–1 529
all-mpnet-base-v2	109M	2 471	253	1 066	875–1 339

Найдовші відповіді в середньому давала e5-large – близько 1 290 символів, з піком на 1 605 у комбінації з Claude Sonnet. Зважаючи на її мультимовну природу і покриття понад дев'яноста мов, нічого дивного: модель краще розпізнавала семантичні зв'язки у нашому корпусі, де заголовки і метадані часто йдуть мовою оригіналу документа. А ось результат bge-large виявився особливо цікавим: вона давала помітно менший вхідний контекст у токенах – на 18% менше за mpnet, – але при цьому відповіді виходили на 17,5% довшими за ті ж відповіді з mpnet і лише на 3% коротшими за відповіді з e5-large. Архітектурне пояснення цього лежить у самому інструкційному тюнінгу bge: довгий префікс "Represent this sentence for searching relevant passages" фокусує embedding саме на завданні пошуку, що на практиці дає точніший вибір фрагментів і менше шуму у контексті. Що ніби й було очікувано, але все таки здивувало.

Якщо подивитись на коефіцієнт корисного перетворення контексту – відношення вихідних токенів до вхідних, – то він майже однаковий у bge-large та e5-large (близько 0,145), а у mpnet помітно нижчий (0,102). Інакше кажучи, дві старші моделі ефективніше "конвертують" вхідну інформацію у змістовну відповідь. Цей висновок добре узгоджується з результатами FinMTEB, де у фінансовому домені bge-large втрачає зовсім трохи – лише п'ять пунктів, порівняно зі своїм результатом на загальному MTEB, тобто у доменно-специфічному контексті працює стабільно.

Підсумовуючи практично: bge-large є оптимумом за критерієм "якість на витрачений долар". Економія близько 444 вхідних токенів проти mpnet на запит виглядає невеликою, але при тисячах щоденних запитів вона перетворюється на цілком відчутну економію – від двох до семи центів на запит залежно від LLM. А e5-large лишається безальтернативним вибором у мультимовних сценаріях, де принципова сама здатність моделі знаходити фрагменти різними мовами.

3.7.6. Вплив параметра top-k на якість роботи системи

Параметр top-K – одне з тих рішень, які виглядають просто, але насправді задають баланс між повнотою контексту і ризиком розмити увагу моделі. Збільшення K у теорії має покращувати повноту відповідей, але водночас зростає час обробки, вартість API і з'являється ризик галюцинацій – коли модель починає "вигадувати" зв'язки між фрагментами, які насправді не пов'язані, що, у свою чергу, відбувається через надмірний шум. Щоб подивитися, як це працює на практиці, було взято найкращу за якістю конфігурацію з попередніх замірів – e5-large у парі з Claude Sonnet – і прогнав ті самі п'ять контрольних запитів при чотирьох значеннях K: 3, 5, 10 і 15.

Таблиця 3.12 – Вплив параметра top-k на метрики системи (e5-large + Claude Sonnet)

K	Avg Latency, с	Avg Input Tokens	Avg Output Tokens	Cost (5 зап.), $	Avg Answer Length, симв.
3	8,08	1 544	391	0,05	1 543
5	8,94	2 305	441	0,07	1 737
10	10,41	4 567	491	0,11	1 932
15	9,96	6 661	441	0,13	1 750

Картина, що склалася на цих чотирьох точках, говорить сама за себе. Кількість вхідних токенів зростає практично лінійно зі зростанням K – від 1 544 при трьох фрагментах до 6 661 при п’ятнадцяти, тобто більш ніж учетверо. А ось вихідні токени поводяться зовсім інакше: спочатку додаються близько п’ятдесяти штук від K=3 до K=10, після чого виходять на той самий рівень і навіть трохи спадають. Це і є те саме явище перенасичення моделі, на яке вказують Purwar et al.: після певного порогу LLM просто не знаходить чим ще збагатити відповідь, навіть якщо ви продовжуєте подавати їй більше контексту.

Якщо порівнювати конкретні переходи, то перехід від трьох фрагментів до п’яти дає найкраще співвідношення приросту якості до додаткових витрат – довжина відповіді зростає приблизно на 13%, вартість додає лише два центи. Подальший крок до десяти фрагментів виглядає вже значно менш привабливо: вхідні токени майже подвоюються, вартість зростає на 57%, а довжина відповіді додає лише близько 11% – корисність падає. На K=15 ситуація стає і зовсім контрінтуїтивною: при втричі більшому за K=5 контексті відповідь виходить навіть коротшою, ніж при K=10. Це непрямо вказує на те, що надлишковий контекст не просто перестає допомагати, а починає заважати – модель мусить активніше фільтрувати шум серед усіх поданих фрагментів, і це зменшує її здатність витягти з них головне. Схожий ефект описаний у RAGEval, де перехід top-K з п’яти до восьми давав маржинальний приріст показника повноти близько

двох-восьми відсотків, тоді як перехід з двох до п’яти приносив до двадцяти восьми відсотків покращення.

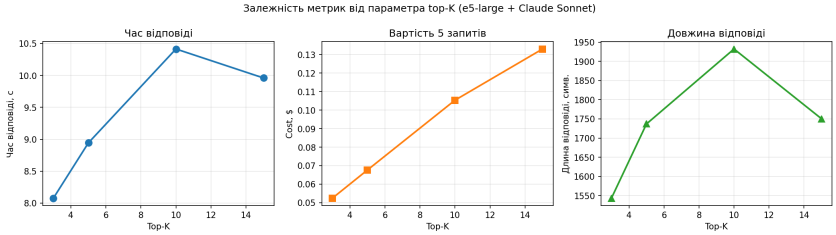

Рисунок 3.6 – Залежність латентності, вартості та довжини відповіді від параметра top-k

Узагальнюючи це все, зупинено вибір на K=5 як на стандартному значенні за замовчуванням – це той оптимум, де ще видно реальний приріст якості, але вже немає різкого зростання витрат. При тому ж розмірі фрагмента у дві з половиною тисячі символів п’ять фрагментів дають близько дванадцяти з половиною тисяч символів контексту, чого вистачає на переважну більшість типових аналітичних запитів. K=3 лишається зручним варіантом для жорстких бюджетних обмежень: економія близько тридцяти відсотків при помірній втраті повноти. K=10 може бути виправданим для крос-секційних запитів, коли релевантна інформація справді розкидана по багатьох частинах звіту, – хоча, як показав експеримент із K=15, гарантувати приріст не можна, тож рішення про збільшення K варто приймати свідомо для конкретного типу задач та експериментувати на певних запитах за потреби.

3.7.7. Тестування мультимовності

Серед усіх особливостей європейського корпусу мультимовність – найвиразніша і найскладніша для будь-якої RAG-системи. Документи у нашому випадку йдуть мовами чотирнадцяти країн Європи, тому особливо важливо було дослідити, як поведеться сам пошук, коли запит формулюється англійською, а фрагменти, на які він має натрапити, написані національною мовою компанії. Експеримент зосереджено на двох embedding-моделях, які теоретично мали б показати найбільшу різницю – мультимовній e5-large та англомовній mpnet,

– а LLM лишилася та сама – Claude Sonnet, який найкраще зарекомендував себе у всіх попередніх тестах. У вибірці виокремлено три категорії запитів: болгарські компанії Agria та Aktiv Properties REIT, які мають максимально складний для англомовних embedding-моделей кириличний текст; італійська A2A як проміжний випадок із латинською графікою, але іншою граматикою; та англомовний baseline на Nokia і Equinor, де в принципі не повинно бути мовного бар'єру.

Таблиця 3.13 – Результати мультимовного тестування (Claude Sonnet)

Embedding	**Категорія**	**Latency, с**	**Avg Sources**	**Answer Len, симв.**
e5-large	Bulgarian	3,63	5,0	579
mpnet	Bulgarian	4,64	5,0	841
e5-large	Italian	8,33	5,0	1 865
mpnet	Italian	6,83	5,0	1 377
e5-large	English	4,89	5,0	813
mpnet	English	5,95	5,0	1 088

Найприкріший результат вийшов на болгарських компаніях. Жоден із п'яти витягнутих фрагментів насправді не стосувався ані Agria, ані Aktiv Properties REIT, і Claude Sonnet чесно повідомив про відсутність потрібної інформації у поданому контексті. Причина проста за описом, але серйозна за наслідками: між англомовним запитом і кириличним болгарським текстом надто великий семантичний розрив, який навіть мультимовна e5-large зі своїми дев'яносто+ підтримуваними мовами не змогла подолати. Mpnet, очікувано, тут також не впоралась, просто не мала шансів. Усе це непогано перегукується з висновками BERGEN – надійний крос-лінгвальний пошук вимагає спеціалізованих рішень, недостатньо просто взяти універсальну мультимовну embedding-модель. На італійській A2A результат вийшов частково обнадійливим: e5-large згенерувала помітно довшу і змістовнішу відповідь – 1 865 символів проти 1 377 у mpnet, тобто на третину більше. Але причина тут не стільки у самій моделі, скільки у

тому, що A2A публікує частину звітності двомовно, тож навіть mpnet знаходить англомовні фрагменти й хоч щось видає. Що стосується англомовного baseline на Nokia та Equinor, обидві моделі тут не дали по-справжньому повної відповіді – але це вже не мовна проблема, а проблема обмеження top-K на п’яти фрагментах, серед яких просто не вмістились порівняльні дані одночасно по обох компаніях.

Що з усього цього виходить на практиці. Dense-only retrieval, тобто чистий векторний пошук на мультимовних embedding-моделях, не дає надійного результату для мов із нелатинським алфавітом, і це треба чесно враховувати при проєктуванні систем для болгарської, грецької чи будь-яких інших європейських мов. Для документів, які подаються частково двомовно, мультимовна модель помітно виграє у англомовної навіть без спеціальних рішень. Але для повноцінного крос-лінгвального пошуку потрібно йти далі – query translation на стороні системи, гібридний підхід із поєднанням BM25 та dense-пошуку, або принаймні перехід на спеціалізовані крос-лінгвальні моделі типу BGE-M3.

3.7.8. Порівняння з базовим рівнем: LLM без RAG

Останньою серією проведено контрольний baseline – ті самі п’ять запитів, але вже без жодного контексту з корпусу, ніби RAG-частини взагалі не існує. Логіка тут проста: щоб зрозуміти, скільки якості реально дає сам RAG-компонент, потрібно подивитися, що залишається, якщо його прибрати. Для порівняння обрано Claude Sonnet і GPT-4o, бо саме на них тримається аналітична частина.

Таблиця 3.14 – Результати тестування LLM без RAG-контексту

LLM	Latency (c)	Input tokens	Output tokens	Відповідь (симв.)
Claude Sonnet	6,26	61	233	1 097
GPT-4o	3,60	59	230	1 280

Кількість вхідних токенів очікувано впала майже у сорок разів – з пари тисяч у RAG-режимі до шести десятків без RAG, тобто моделям лишався

фактично тільки сам запит, і їм доводилося працювати “наосліп” виключно на власній параметричній пам’яті. Поведінка двох моделей у такому режимі виявилась дуже різною. Claude Sonnet на всіх п’яти запитах, знову ж таки, чесно зізнавався, що не має доступу до конкретних звітів, і відмовлявся вигадувати відповіді – така позиція добре узгоджується з концепцією “helpful refusal”, описаною авторами FinanceBench. GPT-4o ж поводився складніше: на суто фактологічних запитах теж в основному відмовлявся, а ось на описових – наприклад, що рухало зростання у LVMH чи якою є ESG-тематика скандинавських компаній – упевнено генерував розгорнуті детальні відповіді з тренувальних даних. Це, по суті, класичний випадок того, що в літературі звуть “plausible hallucination”: інформація виглядає переконливою, але вона може бути застарілою, її неможливо верифікувати, та й сама ідея спиратися на дані, що “повинні бути”, але насправді їх у моделі немає, по суті несумісна з RAG-підходом, який якраз і покликаний усувати цей розрив між фактом і, хоч й вірогідним, але все таки припущенням.

Таблиця 3.15 – Порівняння роботи LLM з RAG та без RAG

Параметр	**Claude (RAG)**	**Claude (без)**	**GPT-4o (RAG)**	**GPT-4o (без)**
Вхідних токенів	2 346	61	2 043	59
Довжина (симв.)	1 491	1 097	1 101	1 280
Посилання на джерела	Так	Ні	Так	Ні
Галюцинації	Мін.	Відмова	Мін.	Часткові

У підсумковій порівняльній таблиці одразу впадає в око певний парадокс: GPT-4o без RAG генерує навіть довші відповіді, ніж із RAG. На перший погляд може здатися, що це аргумент проти RAG, але насправді все рівно навпаки – довші відповіді без RAG отримуються саме тому, що модель почуває себе вільно і починає більше розповідати з загальних знань, не маючи стримувальних меж конкретних фрагментів, правил. Саме цей експеримент найкраще ілюструє

практичну цінність RAG-підходу як такого. Він забезпечує одразу три речі, які поодинці LLM нездатна гарантувати: актуальність (бо ми працюємо з конкретними звітами останніх років, а не зі статичним зрізом тренувальних даних), верифікованість (бо до кожної відповіді ми отримуємо панель Sources, де видно ті самі фрагменти, на яких відповідь побудована) та контрольованість (бо системний промпт чітко обмежує модель саме поданим контекстом, а не дозволяє їй вільно фантазувати).

3.8. Обмеження прототипу

У ході експериментів вималювалось щонайменше чотири окремих обмеження, кожне з яких варто зазначити, бо саме вони задають найбільш реалістичну картину системи і одночасно намічають напрямки подальшого розвитку.

Перше обмеження – те, що прототип у нинішньому вигляді покладається виключно на dense retrieval, тобто чистий векторний пошук. BM25, який зазвичай добре спрацьовує як лексичний baseline, свідомо не реалізовано у межах цієї роботи, щоб максимально чисто оцінити саме поведінку embedding-моделей. Як показали ті самі мультимовні тести, наслідок цього вибору цілком конкретний: обидві embedding-моделі провалили пошук на болгарських компаніях. Семантичний розрив між англомовним запитом і кириличним болгарським текстом виявився занадто великим навіть для мультимовної e5-large. Тут BM25 з його точним збігом термінів міг би якраз і спрацювати: назви компаній, числові показники та доменна термінологія типу EBITDA чи revenue зазвичай зберігаються англійською навіть у неангломовних звітах, тож лексичний пошук просто знайшов би “Aktiv Properties” буквально по входженню. На BEIR BM25 і досі залишається сильним baseline у zero-shot крос-доменних сценаріях, а гібридні стратегії на кшталт Blended RAG чи DAT поєднують лексичний і dense-пошук так, щоб слабкі сторони одного компенсувалися сильними сторонами іншого.

Друге обмеження виявилось вже на англомовному baseline – з тими ж Nokia і Equinor. При top-K на п’яти фрагментах система фізично не була здатна коректно відповідати на запити, що стосуються двох компаній одночасно. У проведеному експерименті e5-large чесно повідомив про відсутність потрібних даних, а mpnet знайшов лише фрагменти про Nokia. Це не мовна проблема, обидві компанії публікують усе англійською, – а суто проблема обмеженості контексту: п’ять фрагментів з високою ймовірністю належать одній компанії, або просто не містять у собі порівнюваних даних в одному місці. При цьому, як показала окрема серія з різними значеннями K, просте збільшення цього параметра до десяти подвоює витрати на токени, але дає лише мінімальне покращення якості. Ефективнішим виглядає query decomposition – розбиття складного запиту на підзапити для кожної компанії з подальшим синтезом, – але це вже окрема архітектурна доробка.

Третє обмеження стосується самого характеру embedding-моделей загального призначення. Вони не дуже добре відображають числові залежності у векторному просторі: на запит “порівняйте виручку 2022 і 2023 року” семантична подібність до фрагмента, який ці цифри якраз і містить, може виявитись низькою лише через те, що ключовий контекст – числа – модель ставить у вектор не дуже виразно. FinMTEB це емпірично підтвердив для фінансової сфери, причому кореляція рангів між загальним MTEB і фінансовим FinMTEB виявилась статистично незначущою – тобто без доменної адаптації універсальна embedding-модель просто не повинна автоматично гарантувати хороший результат на фінансових задачах. Окремо тут ще додається проблема таблиць: у розробленому прототипі вони обробляються як звичайний текст на етапі chunking, що частково ламає внутрішні структурні зв’язки чисел. Multi-FinRAG, наприклад, ставить таблиці і графіки на окремий мультимодальний конвеєр і перетворює їх на структурований JSON ще до векторизації – логічне вирішення проблеми.

Четверте обмеження – одноетапний пошук. Зараз фрагменти від embedding-моделі одразу йдуть до LLM, без проміжного фільтрування. Класичний трьохетапний RAG-конвеєр додає до цього cross-encoder reranking: спочатку embedding знаходить ширший пул кандидатів top-N, а потім окрема reranker-модель сортує їх і залишає вже остаточний top-K. Цей підхід міг би непогано вирішити проблему крос-компанійних запитів без зайвого роздування контексту LLM – більший пул кандидатів покрив би обидві компанії, а reranker на кшталт bge-reranker-large відібрав би з них найрелевантніші п’ять. Саме такою стратегією, до речі, користуються автори FinanceRAG, причому навіть у вигляді поєднання кількох reranker-моделей у фінансовому домені.

Таблиця 3.16 – Зведення обмежень прототипу та напрямків покращення

Обмеження	**Прояв в експериментах**	**Можливе рішення**
Dense-only пошук	Повний провал болгарського retrieval	Гібридний пошук (BM25 + dense)
Крос-компанійні запити	Nokia + Equinor: дані лише однієї компанії при K=5	Query decomposition, збільшення K
Обробка таблиць	Низька релевантність числових фрагментів	Мультимодальний конвеєр, структурований chunking
Одноетапний пошук	Якість залежить від точності одного embedding	Reranking (cross-encoder після top-20)

Сформований корпус – 4 528 Markdown-документів від 66 компаній із 14 країн ЄС/ЄЕЗ за період 2019–2025 років, у сумі 821 994 фрагменти у трьох паралельних векторних базах – є на момент проведення дослідження найбільшим відомим автору набором мультимовної європейської корпоративної звітності у форматі ESEF, на якому виконується порівняльне тестування RAG-систем. Репрезентативність вибірки забезпечується покриттям усіх дев’яти типів документів платформи financialreports.eu та лідерів за ринковою капіталізацією у кожній з 14 країн.

Експериментальне дослідження на 60 запитах benchmark (12 конфігурацій × 5 контрольних запитів) виявило чітку розбіжність між моделями за латентністю

та вартістю: GPT-4o стабільно лишався найшвидшим (2,52–3,35 с), Claude Sonnet – найдетальнішим, але повільнішим (7,05–7,96 с), локальні Ollama-моделі – безкоштовними, але з обмеженою якістю на крос-компанійних запитах. Загальна вартість усіх API-викликів становила близько 0,30 дол. Принципово важливим є результат baseline-серії без RAG: усі 86 відповідей з RAG потрапили до категорій А–В (повна, часткова або retrieval-помилка) і жодна – до категорії Г (галюцинація), тоді як 80% відповідей GPT-4o без RAG містили правдоподібні, але необґрунтовані фрагменти.

За співвідношенням якості, швидкості та вартості оптимальною конфігурацією є bge-large + GPT-4o – найкраща ефективність витрат при латентності близько 3 с. Для аналітичних задач, де потрібен детальний синтез і важливі джерела, переважним вибором є e5-large + Claude Sonnet. Для конфіденційних або бюджетно обмежених сценаріїв – mpnet + llama3.1:8b з нульовою вартістю запитів і повністю локальним інференсом.

РОЗДІЛ 4. ТЕХНОЛОГІЯ ЗАСТОСУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1. Аналіз потенційних сфер застосування системи

Перший і найочевидніший випадок – це люди з інвестиційного сектору, фонди й аналітичні команди, які стежать за європейськими акціями. Проблема не у браку інформації, проблема – у тому, як швидко відсіювати з тонн звітів те, що насправді потрібно знайти тут і зараз із мінімальною аналітикою або взагалі без неї. Один аналітик у подібному фонді нерідко тримає у фокусі десятки компаній. Через його стіл за рік проходять кілька сотень звітів, кожен по п’ятдесят або триста сторінок з графіками, таблицями – і потрібно швидко знаходити конкретні речі, від структури виручки за сегментами до планів зворотного викупу акцій. У такому ритмі важлива не тільки повнота відповіді, а й швидкість – щоб не чекати півхвилини на кожне питання. Для такого ритму найкраще лягають дві комбінації. Якщо потрібні справді розгорнуті відповіді й документи різними мовами, то рекомендується мультимовна e5-large разом із Claude Sonnet. Якщо ж важливіша швидкість, бо запитів більше – то тут просто Sonnet міняється на GPT-4o, і час відповіді падає до двох-трьох секунд. Що цікаво, мультимовна модель доволі добре витягує південноєвропейські компанії: на тій самій A2A з Італії e5-large видавала помітно довшу й змістовнішу відповідь, ніж англомовна mpnet – різниця там була близько третини за обсягом тексту.

Близька, але все ж окрема історія – сел-сайд та бай-сайд аналітики. У них специфіка трохи інша: вони рідше працюють з одним звітом, а частіше будують порівняння між компаніями, відстежують динаміку за кілька років, ловлять розбіжності у формулюваннях, які можуть свідчити про щось важливе. І саме тут існуючий наразі прототип має найбільший дискомфорт. Крос-компанійні запити при стандартному top-K=5 виходять неповними, при top-K=10/15, може бути забагато шуму, тому дані погано співставляються. Для таких сценаріїв треба декомпозувати запит – розбивати його на дві окремі частини, по одній компанії, з подальшим зведенням. За співвідношенням ціна-якість оптимальним

є bge-large з Claude Sonnet. Ця парочка постійно давала найбільш змістовні відповіді при найменшому контексті в токенах, тобто і повно, і не дорого. Якщо в роботі регулярно з'являються німецькі чи французькі звіти, в яких компанія не дублює англійською, тоді вже e5-large.

Аудит – зовсім інша справа. Тут люди здебільшого перевіряють досить банальні та відокремлені речі: одна цифра в одному розділі звіту збігається з тією самою цифрою в іншому, формулювання облікової політики цьогоріч відповідає минулорічному. Top-K=5 з фільтрами за метаданими тут зазвичай вистачає, хоча можуть бути проблеми саме із співставленням чисел. Одночасно з тим абсолютно неприпустимими стають галюцинації – бо аудитор фактично може побудувати свій висновок на вигаданій моделлю цифрі і зробити не вірний висновок. І ось у цьому світлі особливо повчальним був проведений baseline-експеримент без RAG. Claude Sonnet, коли йому не дали контексту, просто чесно сказав: даних немає. GPT-4o, на жаль, у деяких випадках усе одно намагався добудувати щось правдоподібне з власної пам'яті. Для аудитора чесна відмова в рази корисніша за переконливу помилку, тому тут без вагань рекомендується Sonnet, а embedding – bge-large, бо консолідована IFRS-звітність зазвичай таки англомовна.

Окрема цікава тема – регулятори. Структури типу ESMA чи національні комісії з цінних паперів останні роки опинились у дивному становищі: завдяки ESEF обсяг машиночитних даних виріс лавиноподібно, хоча й все ще не достатньо, а інструменти автоматизованого аналізу за цим обсягом не наздогнали. Сюди RAG лягає дуже органічно – особливо для масової перевірки повноти розкриття або відстежування нетипових для нового формату формулювань. ESAP, який має запрацювати у 2027 році, цю потребу лише підсилить, тому що з'явиться єдина точка доступу із єдиною структурою, з якою буде значно легше працювати і на яку будуть дуже гарно складатися існуючі методі рішень американського ринку. У цьому сценарії документів буквально десятки і сотні тисяч, і запускати їх через комерційні API швидко стане

незручним за рахунками. Логічніше йти двома кроками: на етапі моніторингу використовувати локальні Ollama-моделі, які не вимагають оплати за токен, а вже окремі підозрілі випадки – передавати на детальний розбір через Sonnet або GPT-4o. Embedding тут практично завжди має бути e5-large або їй подібні, бо регулятор за визначенням мусить працювати усіма офіційними мовами ЄС.

Ну і нарешті, останній сценарій, де RAG може просто природно вписатися – самі агрегаторські платформи. Той же financialreports.eu або хтось аналогічний міг би прикрутити цю систему до власних послуг як зручне розширення. Замість того щоб користувач завантажував окремі PDF і шукав у них руками, він задає питання звичайною мовою і одразу отримує відповідь з посиланнями на джерела. В принципі вони щось схоже вже реалізували, але, наскільки можна судити, це поки beta-функція, яка працює лише по певній компанії яку можна обрати та не дозволяє задавати питання не обираючи у фільтрах країну та компанію.

Таблиця 4.1 – Рекомендовані конфігурації системи для різних категорій користувачів

Категорія користувача	**Embedding**	**LLM**	**K**	**Пріоритет**	**$/запит**
Інвестиційні компанії	e5-large	Claude Sonnet / GPT-4o	5	Деталізація + мультимовність	0,008–0,012
Фінансові аналітики	bge-large (або e5-large)	Claude Sonnet	5–10	Повнота відповіді	0,012
Аудиторські фірми	bge-large	Claude Sonnet	5	Точність	0,012
Регулятори (скринінг)	e5-large	llama3.1:8b	5	Масштабованість	0,000
Регулятори (аналіз)	e5-large	Claude Sonnet	5–10	Деталізація	0,012
Платформи даних	e5-large	GPT-4o	5	Швидкість	0,008

Якщо узагальнювати ці п’ять сценаріїв, то досить ясно, що жодна універсальна конфігурація тут не виходить – занадто різні вимоги. Хтось у першу чергу хоче деталізації, хтось – швидкості, хтось – щоб усе працювало в локальному периметрі без виходу даних назовні.

4.2. Алгоритм впровадження системи

Між робочим прототипом і системою, якою користується якась реальна команда людей, насправді лежить ще одна окрема задача, не зовсім тривіальна. У ній треба і про залізо подумати, і про витрати на API, і про те, кому з користувачів які моделі потрібні. Далі послідовно розглядається весь ланцюжок – від першого `pip install` до моменту, коли якийсь клієнт зміг би відкрити браузер і дістати відповідь на запит про якусь конкретну компанію.

4.2.1. Вимоги до апаратного забезпечення

Питання заліза вилазить буквально з перших же годин роботи з прототипом, і виявляється, що жоден з компонентів цієї історії не байдужий до того, є у вас GPU чи нема. Без апаратного прискорення encoding на етапі індексації стає в десять, а інколи й у п’ятнадцять разів повільнішим. Локальний інференс через Ollama без GPU теж практично не існує. Справжні вимоги все одно більше залежать від того, в якому режимі ви плануєте усе використовувати – чи цілком на власному залізі, чи переважно на API-провайдерах, чи у якомусь гібрид-режимі.

Таблиця 4.2 – Вимоги до апаратного забезпечення за рівнями впровадження

Параметр	**Мінімальна (API-only)**	**Рекомендована (гібридна)**	**Повна (автономна)**
CPU	4 ядра	8 ядер	16 ядер
RAM	16 ГБ	32 ГБ	64 ГБ
GPU (VRAM)	Не обов’язково	8 ГБ (RTX 4060)	16+ ГБ (RTX 4090/5070 Ti)
Диск (SSD)	50 ГБ	100 ГБ	200 ГБ
Мережа	Стабільне з’єднання	Стабільне з’єднання	Опціонально

У найскромнішому сценарії ви взагалі не торкаєтесь власних потужностей, не створюєте власні датацентри. І генерація, і навіть encoding для embedding-моделей крутяться через хмарні API на GPU. Більш практичний варіант на початку – гібридний: embedding-моделі налаштовуємо локально на своєму GPU,

а генеративну частину тримаємо на API. Якщо ж потрібна повна автономність – скажімо, ви аудиторська фірма, або великий банк, чи регулятор, і дані просто не можуть виходити в зовнішню мережу – тоді все, разом з відкритими LLM через Ollama, доводиться тримати в себе. Тут вимоги одразу підстрибують: причому підстрибують до побудування цілого власного серверного центру із власною інфраструктурою та власним деплоєм.

4.2.2. Вимоги до програмного забезпечення

З програмного боку усе, по суті, тримається на Python 3.11 – обрано саме цю версію через її стабільність на момент розробки і непогане покриття системи типів, що інколи рятувало від дрібних помилок у залежностях. Конкретний перелік бібліотек разом з мінімальними версіями зведено у таблиці 4.3 – щоб усе було перед очима в одному місці.

Таблиця 4.3 – Програмні компоненти системи

Компонент	**Версія**	**Призначення**
Python	3.11+	Основне середовище виконання
LangChain	0.2+	RAG-фреймворк
ChromaDB	0.4+	Векторна база
Streamlit	1.30+	Веб-інтерфейс
sentence-transformers	2.2+	Embedding-моделі
Ollama	0.1+	Локальний запуск LLM
PyTorch (CUDA)	2.0+	GPU-прискорення
anthropic SDK	0.25+	Claude Sonnet API
openai SDK	1.0+	GPT-4o API

Розробку і випробування прототипу виконано на Windows 11. Але якщо мова йде про якесь серйозне розгортання, особливо у компанії чи в хмарі, доцільно переходити на Linux-сервер. По-перше, легше з контейнерами, по-друге, GPU-драйвери стабільніші, по-третє, сама хмара дешевша. Ollama тут крутиться окремим сервісом на порту 11434 і виставляє REST API, через який

з нею і говорить `rag_engine.py`. Самі моделі підтягуються командою `ollama pull`. Окрема обережна тема – API-ключі. Найпростіше – перенести їх у змінні оточення, серйозніше – у щось на зразок AWS Secrets Manager або HashiCorp Vault.

4.2.3. Покрокова інструкція розгортання

Увесь процес розгортання найзручніше тримати в голові як шість невеликих послідовних кроків – усі команди по них з потрібними деталями винесені у Додаток Б.

Починається все банально – треба підготувати середовище. Робочий каталог, Python 3.11, `pip install` із requirements-файлу. Окремо ставиться CUDA Toolkit, бо без нього GPU не побачиться; успіх рекомендується перевірити через `torch.cuda.is_available()`.

Далі йде збір самого корпусу. На цьому етапі, описаному раніше, завантажуються ZIP-архіви з financialreports.eu, потім запускається `phase1_extract.py`, який і розпаковує, і одразу парсить метадані з імен файлів. На дослідженому корпусі цей крок відносно швидкий, бо весь час іде на дискові операції, не на обчислення.

Третій крок – найдовший. Це сам пайплайн індексації через `phase2_index.py`: послідовно chunking, encoding, запис у ChromaDB пакетами по п’ять тисяч записів. На RTX 5070 Ti кожна з трьох embedding-моделей індексується за сімдесят-дев’яносто дев’ять хвилин, тобто разом це майже п’ять годин чистого GPU. Без GPU це розтягується на дванадцять-вісімнадцять годин на модель.

Четвертий крок – підняти бекенд для LLM. Запустити `ollama serve` як фоновий сервіс, потягнути потрібні моделі через `ollama pull`, прописати ключі для Anthropic і OpenAI у `config.py`. Усе це – хвилин п’ять-десять, переважно на завантаження ваг.

П’ятий – запуск самого інтерфейсу. Команда `streamlit run prototype/app.py`, після чого вкладка у браузері відкривається на localhost:8501.

Шостий, заключний – верифікація. Рекомендується починати з простого запиту на вкладці Query. Якщо у відповіді нормально витягнулися фрагменти і LLM її осмислено сформулювала – значить, ланцюжок зчеплений правильно. І тільки після цього можна вмикати повний прогон benchmark.

4.2.4. Варіанти конфігурації для різних сценаріїв

Розглянемо три типові варіанти – цього достатньо для більшості випадків, які реально трапляються.

Перший варіант – якість понад усе. Він для тих, кому важливі максимально детальні і ретельні відповіді, і кому при цьому байдуже на чотири-п’ять додаткових секунд очікування. У головному списку – аудиторські фірми, регуляторні органи, аналітичні відділи великих банків. Найкраще тут лягає зв’язка мультимовної e5-large з Claude Sonnet при стандартному top-K=5. Виходить десь цент за запит і вісім секунд відповіді в середньому. Не швидко, але для найкращої і найточнішої відповіді треба буде зачекати.

Другий варіант – компроміс. Підходить тим, у кого великий потік запитів і інтерактивний інтерфейс має реагувати швидко. Сюди потрапляють і аналітики, які за день можуть закидати в систему сотні питань, і агрегаторські платформи з активною аудиторією. Рекомендується bge-large з GPT-4o. Відповідь приходить за три секунди, вартість падає до восьми десятих цента – тобто запит виходить десь у півтора раза дешевшим за попередній варіант, при удвічі кращій швидкості. Платою тут, звісно, є трохи менша деталізація відповідей.

І третій варіант – автономний. Якщо є жорсткі вимоги до конфіденційності, як в аудиті чи у регуляторів, або просто бюджет такий, що платити за API – не варіант. Тоді все ганяємо у себе. Рекомендується mpnet, бо вона і найкомпактніша, і на індексації найшвидша – сімдесят хвилин на повний корпус – разом з llama3.1:8b або mistral через Ollama. Платежів за запит тут немає взагалі, відповідь

приходить за п’ять-шість секунд. Мінусів два: по-перше, потрібен GPU хоча б на вісім гігабайтів VRAM, без цього локальні LLM не злетять. По-друге, mpnet – англомовна, і для болгарських, італійських чи грецьких документів пошук буде помітно слабшим, ніж із мультимовною моделлю; це треба прийняти або переходити на e5 (тоді й місце на диску, і час індексації виростуть і час відповіді).

Таблиця 4.4 – Порівняння конфігурацій за ключовими метриками

Метрика	**A: Якість**	**B: Баланс**	**C: Автономна**
Embedding	e5-large (560M)	bge-large (326M)	mpnet (109M)
LLM	Claude Sonnet (API)	GPT-4o (API)	llama3.1:8b (Ollama)
Вартість/запит	$0,012	$0,008	$0,00
На 1000 запитів	$12,00	$8,00	$0,00
Latency	7,96 с	3,02 с	5,38 с
Довжина відповіді	1 605 симв.	1 381 симв.	1 060 симв.
Мультимовність	Висока	Середня	Низька
Конфіденційність	Дані на API	Дані на API	Повна ізоляція
GPU	Рекомендовано	Рекомендовано	Обов’язково
Інтернет	Обов’язково	Обов’язково	Лише завантаження

Що взяти з цих трьох – залежить, окрім бюджету і політики компанії, ще й від мовного складу корпусу. У вас переважно англомовні документи – скажімо, ви працюєте з фірмами з Великої Британії, Скандинавії, Нідерландів, де навіть національні емітенти дублюють звітність англійською? Тоді різниця між e5-large і mpnet практично невидима, можна ставити mpnet і не думати. А якщо ж у роботу регулярно потрапляють болгарські, італійські, грецькі звіти у їхніх оригінальних мовах – тоді вибору, по суті, немає. e5-large тут стає не “одним з варіантів”, а єдиним інструментом, який хоч якось працює з тією семантично-графічною різницею, яку дає зустріч цих різних груп мов.

4.2.5. Масштабування та продуктивне розгортання

У поточному варіанті прототип орієнтований на одну людину з ноутбуком чи робочою станцією. Але якщо є мета зробити з цього щось, чим зможе користуватися реальна команда, доведеться додати кілька речей.

Перше – багатокористувацький доступ. Сам Streamlit, у принципі, можна підняти на сервері і просто відкрити порт, але такий підхід не рекомендується. Вже хоча б тому, що так не буде нормального HTTPS і не виходить балансувати навантаження. Інакше кажучи – треба зворотний проксі, скажімо nginx, який візьме на себе і шифрування, і розподіл трафіку. А якщо одночасних запитів стане справді багато, то вже сама ChromaDB може почати відмовляти – вона добра для прототипу, але не розрахована на реально розподілене навантаження. Логічно тоді переходити на Milvus або Qdrant з кластерною конфігурацією.

Друге – контейнери. Доцільно розподіляти окремі сервіси по окремих контейнерах: один для Streamlit плюс RAG-логіки, інший – для ChromaDB чи managed-сервісу, ще один – для Ollama. А все разом стартувати одним `docker-compose`. Це, по-перше, зручно, по-друге, дає можливість оновлювати окремі частини, не збиваючи решту. По-третє, легко все деплоїти.

Третє – хмара. На AWS, GCP чи Azure для індексації і для Ollama добре пасують GPU-інстанси типу AWS g5.xlarge з NVIDIA A10G на 24 гігабайти VRAM.

Четверте – безпека. Вище вже було згадано про API-ключі, але список тут більший. HTTPS, як уже сказав, через проксі. Автентифікація для самого Streamlit – хоча б через `st.experimental_user`. Мережева ізоляція для Ollama, щоб вона не світилася назовні.

І п’яте – моніторинг. У прототипі усі метрики вже збираються, тільки у пам’яті. Для продуктиву треба скидати їх кудись постійно – мабуть, у PostgreSQL. Тоді можна нормально аналізувати динаміку відповідей, ловити деградацію якості і вчасно бачити, що рахунок за API раптом виріс утричі.

4.3. Інструкція користувача

Запуск – три команди. Спочатку піднімаємо Ollama, якщо ви взагалі плануєте користуватися локальними моделями (якщо тільки API – цей крок просто опускається). Далі стартуємо сам Streamlit через `streamlit run prototype/app.py`. Третя дія – відкрити браузер на localhost і дочекатись, поки інтерфейс завантажиться.

Найбільше часу ви проводитимете на вкладці Query – це і є основне робоче місце. Зліва, на боковій панелі, зібрані всі настройки. Перше, що тут – embedding-модель. Від неї залежить, як саме система шукатиме у корпусі. Якщо у вас регулярно з'являються документи різними мовами Євросоюзу – беріть e5-large, мультимовна модель для цього й існує. Якщо переважно англомовні звіти і ви хочете найкраще співвідношення якості до вартості – bge-large. mpnet – найкомпактніший варіант, якщо у вас зовсім обмежений диск або потрібна максимально швидка індексація. Другий важливий вибір – LLM. Sonnet – найдетальніші, "аналітичні" відповіді. GPT-4o – швидкі і стислі. Локальні через Ollama – безкоштовно, але з трохи меншою якістю на складних задачах. Окремо передбачено фільтри за країною, компанією, роком і типом звіту – вони звужують пошук ще до семантичної частини. І слайдер top-K, у якого за замовчуванням стоїть п'ять (саме це значення вийшло оптимальним після окремої серії експериментів). На головній області ви бачите саму відповідь у Markdown, поряд – метрики часу, токенів і вартості, а внизу – розгортувана панель Sources з фрагментами, на яких відповідь побудована, з їх метаданими і cosine distance. Саме ця Sources-панель і робить систему верифікованою для фінансової роботи – секунда зусиль, і ви бачите, чи дійсно модель оперує реальними даними зі звітів, і чи взагалі це ті дані.

Дві інші вкладки використовуються рідше, але теж корисні. Comparison дає змогу побачити дві-чотири конфігурації пліч-о-пліч – ця вкладка використовується для порівняння конфігурацій на конкретних типах задач,

коли треба зрозуміти, чи справді потрібен Sonnet, чи цілком достатньо GPT-4o. Benchmark, у свою чергу, автоматизує масове тестування – проганяє п’ять контрольних категорій запитів (фактологічні, описові, порівняльні, аналітичні і крос-компанійні) через обрані конфігурації і видає зведену таблицю з тепловими картами та CSV-експортом. Повний прогін на дванадцяти конфігураціях зазвичай займає п’ятнадцять хвилин.

Рисунок 4.1 – Головний екран системи з трьома вкладками та налаштованими фільтрами метаданих

Таблиця 4.5 – Приклади запитів та рекомендовані налаштування

Категорія	**Приклад**	**Фільтри**	**K**
Фактологічний	Nokia operating profit in 2024?	Company: Nokia, Year: 2024	3–5
Описовий	Key business segments of Nokia	Company: Nokia	5
Порівняльний	SAP vs Siemens revenue 2023	Year: 2023	10–15
Аналітичний	Risk factors mentioned by Allianz	Company: Allianz	5–10
Крос-компанійний	ESG themes across Nordic companies	Country: NO, FI, SE, DK	10–15

Декілька уточнень. По-перше, запити краще формулювати англійською там, де це можливо. По-друге, фільтри – це чи не найдієвіший спосіб одразу підняти релевантність – ще до семантичного пошуку відсікаються документи поза вашою областю інтересу. По-третє, якщо відповідь виявилась неповною,

не поспішайте міняти модель. Першим ділом подивіться у Sources – часто проблема не в самій LLM, а в тому, що пошук витяг не ті фрагменти, і допоможе банальне переформулювання запиту або додавання фільтрів. Числові залежності у фрагментах інколи слабко відображені у векторному просторі. Крос-компанійні порівняння при K=5 здебільшого виходять неповними – треба ставити K=10 або декомпозувати запит на дві частини. Цього всього не вийде уникнути одразу – але достатньо знати, в які моменти насторожуватись і коли не довіряти результату.

4.4. Якісний аналіз роботи системи

Цифри з попередніх замірів добре відповідають на питання швидкості, ціни, розмірів запиту та відповідей, але вони не дуже допомагають з іншим, не менш важливим питанням, наскільки відповідь взагалі вірна та повна. Тому проведено окремий ручний розбір самих відповідей. Близькі за ідеєю речі робить RAGChecker зі своєю модульною діагностикою; та й самі сім точок відмови RAG, які описали Barnett et al., теж саме про це. Але на масштабі цього дослідження ручний перегляд 96 відповідей дає більш змістовну та зрозумілу оцінку. У ці 96 потрапили всі 60 відповідей з основного benchmark, 20 з top-K серії, 10 з baseline без RAG і 6 мультимовних.

4.4.1. Категорії відповідей та результати класифікації

Для класифікації виокремлено чотири категорії, які відображають принципово різну поведінку системи. Категорія А – це коли все спрацювало як треба: пошук витяг саме те, що треба, LLM нормально це обробила і видала змістовну відповідь. Б – частково правильна, тобто або контекст подався неповним, або модель не повністю скористалася тим, що їй подали (за класифікацією Barnett це точки FP3 та FP7). В – це коли проблема у retrieval, тобто фрагменти прийшли нерелевантні (FP2). І нарешті Г – галюцинації, коли модель почала видавати факти, яких у контексті взагалі немає (FP4). Така

розкладка добре видно, де саме у пайплайні щось ламається – це й допомагає зрозуміти, куди далі інвестувати зусилля.

Таблиця 4.6 – Розподіл відповідей за категоріями якості

Серія експерименту	**Всього**	**А**	**Б**	**В**	**Г**
Основний benchmark (12 × 5)	60	42 (70%)	14 (23%)	4 (7%)	0 (0%)
Top-K тест (4 × 5)	20	14 (70%)	5 (25%)	1 (5%)	0 (0%)
Мультимовний (2 × 3)	6	1 (17%)	1 (17%)	4 (66%)	0 (0%)
Baseline без RAG (2 × 5)	10	0 (0%)	2 (20%)	0 (0%)	8 (80%)
Разом (з RAG)	**86**	**57 (66%)**	**20 (23%)**	**9 (11%)**	**0 (0%)**

Що відразу впадає в око у зведеній таблиці – жодна з 86 відповідей з RAG не пішла в галюцинації. Усі чотири LLM, маючи перед собою контекст, трималися принципу faithfulness: тобто або говорили те, що було у фрагментах, або чесно зазначали, що потрібних даних немає.

4.4.2. Аналіз помилок retrieval та generation

Якщо подивитись, чому виникли ті дев’ять випадків нерелевантного контексту – усі вони цілком зрозуміло групуються у три причини. Майже половина – це мультимовний семантичний розрив, коли англомовний запит ішов до болгарських чи італійських фрагментів і просто не міг знайти семантично близьке. Близько третини – це нестача top-K при крос-компанійних запитах: п’яти фрагментів буквально не вистачало, щоб покрити одразу і Nokia, і Equinor. Решта – термінологічний mismatch, коли dense-only пошук без BM25 не давав ради точному терміну без чіткого семантичного еквівалента.

Категорія Б, частково правильні відповіді, теж розпадається на два характерні шаблони. Перший і частіший – неповне використання контексту: модель бачить потрібний фрагмент, але переносить з нього не все. Цей паттерн часто фіксувався у llama3.1:8b і mistral, рідше або взагалі не фіксувався – у

Sonnet з GPT-4o, що очікувано через різницю у context utilization. Другий шаблон – помилки з табличними числовими даними: модель плутає колонки або не зчитує одиниці виміру. Це прямо випливає з того, що embedding-моделі загального призначення слабко відображають структурні зв'язки всередині таблиць у векторному просторі.

Таблиця 4.7 – Зведена діагностика помилок за модулями системи

Модуль	**Тип помилки**	**Частота**	**Рішення**
Retrieval	Мультимовний розрив	4,7%	Query translation, hybrid BM25+dense
Retrieval	Недостатній K	3,5%	Query decomposition, збільшення K
Retrieval	Термінологічний mismatch	2,3%	BM25 для exact-match
Generation	Неповне використання	14,0%	Потужніша модель
Generation	Помилки таблиць	9,3%	Спеціалізований chunking
–	Галюцинація (з RAG)	0%	–

4.4.3. Галюцинації у baseline-режимі

Таблиця 4.8 – Порівняння поведінки LLM з RAG та без RAG

Характеристика	**З RAG**	**Без RAG (Claude)**	**Без RAG (GPT-4o)**
Input tokens	≈ 2 300	≈ 60	≈ 60
Галюцинації	0%	0% (відмова)	≈ 60% (описові)
Верифікованість	Так (Sources)	Н/з	Ні

Загалом по всій вибірці з включеним RAG понад дві третини відповідей потрапили в категорію А. Але важливіше навіть не сама ця цифра, а характер тих помилок, що лишилися. Усі вони мають характер неповноти, а не фактичної неточності. Тобто система може щось упустити, не помітити, обмежитися половинною відповіддю – але вона не вигадує цифр, яких у звіті немає. Саме це

і робить її придатною для фінансової роботи, де ціна вигаданої цифри значно вища за ціну неповного покриття.

4.5. Техніко-економічне обґрунтування

Будь-яке розгортання такої системи у підсумку зводиться до досить простого вибору з двох сторін. Перша – хмарна модель, де платите за кожен токен API. Друга – локальна, з одноразовим вкладенням у залізо плюс місячний рахунок за світло. Станом на кінець 2025 – початок 2026 року тарифи виглядають приблизно так. GPT-4o коштує близько двох з половиною доларів за мільйон вхідних токенів і десять доларів за мільйон вихідних. Claude Sonnet трохи дорожчий, у нього три і п’ятнадцять відповідно. Обидва провайдери, втім, пропонують механізми зменшення витрат – Batch API дає мінус п’ятдесят відсотків, якщо ви готові почекати до доби на результат, а кешування промптів збиває ціну за повторні фрагменти контексту аж на дев’яносто відсотків. Якщо ж ви хочете локально, базова інвестиція – це GPU на 12-16 гігабайт VRAM. Ollama-моделі при цьому за запит не платяться взагалі.

Сама величина витрат на benchmark дає непогане відчуття масштабів. Шістдесят запитів через дванадцять конфігурацій разом обійшлися приблизно у тридцять центів. З цих тридцяти близько 18 пішло на Sonnet і десь 12 – на GPT-4o. Окрема практично цікава деталь – вибір embedding-моделі тут впливає на рахунок не менше, ніж сам вибір LLM. Bge-large передає у середньому на 10-15 відсотків менше вхідних токенів, ніж mpnet, при цьому якість відповідей не страждала. Цей компактніший контекст напряму перетворюється на менший рахунок у API-провайдера.

Таблиця 4.9 – Проєкція місячних витрат за сценаріями використання

Сценарій	Запитів/міс.	GPT-4o (bge)	Claude Sonnet (bge)	Ollama
Індивідуальний аналітик	200	$1,60	$2,40	$0,00
Команда (5 осіб)	1 000	$8,00	$12,00	$0,00
Департамент (20 осіб)	5 000	$40,00	$60,00	$0,00
Платформа	50 000	$400,00	$600,00	$0,00

У Ollama-варіанта є одна особливість, про яку часто забувають: нульова вартість запиту тут не з повітря. Вона просто перенесена з OPEX у CAPEX. Повна робоча станція для комфортного локального інференсу обходиться у півтори-дві тисячі доларів. GPU тут і є основна частина – 850 за саму карту. До неї треба пристойний CPU доларів за триста-п’ятсот, оперативка і SSD добирають ще десь півтори сотні. Плюс рахунок за електрику – близько двадцяти євро на місяць при активній роботі. Все це разом і створює ту ілюзію “безкоштовності”, яку треба правильно розуміти при плануванні.

Таблиця 4.10 – TCO за 12 місяців для сценарію ”Команда” (1 000 запитів/міс)

Конфігурація	CAPEX, $	OPEX/міс, $	TCO, $
GPT-4o (API) + bge (локально)	850	8,00	946
Claude Sonnet (API) + bge (локально)	850	12,00	994
Повністю локальна (Ollama + bge)	1 500	≈ 20	1 740
Повністю хмарна (GPT-4o + OpenAI emb.)	0	≈ 15,00	180

При невеликому навантаженні – скажімо, тисяча запитів на місяць – дешевшою виходить хмарна конфігурація. Залізо просто не встигає себе окупити. Локальне розгортання починає економити по-справжньому з зовсім інших обсягів. Точка беззбитковості, лежить приблизно на десятці тисяч запитів на місяць, тобто це рівень повноцінного департаменту чи аналітичної команди. Утім, у локального варіанта є аргументи, які рахунок взагалі не

вимірює. Контроль над даними. Незалежність від політики тарифів зовнішніх провайдерів. Можливість працювати без інтернету і без ризику, що завтра ціни виростуть удвічі. Як орієнтир тут можна згадати результати Singireddy et al., які побачили зниження річних витрат майже вдвічі при переході з хмарного GPT-3.5 на локальну зв'язку Mistral 7B з BGE – причому в окремих сценаріях локальна навіть випереджала комерційну за метрикою faithfulness. Загалом економічна доцільність локального підходу очевидно зростає зі збільшенням масштабу організації, і десь у середніх компаніях вона вже починає переважати чисту хмару.

Таблиця 4.11 – Рекомендовані конфігурації за сценаріями

Сценарій	**Конфігурація**	**Обґрунтування**
Дослідження, прототипування	bge + GPT-4o (API)	Найкраща якість/ціна, швидкий відгук
Конфіденційний аналіз	bge + llama3.1:8b (Ollama)	Дані не покидають периметр
Максимальна якість	e5 + Claude Sonnet (API)	Найдовші відповіді, мультимовність
Масове використання	bge + GPT-4o (Batch API)	-50% через Batch, компактний контекст
Обмежений бюджет	mpnet + mistral (Ollama)	Мінімальні вимоги до GPU
Мультимовний аналіз	e5 + Claude Sonnet (API)	Єдина мультимовна embedding

Якщо ці цифри помістити у контекст комерційних аналітичних платформ, які сьогодні фактично закривають професійний ринок, картина стає виразнішою. Bloomberg Terminal коштує близько 24 тисяч доларів на рік. Refinitiv Eikon починається від 3 600. AlphaSense, Tegus – 10-30 тисяч щорічно. Більш оптимізована під потреби невеликої компанії система може коштувати до 300-400$ на рік.

4.6. Перспективи розвитку системи

Результати, які я отримав у експериментах, разом з обмеженнями, що з них виплили, цілком природно намічають конкретні напрямки, у яких хотілося б цей прототип розвивати далі. Я виокремив сім таких напрямків – від найгостріших

речей, які реально заважали у тестуванні, до більш стратегічних, які працюють вже на масштабах. Усі вони у тому чи іншому вигляді вже описані в літературі, тож тут радше карта місцевості, ніж відкритий пошук чогось небаченого.

Найгостріша річ, з якою я зіткнувся – це повний провал dense-only пошуку на болгарських документах, про що було досить детально у попередньому розділі. Тому першим напрямком, на мою думку, має йти гібридний пошук. Технічно це означає поставити поруч з ChromaDB ще й BM25-індекс через бібліотеку `rank_bm25`, після чого зливати результати або через Reciprocal Rank Fusion, або через щось більш адаптивне на кшталт DAT. На BEIR BM25 і досі тримається сильним zero-shot baseline у крос-доменних сценаріях, а у поєднанні з dense, як показує Blended RAG, гібридна схема часом обходить навіть fine-tuned dense-моделі. Сам же BM25 чудово знаходить документи за точними назвами компаній, числовими показниками і технічною термінологією, яка часто залишається англійською навіть у не-англомовних звітах – саме там, де dense нас підвів.

З першим тісно перетинається другий напрямок – query translation як крок попередньої обробки. Якщо за метаданими видно, що документ написаний, скажімо, болгарською, то англомовний запит можна пропустити через LLM-перекладача ще до моменту embedding. Або зробити простіше – одразу дублювати запит кількома мовами і потім зливати результати. Це досить прямо адресує семантичну прірву між запитом і документами з нелатинським алфавітом.

Третій напрямок – це cross-encoder reranker. Просте збільшення top-K не дає лінійного приросту якості, а інколи навіть навпаки. Логічна альтернатива – двоступенева схема: спочатку embedding витягує ширший пул, скажімо двісті кандидатів, після чого окрема reranker-модель типу bge-reranker-v2-m3 або jina-reranker-v2-base-multilingual ранжує їх і залишає вже остаточну п’ятірку. Саме такою стратегією користувались учасники ACM-ICAIF ’24 з FinanceRAG, причому навіть як ансамблем кількох reranker-моделей. Цей

підхід має додатковий бонус для багатомовних задач – мультимовний reranker допомагає з тим самим, з чим бореться гібридний пошук.

Четвертий напрямок – декомпозиція запитів. Це річ зокрема для крос-компанійних задач, де top-K=5 фізично не вміщує достатньо контексту для двох компаній одразу. Простий вихід – замість того щоб сподіватися на удачу, програма сама розбиває складний запит на два окремих retrieval-запити, по одному на кожну компанію, а потім синтезує відповіді на стороні генерації. Це гарантує релевантний контекст по кожній з компаній навіть при невеликому K і добре стикується з усіма попередніми пунктами.

П’ятий напрямок – зовсім іншого характеру. Це автоматизація самого оцінювання. Зараз я роблю цей аналіз якості повністю вручну, хоч і по методиці. Але при масштабувані так не вийде. Якщо інтегрувати RAGAS, то отримаємо три ключові метрики – faithfulness, answer relevancy і context relevance – без потреби в еталонних відповідях. Кореляція цих метрик із людськими оцінками, як показали автори, доходить до 95% на WikiEval. Особливо цінно це було б для автоматичного виявлення тих самих “plausible hallucinations” GPT-4o з baseline-розділу. А RAGChecker зверху додасть розкладання помилок на retrieval-частину і generation-частину, що дуже добре лягає на класифікацію за точками відмови від Barnett.

Шостий напрямок – мультиагентна верифікація. Архітектурно це доволі природне розширення поточного прототипу. Кілька LLM генерують відповіді незалежно, після чого агент-критик аналізує розбіжності і виявляє потенційні помилки. Fatemi та Hu показали, що такий підхід додає до 15% точності для LLaMA3-8B, а спеціалізовані критики – з фокусом, скажімо, на числову точність або на повноту покриття – спромоглися підтягнути загальну якість до рівня GPT-4o-mini за значно меншої вартості. Для фінансового аналізу з його чутливістю до числових помилок це, по факту, є обов’язковим наступним кроком.

І сьомий, напрямок на майбутне – це масштабування самого корпусу. У нинішньому вигляді 66 компаній з 14 країн – це невелика частка загального

обсягу європейських компаній, тож автоматизація збору даних через офіційні джерела типу filings.xbrl.org – очевидний крок. На довшу перспективу це усе впирається у запуск ESAP, який стане єдиною централізованою платформою для фінансової інформації по всьому ЄС у 2027 році. Технічна сторона при такому масштабі потребуватиме перейти з ChromaDB на щось серйозніше – Milvus або Qdrant з підтримкою горизонтального масштабування. І окремо тут стоїть мультимодальний RAG для таблиць і графіків.

Таблиця 4.16 – Напрямки розвитку системи за пріоритетністю

Напрямок	**Пріоритет**	**Складність**	**Проблема, що вирішується**
Гібридний пошук (BM25 + dense)	Високий	Середня	Провал мультимовного retrieval, доменна термінологія
Re-ranker (cross-encoder)	Високий	Низька	Якість ранжування top-K
Автоматизація RAGAS	Високий	Низька	Відсутність автоматизованої оцінки
Query decomposition	Середній	Середня	Провал крос-компанійних запитів при K=5
Query translation	Середній	Низька	Мовний розрив для нелатинських мов
Мультиагентна верифікація	Середній	Середня	Числові помилки та галюцинації
Масштабування + ESAP	Стратегічний	Висока	Обмежене покриття (66 з тисяч компаній ЄС)

Перші три напрямки (гібридний пошук, re-ranker, RAGAS) потребують відносно невеликих зусиль. Решта орієнтовані на перетворення у повноцінну продуктивну систему аналізу європейської корпоративної звітності.

Технологія застосування розробленої системи представлена у вигляді трьох практичних профілів впровадження: персонального (одна аналітична робоча станція, гібридний режим API + локальні embedding), корпоративного (контейнеризація Streamlit + ChromaDB/Milvus + Ollama, reverse proxy, моніторинг метрик) та повністю автономного (відсутність зовнішніх API, e5-large або mpnet + локальні LLM для конфіденційних сценаріїв). Покрокова

інструкція розгортання охоплює шість етапів від `pip install` до запуску benchmark; рекомендації щодо вибору embedding-моделі і LLM прив'язані до конкретних категорій користувачів – інвестиційні аналітики, сел-сайд/бай-сайд, аудитори, регулятори, агрегатори.

З економічного боку запропонований підхід демонструє порядкову перевагу над комерційними аналітичними платформами: найдорожчий комерційний варіант системи (e5-large + Claude Sonnet) обходиться приблизно у 36 дол. на місяць на 3 000 запитів, тоді як підписка на Bloomberg Terminal коштує близько 24 000 дол. на рік, а Refinitiv Eikon чи AlphaSense – співмірно. Така економіка робить технологію доступною не лише великим інституціям, а й окремим аналітикам та малим командам. Подальший розвиток системи спрямовується на гібридний пошук (BM25 + dense), cross-encoder reranking, автоматизовану оцінку якості через RAGAS та інтеграцію з єдиною платформою ESAP після її запуску у 2027 році.

ВИСНОВКИ

У магістерській роботі розв’язано актуальну задачу інтелектуального аналізу європейської корпоративної фінансової звітності на основі технології Retrieval-Augmented Generation, реалізованої у вигляді програмного прототипу з порівняльним оцінюванням різних конфігурацій його ключових компонентів. Для досягнення поставленої мети було послідовно виконано визначені завдання дослідження.

У межах виконання першого завдання проаналізовано проблематику автоматизованого аналізу європейської корпоративної звітності та проведено огляд існуючих підходів і рішень у сфері фінансового NLP. Показано, що європейська регуляторна інфраструктура – від Директиви про прозорість 2004/109/ЄС і єдиного електронного формату ESEF до перспективної платформи ESAP – формує специфічний контекст із низкою бар’єрів для автоматизованого аналізу. Визначено основні з цих бар’єрів, серед яких масштаб корпусу у тисячі документів щорічно, мультимовність із понад двадцятьма чотирма офіційними мовами ЄС, структурна різнорідність між юрисдикціями та систематичні помилки XBRL-тегування. Огляд існуючих рішень підтвердив, що переважна більшість спеціалізованих мовних моделей, бенчмарків і RAG-систем орієнтована на документи SEC та англомовний контент, а комплексне рішення саме під європейський контекст у літературі відсутнє. Кількісно цей розрив підтверджується відомими бенчмарками: FinQA фіксує перевагу експертів-аналітиків (91,16%) над спеціалізованою моделлю FinQANet (61,24%) у понад 30 п.п., а FinanceBench показав, що GPT-4-Turbo з retrieval помилявся приблизно на 81% питань з набору відкритих фінансових запитань – це задає об’єктивну планку складності задачі.

У межах виконання другого завдання досліджено теоретичні основи та компоненти RAG-систем, виконано порівняльний аналіз стратегій фрагментації документів, моделей векторних представлень, генеративних мовних моделей

і методик оцінювання якості. Розглянуто шість стратегій chunking – від фіксованого поділу до гібридних мультигранулярних підходів, три покоління embedding-моделей з еволюцією від статичних векторних представлень до інструкційно-тюнінгованих моделей на основі LLM, п’ять провідних векторних баз даних (FAISS, ChromaDB, Milvus, Qdrant, Pinecone), методи гібридного пошуку (BM25 з dense, Blended RAG, GeAR) та фреймворки автоматизованої оцінки якості (RAGAS, CoFE-RAG, RAGChecker). На основі проведеного аналізу обґрунтовано вибір технологічного стеку прототипу, що включає ChromaDB як вбудовувану векторну базу, LangChain як оркестратор та рекурсивний Markdown-aware chunking з розміром фрагмента 2 500 символів і перекриттям 250 символів.

У межах виконання третього завдання сформовано корпус європейських фінансових звітів та реалізовано програмний прототип RAG-системи. Корпус сформовано на основі 339 ZIP-архівів платформи financialreports.eu і включає 4 528 Markdown-документів від 66 компаній із 14 країн ЄС/ЄЕЗ за період 2019–2025 років, що охоплюють дев’ять типів документів – від річних ESEF-звітів до ESG-звітності – та матеріали різними мовами регіону. Після дворівневої фрагментації за допомогою MarkdownHeaderTextSplitter і RecursiveCharacterTextSplitter отримано 821 994 фрагменти, які паралельно проіндексовано трьома embedding-моделями: multilingual-e5-large (560M параметрів), bge-large-en-v1.5 (326M) та all-mpnet-base-v2 (109M). Сам прототип побудовано як модульну Python-систему із Streamlit-інтерфейсом, що містить три функціональні режими: одиночні запити, паралельне порівняння конфігурацій side-by-side та автоматизований benchmark.

У межах виконання четвертого завдання проведено порівняльне тестування дванадцяти конфігурацій прототипу за критеріями якості відповідей, швидкості генерації та вартості. Експерименти охоплювали п’ять контрольних категорій запитів – фактологічні, описові, порівняльні, аналітичні та крос-компанійні – через комбінації трьох embedding-моделей із чотирма LLM

(llama3.1:8b, mistral, Claude Sonnet, GPT-4o). Показано, що GPT-4o забезпечує найвищу швидкість відповіді при помірній вартості та збалансованій якості, тоді як Claude Sonnet генерує найдетальніші аналітичні відповіді ціною дещо вищої затримки. Відкриті моделі через Ollama продемонстрували нульову вартість при прийнятній якості на фактологічних запитах, але помітно поступалися комерційним на складних аналітичних задачах. Серед embedding-моделей bge-large-en-v1.5 показала найкращий компроміс якості та вартості за рахунок компактнішого контексту. Експерименти з параметром top-K підтвердили оптимальність значення K=5, а мультимовне тестування виявило принциповий провал dense-only пошуку на нелатинських алфавітах. Контрольний baseline без RAG продемонстрував, що саме retrieval усуває галюцинації та забезпечує верифіковану обґрунтованість відповідей. Загальна вартість 60 експериментальних запитів (12 конфігурацій × 5 контрольних запитів) становила приблизно 0,30 дол.; GPT-4o стабільно лишався найшвидшим (2,52–3,35 с), Claude Sonnet – найдетальнішим, але повільнішим (7,05–7,96 с). Принципово важливим є результат категорії «галюцинації»: жодна з 86 відповідей з увімкненим RAG не потрапила до цієї категорії, тоді як близько 80% відповідей GPT-4o у baseline без RAG містили правдоподібні, але необґрунтовані фрагменти. Це підтверджує практичну цінність RAG як механізму забезпечення верифікованості у фінансовому домені та робить систему придатною до використання у задачах, де ціна вигаданої цифри значно вища за ціну неповного покриття.

У межах виконання п’ятого завдання розроблено рекомендації щодо впровадження системи та оцінено техніко-економічну доцільність варіантів розгортання. Сформульовано конкретні конфігурації для п’яти категорій цільових користувачів – інвестиційних компаній, фінансових аналітиків, аудиторських фірм, регуляторних органів та платформ фінансових даних – з урахуванням їхніх вимог до деталізації, швидкості, вартості та мультимовності. Виконано порівняння хмарної та локальної моделей витрат, яке показало, що

повністю локальна конфігурація з e5-large та llama3.1:8b забезпечує нульові операційні витрати при амортизованій вартості GPU-обладнання, тоді як найдорожчий комерційний варіант з e5-large та Claude Sonnet залишається доступним на рівні близько 36 доларів на три тисячі запитів на місяць, що на один-два порядки нижче за комерційні аналітичні платформи Bloomberg Terminal, Refinitiv Eikon чи AlphaSense. Визначено сім пріоритетних напрямків подальшого розвитку, серед яких гібридний пошук BM25 з dense, cross-encoder reranking, query translation, автоматизація оцінки якості через RAGAS, мультиагентна верифікація та інтеграція з майбутньою платформою ESAP.

Наукова новизна одержаних результатів полягає в запропонованому підході до побудови RAG-системи, спеціалізованої на корпусі європейської корпоративної звітності у форматі ESEF/Markdown, у межах якого поєднано кілька аналітичних рівнів в єдиному дослідницькому контурі. По-перше, запропоновано та експериментально досліджено архітектуру RAG-системи, орієнтовану саме на мультимовний європейський контекст, на відміну від існуючих рішень, що зосереджені переважно на документах SEC та англомовному контенті. По-друге, проведено порівняльний аналіз впливу вибору embedding-моделі та генеративної LLM на якість аналітичних відповідей у фінансовому домені, що дозволив виявити нетривіальні ефекти – зокрема перевагу bge-large-en-v1.5 у компактності контексту без втрати якості. По-третє, у межах єдиного прототипу поєднано порівняльне оцінювання відкритих та комерційних компонентів RAG-системи на даних з 14 країн ЄС/ЄЕЗ, що формує методологічну базу для подібних досліджень в інших регіональних доменах.

Практичне значення роботи полягає в тому, що запропонований підхід і реалізований програмний прототип дають змогу обґрунтовано формувати конфігурацію RAG-системи для аналізу європейської фінансової звітності з урахуванням мовного складу корпусу, обсягу запитів, вимог до конфіденційності та бюджетних обмежень. Система забезпечує верифікований і пояснюваний характер відповідей за рахунок повернення джерел, що є

важливим для практичного використання у фінансово-аналітичному середовищі. Сформульовані рекомендації щодо оптимальних конфігурацій (вибір embedding-моделі, LLM, параметрів пошуку) для різних сценаріїв мають практичну цінність, доведену експериментально, що знадобиться при впровадженні подібних систем. Виявлені обмеження прототипу та визначені напрямки розвитку формують основу для подальшої інженерної роботи.

Подальший розвиток роботи доцільно спрямувати на впровадження гібридного пошуку через поєднання BM25 з dense retrieval для подолання семантичного розриву на нелатинських алфавітах, додавання cross-encoder reranking для підвищення точності ранжування фрагментів, реалізацію query translation як кроку попередньої обробки запитів, інтеграцію автоматизованих фреймворків оцінки якості (RAGAS, RAGChecker) замість ручного аналізу, а також розширення корпусу через автоматизований збір даних з офіційних джерел і подальшу інтеграцію з платформою ESAP після її запуску. Перспективним є також впровадження мультимодального RAG-конвеєра для роботи з таблицями та графіками і застосування мультиагентних архітектур із спеціалізованими критиками для підвищення точності числових відповідей у фінансовому домені.

Отже, мету магістерської роботи досягнуто. Запропонований підхід, реалізований у вигляді працездатного прототипу інтелектуальної системи аналізу європейської корпоративної фінансової звітності на корпусі 821 994 фрагментів із 4 528 документів 66 компаній 14 країн ЄС/ЄЕЗ за період 2019–2025 років, забезпечує підвищення якості автоматизованого аналізу за рахунок поєднання семантичного пошуку, верифікованої генерації з нульовим рівнем галюцинацій у режимі RAG та пояснюваності через панель Sources із прив’язкою кожної відповіді до конкретних фрагментів. Сукупні експериментальні витрати на API становили близько 0,30 дол. за 60 запитів, а місячна проєкція найдорожчої комерційної конфігурації – близько 36 дол. на 3 000 запитів, що на один-два порядки нижче за підписку на Bloomberg

Terminal (~24 000 дол. на рік) і робить технологію доступною не лише великим інституціям, а й окремим аналітикам та малим командам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Directive 2004/109/EC of the European Parliament and of the Council on the harmonisation of transparency requirements. Official Journal of the European Union. 2004. L 390. P. 38–57.
2. Commission Delegated Regulation (EU) 2019/815 supplementing Directive 2004/109/EC with regard to regulatory technical standards on the specification of a single electronic reporting format. Official Journal of the European Union. 2019. L 143. P. 1–792.
3. Sran G., Tuijn M., Vollon L. The Capital Market Effects of Centralizing Regulated Financial Information. Working Paper. 2023.
4. Regulation (EU) 2023/2859 of the European Parliament and of the Council establishing a European single access point. Official Journal of the European Union. 2023.
5. Kobiela-Pionnier K., Karwowski M. When Polish companies met the ESEF Taxonomy: The correctness of XBRL tags in the first year of the ESMA ESEF mandate. Zeszyty Teoretyczne Rachunkowości. 2024. Vol. 48, Nr 3. P. 177–195.
6. Financialreports.eu – платформа європейської корпоративної звітності. URL: https://financialreports.eu (дата звернення: 10.03.2026).
7. Jørgensen P., Brandt T., Hartmann M. et al. MultiFin: A Dataset for Multilingual Financial NLP. Findings of EACL 2023. P. 894–909.
8. El Haj M. et al. The Financial Narrative Summarisation Shared Task (FNS 2023). FNP 2023 Workshop, IEEE. 2023.
9. Wu S., Irsoy O., Lu S. et al. BloombergGPT: A Large Language Model for Finance. arXiv preprint. 2023. arXiv:2303.17564.
10. Yang H., Liu X., Wang C. FinGPT: Open-Source Financial Large Language Models. arXiv preprint. 2023. arXiv:2306.06031.

11. Yang Y., Tang Y., Tam K. InvestLM: A Large Language Model for Investment using Financial Domain Instruction Tuning. arXiv preprint. 2023. arXiv:2309.13064.
12. DISC-FinLLM: A Chinese Financial Large Language Model based on Multiple Experts Fine-tuning. arXiv preprint. 2023. arXiv:2310.15205.
13. A Survey of Large Language Models in Finance (FinLLMs). arXiv preprint. 2024. arXiv:2402.02315.
14. A Survey of Large Language Models for Financial Applications: Progress, Prospects and Challenges. arXiv preprint. 2024. arXiv:2406.11903.
15. Zhu F. et al. TAT-LLM: A Specialized Language Model for Discrete Reasoning over Financial Tabular and Textual Data. Proceedings of ACM ICAIF 2024.
16. Chen Z., Chen W., Smiley C. et al. FinQA: A Dataset of Numerical Reasoning over Financial Data. Proceedings of EMNLP 2021. P. 3697–3711.
17. Zhu F., Lei W., Huang Y. et al. TAT-QA: A Question Answering Benchmark on a Hybrid of Tabular and Textual Content in Finance. Proceedings of ACL 2021. P. 3277–3287.
18. Chen Z., Li S., Smiley C. et al. ConvFinQA: Exploring the Chain of Numerical Reasoning in Conversational Finance QA. Proceedings of EMNLP 2022.
19. Islam P., Kannappan A., Kiela D. et al. FinanceBench: A New Benchmark for Financial Question Answering. arXiv preprint. 2023. arXiv:2311.11944.
20. Xie Q. et al. FinBen: A Holistic Financial Benchmark for Large Language Models. arXiv preprint. 2024. arXiv:2402.12659.
21. FAMMA: A Benchmark for Financial Domain Multilingual Multimodal Question Answering. arXiv preprint. 2024. arXiv:2410.04526.
22. OmniEval: An Omnidirectional RAG Evaluation Benchmark in Financial Domain. arXiv preprint. 2024. arXiv:2412.13018.
23. Sarmah B. et al. HybridRAG: Integrating Knowledge Graphs and Vector Retrieval Augmented Generation. Proceedings of ACM ICAIF 2024.

24. MultiFinRAG: An Optimized Multimodal RAG Framework for Financial QA. arXiv preprint. 2025. arXiv:2506.20821.
25. Kim S., Song H., Seo H., Kim H. Optimizing Retrieval Strategies for Financial Question Answering Documents in RAG Systems. ICLR 2025 Workshop on Advances in Financial AI. 2025. arXiv:2503.15191.
26. Setty S., Thakkar H., Lee A. et al. Improving Retrieval for RAG based Question Answering Models on Financial Documents. arXiv preprint. 2024. arXiv:2404.07221.
27. German FinBERT: A German Pre-trained Language Model for Financial Texts. arXiv preprint. 2023. arXiv:2311.08793.
28. Hillebrand L. et al. KPI-BERT: A Joint Named Entity Recognition and Relation Extraction Model for Financial Reports. arXiv preprint. 2022. arXiv:2208.02140.
29. Lewis P., Perez E., Piktus A. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Advances in Neural Information Processing Systems. 2020. Vol. 33. P. 9459–9474.
30. Gao Y., Xiong Y., Gao X. et al. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv preprint. 2024. arXiv:2312.10997.
31. Gupta S., Ranjan R., Singh S. A Comprehensive Survey of Retrieval-Augmented Generation (RAG): Evolution, Current Landscape and Future Directions. arXiv preprint. 2024. arXiv:2410.12837.
32. Fan W., Ding Y., Ning L. et al. A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models. Proceedings of KDD 2024. arXiv:2405.06211.
33. Huang Y., Huang J. A Survey on Retrieval-Augmented Text Generation for Large Language Models. arXiv preprint. 2024. arXiv:2404.10981.
34. Barnett S., Kurniawan S., Thudumu S. et al. Seven Failure Points When Engineering a Retrieval Augmented Generation System. arXiv preprint. 2024. arXiv:2401.05856.

35. Jimeno Yepes A. et al. Financial Report Chunking for Effective Retrieval Augmented Generation. arXiv preprint. 2024. arXiv:2402.05131.
36. ChunkRAG: Novel LLM-Chunk Filtering Method for RAG Systems. arXiv preprint. 2024. arXiv:2410.19572.
37. SAGE: A Framework of Precise Retrieval for RAG. arXiv preprint. 2025. arXiv:2503.01713.
38. Sarthi P., Abdullah S., Tuli A. et al. RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval. arXiv preprint. 2024. arXiv:2401.18059.
39. S² Chunking: A Hybrid Framework for Document Segmentation Through Integrated Spatial and Semantic Analysis. arXiv preprint. 2025. arXiv:2501.05485.
40. Muennighoff N. et al. MTEB: Massive Text Embedding Benchmark. Proceedings of EACL 2023.
41. MMTEB: Massive Multilingual Text Embedding Benchmark. arXiv preprint. 2025. arXiv:2502.13595.
42. Thakur N. et al. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. NeurIPS 2021 Datasets and Benchmarks. arXiv:2104.08663.
43. Arctic-Embed: Scalable, Efficient, and Accurate Text Embedding Models. arXiv preprint. 2024. arXiv:2405.05374.
44. Nomic Embed: Training a Reproducible Long Context Text Embedder. Technical Report. 2024. arXiv:2402.01613.
45. NV-Embed: Improved Techniques for Training LLMs as Generalist Embedding Models. arXiv preprint. 2024. arXiv:2405.17428.
46. Cao H. Recent Advances in Text Embedding: A Comprehensive Review of Top-Performing Methods on the MTEB Benchmark. arXiv preprint. 2024. arXiv:2406.01607.
47. Tang Y., Yang Y. Do We Need Domain-Specific Embedding Models? An Empirical Investigation. arXiv preprint. 2024. arXiv:2409.18511.

48. BRIGHT: A Realistic and Challenging Benchmark for Reasoning-Intensive Retrieval. arXiv preprint. 2024. arXiv:2407.12883.
49. Jina Embeddings v3: Multilingual Embeddings With Task LoRA. arXiv preprint. 2024. arXiv:2409.10173.
50. Johnson J., Douze M., Jégou H. Billion-scale similarity search with GPUs. IEEE Transactions on Big Data. 2021. Vol. 7, № 3. P. 535–547.
51. Blended RAG: Improving RAG Accuracy with Semantic Search and Hybrid Query-Based Retrievers. IEEE. 2024.
52. DAT: Dynamic Alpha Tuning for Hybrid Retrieval in RAG. arXiv preprint. 2025. arXiv:2503.23013.
53. Cormack G., Clarke C., Buettcher S. Reciprocal Rank Fusion outperforms Condorcet and individual Rank Learning Methods. Proceedings of SIGIR 2009.
54. GeAR: Graph-enhanced Agent for Retrieval-augmented Generation. arXiv preprint. 2024. arXiv:2412.18431.
55. Es S. et al. RAGAS: Automated Evaluation of Retrieval Augmented Generation. arXiv preprint. 2024. arXiv:2309.15217.
56. Saad-Falcon J. et al. ARES: An Automated Evaluation Framework for Retrieval-Augmented Generation Systems. arXiv preprint. 2023. arXiv:2311.09476.
57. CoFE-RAG: A Comprehensive Full-chain Evaluation Framework for RAG with Enhanced Data Diversity. arXiv preprint. 2024. arXiv:2410.12248.
58. RAGChecker: A Fine-grained Framework for Diagnosing RAG. arXiv preprint. 2024. arXiv:2408.08067.
59. Enterprise RAG Evaluation with Open-Source Components. arXiv preprint. 2024. arXiv:2406.11424.
60. ChatQA: Surpassing GPT-4 on Conversational QA and RAG. arXiv preprint. 2024. arXiv:2401.10225.
61. Can Open-Source LLMs Compete with Commercial Models? Exploring the Few-Shot Performance of Current GPT Models in Biomedical Tasks. arXiv preprint. 2024. arXiv:2407.13511.

62. Bridging the LLM Accessibility Divide: Performance, Fairness, and Cost of Closed versus Open LLMs. arXiv preprint. 2025. arXiv:2503.11827.
63. Wu S., Xiong Y., Cui Y. et al. Retrieval-Augmented Generation for Natural Language Processing: A Survey. arXiv preprint. 2025. arXiv:2407.13193.
64. Zhao P., Zhang H., Yu Q. et al. Retrieval-Augmented Generation for AI-Generated Content: A Survey. arXiv preprint. 2024. arXiv:2402.19473.
65. Towards Trustworthy RAG for LLMs: A Survey. arXiv preprint. 2025. arXiv:2502.06872.
66. A Survey on Knowledge-Oriented Retrieval-Augmented Generation. arXiv preprint. 2025. arXiv:2503.10677.
67. Wang L. et al. E5-Mistral: Improving Text Embeddings with Large Language Models. arXiv preprint. 2024. arXiv:2401.00368.
68. Lee J. et al. Gecko: Versatile Text Embeddings Distilled from Large Language Models. arXiv preprint. 2024. arXiv:2403.20327.
69. Wang L. et al. Text Embeddings by Weakly-Supervised Contrastive Pre-training. arXiv preprint. 2022. arXiv:2212.03533.
70. FinDABench: Benchmarking Financial Data Analysis Ability of LLMs. arXiv preprint. 2024. arXiv:2401.02982.
71. Koncel-Kedziorski R. et al. BizBench: A Quantitative Reasoning Benchmark for Business and Finance. arXiv preprint. 2023. arXiv:2311.06602.
72. FinTextQA: A Dataset for Long-form Financial Question Answering. arXiv preprint. 2024. arXiv:2405.09980.
73. Enhancing Financial QA with a Multi-Agent Reflection Framework. arXiv preprint. 2024. arXiv:2410.21741.
74. Bridging Language Models and Financial Analysis. arXiv preprint. 2025. arXiv:2503.22693.
75. FinanceRAG: ACM-ICAIF ’24 Competition System. arXiv preprint. 2024. arXiv:2411.16732.

76. Zhu F. et al. TAT-LLM: A Specialized Language Model for Discrete Reasoning over Financial Tabular and Textual Data. arXiv preprint. 2024. arXiv:2401.13223.
77. IRSC: Zero-shot Evaluation Benchmark for Information Retrieval in RAG Scenarios. arXiv preprint. 2024. arXiv:2409.15763.
78. RAGEval: Scenario Specific RAG Evaluation Dataset Generation. arXiv preprint. 2024. arXiv:2408.01262.
79. BERGEN: A Benchmarking Library for Retrieval-Augmented Generation. arXiv preprint. 2024. arXiv:2407.01102.
80. RAGBench: Explainable Benchmark for Retrieval-Augmented Generation Systems. arXiv preprint. 2025. arXiv:2407.11005.
81. Passage Segmentation of Documents for Extractive Question Answering. arXiv preprint. 2025. arXiv:2501.09940.
82. Comparative Analysis of Retrieval Systems in RAG Pipelines. arXiv preprint. 2024. arXiv:2405.02048.

Додаток А

(довідковий)

Код модуля `rag_engine.py` – ядро RAG-пайплайну

```
import time
from pathlib import Path

import chromadb
from sentence_transformers import SentenceTransformer

from config import (
    CHROMA_BASE_DIR,
    DEFAULT_TOP_K,
    EMBEDDING_MODELS,
    LLM_CONFIGS,
    PRICING,
    SYSTEM_PROMPT,
)

_embedding_cache: dict[str, SentenceTransformer] = {}

def get_embedding_model(model_name: str) -> SentenceTransformer:
    if model_name not in _embedding_cache:
        model_id = EMBEDDING_MODELS[model_name]
        _embedding_cache[model_name] = SentenceTransformer(model_id, trust_remote_code=True)
    return _embedding_cache[model_name]

def get_collection(model_name: str) -> chromadb.Collection:
    db_path = str(Path(CHROMA_BASE_DIR) / model_name)
    client = chromadb.PersistentClient(path=db_path)
    return client.get_collection("financial_reports")

def retrieve(
    query: str,
    embedding_model: str,
    top_k: int = DEFAULT_TOP_K,
    filters: dict | None = None,
) -> list[dict]:
    model = get_embedding_model(embedding_model)
```

```
    if "e5" in embedding_model:
        encode_query = f"query: {query}"
    elif "bge" in embedding_model:
        encode_query = f"Represent this sentence for searching relevant passages: {query}"
    else:
        encode_query = query

    query_embedding = model.encode(
        [encode_query], normalize_embeddings=True
    ).tolist()

    where = _build_where_filter(filters)

    collection = get_collection(embedding_model)

    query_params = {
        "query_embeddings": query_embedding,
        "n_results": top_k,
    }
    if where:
        query_params["where"] = where

    results = collection.query(**query_params)

    chunks = []
    if results["documents"] and results["documents"][0]:
        for i, doc in enumerate(results["documents"][0]):
            chunks.append({
                "text": doc,
                "metadata": results["metadatas"][0][i] if results["metadatas"] else {},
                "distance": results["distances"][0][i] if results["distances"] else None,
            })

    return chunks

def _build_where_filter(filters: dict | None) -> dict | None:
    if not filters:
        return None
```

```
    conditions = []
    for key in ["company", "country", "year", "report_type"]:
        values = filters.get(key)
        if not values:
            continue
        if isinstance(values, str):
            conditions.append({key: values})
        elif isinstance(values, list) and len(values) == 1:
            conditions.append({key: values[0]})
        elif isinstance(values, list) and len(values) > 1:
            conditions.append({"$or": [{key: v} for v in values]})

    if not conditions:
        return None
    if len(conditions) == 1:
        return conditions[0]
    return {"$and": conditions}

def generate(
    query: str,
    context_chunks: list[dict],
    llm_name: str,
) -> dict:
    llm_config = LLM_CONFIGS[llm_name]
    provider = llm_config["provider"]

    context_parts = []
    for i, chunk in enumerate(context_chunks, 1):
        meta = chunk["metadata"]
        header = f"[Source {i}: {meta.get('company', '?')}, {meta.get('year', '?')}, {meta.get('report_type
        context_parts.append(f"{header}\n{chunk['text']}")

    context_str = "\n\n---\n\n".join(context_parts)

    user_message = f"""Context from financial reports:

{context_str}

---
```

```
Question: {query}"""

    t0 = time.time()

    if provider == "ollama":
        result = _generate_ollama(llm_config["model"], user_message)
    elif provider == "anthropic":
        result = _generate_anthropic(llm_config, user_message)
    elif provider == "openai":
        result = _generate_openai(llm_config, user_message)
    else:
        raise ValueError(f"Unknown provider: {provider}")

    latency = time.time() - t0
    result["latency"] = round(latency, 2)

    if provider == "ollama":
        result["cost"] = 0.0
    else:
        pricing = PRICING.get(llm_config["model"], {"input": 0, "output": 0})
        result["cost"] = round(
            (result["input_tokens"] * pricing["input"] / 1_000_000)
            + (result["output_tokens"] * pricing["output"] / 1_000_000),
            6,
        )

    return result

def _generate_ollama(model: str, user_message: str) -> dict:
    import requests

    response = requests.post(
        "http://localhost:11434/api/chat",
        json={
            "model": model,
            "messages": [
                {"role": "system", "content": SYSTEM_PROMPT},
                {"role": "user", "content": user_message},
            ],
            "stream": False,
```

```
        },
        timeout=120,
    )
    response.raise_for_status()
    data = response.json()

    return {
        "answer": data["message"]["content"],
        "input_tokens": data.get("prompt_eval_count", 0),
        "output_tokens": data.get("eval_count", 0),
    }

def _generate_anthropic(config: dict, user_message: str) -> dict:
    from anthropic import Anthropic

    client = Anthropic(api_key=config["api_key"])
    response = client.messages.create(
        model=config["model"],
        max_tokens=1024,
        system=SYSTEM_PROMPT,
        messages=[{"role": "user", "content": user_message}],
    )

    return {
        "answer": response.content[0].text,
        "input_tokens": response.usage.input_tokens,
        "output_tokens": response.usage.output_tokens,
    }

def _generate_openai(config: dict, user_message: str) -> dict:
    from openai import OpenAI

    client = OpenAI(api_key=config["api_key"])
    response = client.chat.completions.create(
        model=config["model"],
        max_tokens=1024,
        messages=[
            {"role": "system", "content": SYSTEM_PROMPT},
            {"role": "user", "content": user_message},
        ],
```

```
    )

    return {
        "answer": response.choices[0].message.content,
        "input_tokens": response.usage.prompt_tokens,
        "output_tokens": response.usage.completion_tokens,
    }

def query_rag(
    query: str,
    embedding_model: str,
    llm_name: str,
    top_k: int = DEFAULT_TOP_K,
    filters: dict | None = None,
) -> dict:

    chunks = retrieve(query, embedding_model, top_k, filters)

    result = generate(query, chunks, llm_name)
    result["sources"] = chunks

    return result
```

Додаток Б

(довідковий)

Код модуля `phase2_index.py` – індексація корпусу

```
import json
import gc
import os
import time
from pathlib import Path

import numpy as np
import chromadb
from langchain_text_splitters import (
    MarkdownHeaderTextSplitter,
    RecursiveCharacterTextSplitter,
)
from sentence_transformers import SentenceTransformer

METADATA_PATH = Path("data/metadata.json")
CHROMA_BASE_DIR = Path("chroma_db")
EMBEDDINGS_CACHE_DIR = Path("embeddings_cache")

EMBEDDING_MODELS = {
    "multilingual-e5-large": "intfloat/multilingual-e5-large",
    "all-mpnet-base-v2": "sentence-transformers/all-mpnet-base-v2",
    "bge-large-en-v1.5": "BAAI/bge-large-en-v1.5",
}

CHUNK_SIZE = 2500
CHUNK_OVERLAP = 250
CHROMA_BATCH_SIZE = 5000

ENCODE_BATCH_SIZES = {
    "bge-large-en-v1.5": 512,
    "all-mpnet-base-v2": 512,
}

def load_metadata() -> list[dict]:
    with open(METADATA_PATH, "r", encoding="utf-8") as f:
```

```
        return json.load(f)


def split_document(text: str, meta: dict) -> list[dict]:
    headers_to_split = [
        ("#", "section_h1"),
        ("##", "section_h2"),
        ("###", "section_h3"),
    ]
    md_splitter = MarkdownHeaderTextSplitter(
        headers_to_split_on=headers_to_split,
        strip_headers=False,
    )
    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size=CHUNK_SIZE,
        chunk_overlap=CHUNK_OVERLAP,
        separators=["\n\n", "\n", ". ", " "],
    )

    md_docs = md_splitter.split_text(text)
    chunks = []
    for md_doc in md_docs:
        section = (
            md_doc.metadata.get("section_h3")
            or md_doc.metadata.get("section_h2")
            or md_doc.metadata.get("section_h1")
            or ""
        )
        sub_chunks = text_splitter.split_text(md_doc.page_content)
        for chunk_text in sub_chunks:
            if len(chunk_text.strip()) < 20:
                continue
            chunks.append({
                "text": chunk_text,
                "metadata": {
                    "company": meta["company"],
                    "country": meta.get("country") or "",
                    "year": str(meta.get("year") or ""),
                    "report_type": meta.get("report_type") or "",
                    "section": section,
                    "source_file": meta.get("source_file") or "",
```

```
                },
            })
    return chunks


def build_all_chunks(metadata: list[dict]) -> tuple[list[str], list[dict]]:
    print("Building chunks from all documents...")
    all_texts = []
    all_metas = []
    errors = 0

    for i, meta in enumerate(metadata):
        fpath = Path(meta["source_file"])
        try:
            text = fpath.read_text(encoding="utf-8", errors="ignore")
            chunks = split_document(text, meta)
            for c in chunks:
                all_texts.append(c["text"])
                all_metas.append(c["metadata"])
        except Exception as e:
            errors += 1
            if errors <= 5:
                print(f"  Error reading {fpath}: {e}")

        if (i + 1) % 1000 == 0 or (i + 1) == len(metadata):
            print(f"  Processed {i + 1}/{len(metadata)} docs, {len(all_texts)} chunks")

    if errors:
        print(f"  Total read errors: {errors}")
    return all_texts, all_metas


def encode_or_load(model_name: str, model_id: str, texts: list[str]) -> np.ndarray:
    os.makedirs(EMBEDDINGS_CACHE_DIR, exist_ok=True)
    cache_path = EMBEDDINGS_CACHE_DIR / f"{model_name}.npy"

    if cache_path.exists():
        print(f"  Found cached embeddings at {cache_path}")
        embeddings = np.load(str(cache_path))
        if len(embeddings) == len(texts):
            print(f"  Loaded {len(embeddings)} embeddings from cache (skipping encoding)")
            return embeddings
```

```
    else:
        print(f"  Cache size mismatch ({len(embeddings)} vs {len(texts)}), re-encoding...")

import torch
device = "cuda" if torch.cuda.is_available() else "cpu"

t0 = time.time()
print(f"  Loading model...")
model = SentenceTransformer(model_id, trust_remote_code=True, device=device)
model.half()
print(f"  Model loaded in {time.time() - t0:.1f}s (device={device}, fp16)")

batch_size = ENCODE_BATCH_SIZES.get(model_name, 256)
total = len(texts)
print(f"  Encoding {total} chunks (batch_size={batch_size})...")

if "e5" in model_name:
    encode_texts = [f"passage: {t}" for t in texts]
elif "bge" in model_name:
    encode_texts = [f"Represent this sentence for searching relevant passages: {t}" for t in texts]
else:
    encode_texts = texts

t_enc = time.time()
embeddings = model.encode(
    encode_texts,
    batch_size=batch_size,
    show_progress_bar=True,
    normalize_embeddings=True,
    convert_to_numpy=True,
)
if embeddings.dtype != np.float32:
    embeddings = embeddings.astype(np.float32)
elapsed = time.time() - t_enc
print(f"  Encoding done in {elapsed:.1f}s ({total/elapsed:.0f} chunks/s)")

print(f"  Saving embeddings to {cache_path}...")
np.save(str(cache_path), embeddings)
print(f"  Saved ({cache_path.stat().st_size / 1e9:.1f} GB)")
```

```
    del model
    gc.collect()
    if device == "cuda":
        torch.cuda.empty_cache()

    return embeddings

def insert_into_chroma(db_path: str, texts: list[str], metas: list[dict], embeddings: np.ndarray) -> float
    client = chromadb.PersistentClient(path=db_path)
    try:
        client.delete_collection("financial_reports")
    except Exception:
        pass

    collection = client.create_collection(
        name="financial_reports",
        metadata={"hnsw:space": "cosine"},
    )

    total = len(texts)
    t0 = time.time()
    print(f"  Inserting {total} chunks into ChromaDB (batch_size={CHROMA_BATCH_SIZE})...")

    for batch_start in range(0, total, CHROMA_BATCH_SIZE):
        batch_end = min(batch_start + CHROMA_BATCH_SIZE, total)
        batch_ids = [f"chunk_{j}" for j in range(batch_start, batch_end)]

        collection.add(
            ids=batch_ids,
            embeddings=embeddings[batch_start:batch_end].tolist(),
            documents=texts[batch_start:batch_end],
            metadatas=metas[batch_start:batch_end],
        )

        elapsed = time.time() - t0
        print(f"    {batch_end}/{total} inserted ({elapsed:.0f}s)")

    total_time = time.time() - t0
    print(f"  ChromaDB insert done in {total_time:.1f}s")
    return total_time
```

```
def index_with_model(model_name: str, model_id: str, texts: list[str], metas: list[dict]):
    print(f"\n{'='*60}")
    print(f"Model: {model_name} ({model_id})")
    print(f"{'='*60}")

    db_path = str(CHROMA_BASE_DIR / model_name)
    os.makedirs(db_path, exist_ok=True)

    t_start = time.time()
    embeddings = encode_or_load(model_name, model_id, texts)
    encode_time = time.time() - t_start

    insert_time = insert_into_chroma(db_path, texts, metas, embeddings)

    del embeddings
    gc.collect()

    total_time = encode_time + insert_time
    return {
        "encode_time": round(encode_time, 1),
        "insert_time": round(insert_time, 1),
        "total_time": round(total_time, 1),
    }

def main():
    metadata = load_metadata()
    print(f"Loaded metadata for {len(metadata)} documents")

    texts, metas = build_all_chunks(metadata)
    total_chunks = len(texts)
    print(f"\nTotal chunks: {total_chunks}")

    results = {}
    for model_name, model_id in EMBEDDING_MODELS.items():
        stats = index_with_model(model_name, model_id, texts, metas)
        results[model_name] = stats

    print(f"\n{'='*60}")
    print("PHASE 2 — INDEXING SUMMARY (bge + mpnet)")
```

```
    print(f"{'='*60}")
    print(f"Total chunks: {total_chunks}")
    print(f"Note: multilingual-e5-large was indexed earlier (~4145s encode + ~1775s insert)")
    print(f"\n{'Model':30s} {'Encode(s)':>10s} {'Insert(s)':>10s} {'Total(s)':>10s} {'Speed':>12s}")
    print("-" * 74)
    for name, r in results.items():
        speed = total_chunks / r["encode_time"] if r["encode_time"] > 0 else 0
        print(f"{name:30s} {r['encode_time']:10.1f} {r['insert_time']:10.1f} {r['total_time']:10.1f} {speed

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

Додаток В

(довідковий)

Код модуля `phase1_extract.py` – препроцесинг корпусу

```
import json
import os
import re
import shutil
import zipfile
from collections import Counter, defaultdict
from pathlib import Path

RAW_DATA_DIR = Path("raw_data")
OUTPUT_DIR = Path("data/reports")
METADATA_PATH = Path("data/metadata.json")

def get_company_name(zip_name: str) -> str:

    name = zip_name.replace(".zip", "")
    name = re.sub(r"-filings(\(\d+\))?$", "", name)
    return name

def parse_filename_metadata(filename: str, company: str) -> dict:
    base = filename.replace(".md", "")

    date_match = re.match(r"^(\d{4}-\d{2}-\d{2})_", base)
    date = date_match.group(1) if date_match else None
    year = int(date[:4]) if date else None

    filing_match = re.search(r"_(\d{5,})$", base)
    filing_id = filing_match.group(1) if filing_match else None

    if date and filing_id:
        report_type = base[len(date) + 1 : -(len(filing_id) + 1)]
    elif date:
        report_type = base[len(date) + 1 :]
    else:
        report_type = base
```

```
    report_type = re.sub(r"_/_", "/", report_type)

    return {
        "filename": filename,
        "company": company,
        "date": date,
        "year": year,
        "report_type": report_type,
        "filing_id": filing_id,
        "country": None,
    }

def sanitize_path(name: str) -> str:
    return re.sub(r'[<>:"/\\|?*]', '_', name)

def extract_all():
    OUTPUT_DIR.mkdir(parents=True, exist_ok=True)
    METADATA_PATH.parent.mkdir(parents=True, exist_ok=True)

    all_metadata = []
    zip_files = sorted(RAW_DATA_DIR.glob("*.zip"))
    print(f"Found {len(zip_files)} ZIP archives")

    companies_processed = set()
    errors = []

    for i, zpath in enumerate(zip_files):
        company = get_company_name(zpath.name)
        companies_processed.add(company)
        company_dir = OUTPUT_DIR / company
        company_dir.mkdir(parents=True, exist_ok=True)

        try:
            with zipfile.ZipFile(zpath, "r") as zf:
                for member in zf.namelist():
                    if not member.endswith(".md"):
                        continue

                    fname = member
                    meta = parse_filename_metadata(fname, company)
```

```
                    safe_fname = sanitize_path(fname)
                    dest = company_dir / safe_fname

                    if dest.exists():
                        continue

                    data = zf.read(member)
                    dest.write_bytes(data)

                    meta["source_file"] = f"data/reports/{company}/{safe_fname}"
                    meta["size_bytes"] = len(data)
                    all_metadata.append(meta)
            except Exception as e:
                errors.append(f"{zpath.name}: {e}")

            if (i + 1) % 50 == 0 or (i + 1) == len(zip_files):
                print(f"  Processed {i + 1}/{len(zip_files)} archives...")

    with open(METADATA_PATH, "w", encoding="utf-8") as f:
        json.dump(all_metadata, f, ensure_ascii=False, indent=2)

    if errors:
        print(f"\nErrors ({len(errors)}):")
        for e in errors:
            print(f"  {e}")

    return all_metadata, companies_processed


def print_statistics(metadata: list, companies: set):
    print("\n" + "=" * 60)
    print("PHASE 1 — STATISTICS")
    print("=" * 60)

    print(f"\nTotal companies:  {len(companies)}")
    print(f"Total documents:  {len(metadata)}")

    type_counts = Counter(m["report_type"] for m in metadata)
    print(f"\nDocuments by report type:")
    for rt, count in type_counts.most_common():
```

```
        print(f"  {rt:50s} {count:5d}")

    year_counts = Counter(m["year"] for m in metadata if m["year"])
    print(f"\nDocuments by year:")
    for year in sorted(year_counts.keys()):
        print(f"  {year}: {year_counts[year]:5d}")

    sizes = [m["size_bytes"] for m in metadata]
    avg_size = sum(sizes) / len(sizes) if sizes else 0
    total_size = sum(sizes)
    print(f"\nFile sizes:")
    print(f"  Total:   {total_size / (1024**2):.1f} MB")
    print(f"  Average: {avg_size / 1024:.1f} KB")
    print(f"  Min:     {min(sizes) / 1024:.1f} KB")
    print(f"  Max:     {max(sizes) / (1024**2):.2f} MB")

    comp_counts = Counter(m["company"] for m in metadata)
    print(f"\nTop 10 companies by document count:")
    for comp, count in comp_counts.most_common(10):
        print(f"  {comp:45s} {count:5d}")

    print(f"\nMetadata saved to: {METADATA_PATH}")
    print(f"Reports saved to:  {OUTPUT_DIR}/")

if __name__ == "__main__":
    metadata, companies = extract_all()
    print_statistics(metadata, companies)
```

Додаток Г

(довідковий)

Код модуля `config.py` – конфігурація моделей та промптів

```
EMBEDDING_MODELS = {
    "multilingual-e5-large": "intfloat/multilingual-e5-large",
    "bge-large-en-v1.5": "BAAI/bge-large-en-v1.5",
    "all-mpnet-base-v2": "sentence-transformers/all-mpnet-base-v2",
}

LLM_CONFIGS = {
    "llama3.1:8b": {
        "provider": "ollama",
        "model": "llama3.1:8b",
    },
    "mistral": {
        "provider": "ollama",
        "model": "mistral",
    },
    "claude-sonnet": {
        "provider": "anthropic",
        "model": "claude-sonnet-4-20250514",
        "api_key": "<ANTHROPIC_API_KEY>",
    },
    "gpt-4o": {
        "provider": "openai",
        "model": "gpt-4o",
        "api_key": "<OPENAI_API_KEY>",
    },
}

CHROMA_BASE_DIR = "chroma_db"
DEFAULT_TOP_K = 5

PRICING = {
    "claude-sonnet-4-20250514": {"input": 3.0, "output": 15.0},
    "gpt-4o": {"input": 2.5, "output": 10.0},
}
```

```
SYSTEM_PROMPT = """You are an expert financial analyst specializing in European corporate reports.

Instructions:
- Answer questions based ONLY on the provided context from financial reports.
- Always cite your sources: mention the company name, year, and report section.
- If the context doesn't contain enough information to answer the question, clearly state this.
- Provide specific numbers, dates, and facts when available.
- Compare data across companies or years when relevant.
- Answer in the same language as the question.
"""
```

Додаток Д

(довідковий)

Код модуля `benchmark_queries.py` – тестові запити

```
BENCHMARK_QUERIES = [

    {
        "query": "What was the total revenue of Siemens in 2023?",
        "category": "factual",
    },
    {
        "query": "What is the EBITDA margin reported by ASML in their latest annual report?",
        "category": "factual",
    },
    {
        "query": "How many employees does SAP have according to their most recent report?",
        "category": "factual",
    },

    {
        "query": "Describe the main risk factors mentioned by Allianz in their annual report.",
        "category": "descriptive",
    },
    {
        "query": "What is the sustainability strategy of TotalEnergies?",
        "category": "descriptive",
    },
    {
        "query": "Summarize the key business segments of Nokia.",
        "category": "descriptive",
    },

    {
        "query": "Compare the revenue growth of SAP and Siemens over the reported periods.",
        "category": "comparative",
    },
    {
        "query": "Which company has the highest profit margin among the French companies in the dataset?",
        "category": "comparative",
```

```
    },
    {
        "query": "How do dividend policies differ between German and Dutch companies?",
        "category": "comparative",
    },

    {
        "query": "What are the main drivers of revenue growth for LVMH?",
        "category": "analytical",
    },
    {
        "query": "How has the debt-to-equity ratio changed for Equinor over the reported years?",
        "category": "analytical",
    },
    {
        "query": "What impact did inflation have on operating costs according to the Spanish companies?",
        "category": "analytical",
    },

    {
        "query": "What are the common ESG reporting themes across Nordic companies?",
        "category": "cross-company",
    },
    {
        "query": "Which sectors are most represented among the Italian companies in the dataset?",
        "category": "cross-company",
    },
    {
        "query": "Summarize capital expenditure trends across the largest European banks.",
        "category": "cross-company",
    },
]
```