Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: « **МУЛЬТИМОДАЛЬНИЙ RAG З ВИКОРИСТАННЯМ ТЕКСТОВИХ ТА ВІЗУАЛЬНИХ ДАНИХ** »

Виконав: студент 4-го року навчання,

Освітньої програми «Інженерія програмного забезпечення», 121

Шевченко Михайло Григорович

Керівник Андрощук М.В.,_____

старший викладач

Рецензент _____

(прізвище та ініціали)

Кваліфікаційна робота захищена з оцінкою _____

Секретар ЕК _____

«____» _____ 20____ р.

Київ – 2025

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,

доцент, к. ф-м. н.

_____ Гороховський С.С.

(підпис)

„____”_____2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студенту Шевченку Михайлу Григоровичу факультету інформатики 4-го курсу
ТЕМА: Мультимодальний RAG з використанням текстових та візуальних даних
Зміст ТЧ до кваліфікаційної роботи:

Індивідуальне завдання
Календарний план виконання роботи
Анотація
Вступ
1 RAG та його можливості. Мультимодальний RAG
2 Вибір інструментів та засобів розробки
3 Створення мультимодальної RAG-системи
4 Тестування розробленої системи
Висновки
Список використаних джерел

Дата видачі „___” _____ 2024 р. Керівник _____
(підпис)

Завдання отримав _____
(підпис)

Тема: Мультимодальний RAG з використанням текстових та візуальних даних

Календарний план виконання роботи:

№ п/п	Назва етапу	Термін виконання етапу	Примітка
1.	Отримання завдання на кваліфікаційну роботу.	07.10.2024	
2.	Огляд літератури за темою роботи.	31.10.2024	
3.	Написання теоретичної частини роботи.	30.11.2024	
4.	Розробка практичної частини.	31.01.2025	
5.	Тестування практичної частини.	28.02.2025	
6.	Написання текстової частини роботи.	30.04.2025	
7.	Створення презентації.	14.05.2025	
8.	Передзахист роботи.	21.05.2025	
9.	Захист роботи.	04.06.2025	

Студент _____

Керівник _____

“_____”_____

ЗМІСТ

АНОТАЦІЯ

У роботі розглянуто техніку мультимодального Retrieval-Augmented Generation (з англ. генерація з доповненою вибіркою, RAG) для покращення результатів роботи систем генерування контенту. Проведено аналіз сутності RAG і мультимодального RAG, їхніх переваг, архітектур, а також популярних інструментів для розробки мультимодальних RAG-систем. Було розроблено мультимодальну RAG-систему для аналізу та інтерпретації рентгенівських знімків грудної клітки та їхніх звітів. Розроблену систему протестовано у різних конфігураціях за низкою показників. Результати тестування засвідчили значне покращення якості генерації контенту при використанні мультимодального RAG порівняно з його відсутністю.

Ключові слова: RAG, мультимодальний RAG, Retrieval-Augmented Generation, LLM, Large Language Models, Python, PyTorch, CLIP, BiomedCLIP, LLaVA, LLaVA-Med

ВСТУП

З розвитком цифрових технологій дедалі активніше в повсякденне та професійне життя інтегруються різні системи генерування контенту, які створюють інформацію відповідно до запитів користувачів. Найчастіше для цього використовують різні великі мовні моделі (англ. Large Language Models, LLM), але також застосовують системи для генерування зображень, програмного коду тощо. Такі системи називають генераторами. Однак результати генераторів часто можуть не відповідати запитам користувачів: вони можуть містити неправдиву, розпливчасту або нечітку інформацію або не використовувати актуальні дані для створення результату. Тому існує потреба у вдосконаленні генераторів, їх зручному використанні для різних предметних областей і забезпеченні доступу до актуальної інформації під час генерації.

Для вирішення цих завдань використовують техніку Retrieval-Augmented Generation [1] (з англ. генерація з доповненою вибіркою, далі — RAG). RAG доповнює запити користувача до генератора релевантними даними відповідно до запиту. Основна ідея RAG полягає у знаходженні релевантних даних до запиту користувача та передачі генератору не лише самого запиту, а й знайденої додаткової інформації. RAG використовує дані зі сховища даних. Як сховище диних можуть використовуватися різні джерела: бази даних, каталоги, файли, Інтернет тощо. Вибір джерел даних для RAG-системи залежить від мети її створення та предметної області. Наприклад, якщо потрібно, щоб генератор краще працював з медичними даними, як дані для доповнення можна використовувати медичні звіти.

Більшість реалізацій RAG працюють виключно з текстовими даними [2]. Проте в реальному світі інформація представлена у вигляді різноманітних типів даних: текст, зображення, відео, аудіо тощо. Нетекстові типи даних часто містять важливу інформацію, яку неможливо точно передати словами без втрат змісту чи контексту. RAG, який працює з двома або більше типами даних, називають

мультимодальним RAG [3]. Використання декількох типів даних дозволяє підвищити релевантність знайдених даних, і як наслідок, вичерпність і повноту згенерованих відповідей генератором.

RAG-системи зазвичай створюють для певної предметної області для покращення роботи генератора в певних сценаріях використання. В роботі для реалізації мультимодального RAG було вибрано медичну предметну область, зокрема аналізу та інтерпретації рентгенівських знімків грудної клітки та їхніх звітів. Головним фактором вибору такої предметної області є достатня кількість доступних наборів даних у відкритому доступі [4][5][6], що не завжди характерно іншим предметним областям.

Виходячи з описанного вище, метою даної роботи є розробка мультимодальної RAG-системи для аналізу та інтерпретації рентгенівських знімків грудної клітки та їхніх звітів із використанням сучасних інструментів і технологій, а також аналіз і тестування цієї системи за різними показниками. Об’єктом дослідження є техніка мультимодального RAG і мультимодальні RAG-системи. Предметом дослідження є методи, моделі та програмні засоби побудови мультимодальних RAG-систем, що використовують текстові та візуальні дані для аналізу та інтерпретації рентгенівських знімків грудної клітки та їхніх звітів.

Мета роботи зумовила наступне наукове завдання:

1. Дослідити можливості та функції RAG і мультимодального RAG.
2. Виконати аналіз інструментів та програмних засобів для створення мультимодальних RAG-систем з використанням візуальних та текстових даних.
3. Розробити мультимодальну RAG-систему для аналізу та інтерпретації рентгенівських знімків грудної клітки та їхніх звітів.
4. Протестувати розроблену систему за різними показниками.

Робота складається з чотирьох розділів.

Перший розділ присвячено аналізу можливостей RAG-систем, зокрема їхнім перевагам, архітектурам та компонентам. Також розглянуто особливості

мультимодального RAG, особливо, приділено увагу мультимодальному RAG з використанням текстових та візуальних даних.

У другому розділі оглянуто й охарактеризовано популярні інструменти та програмні засоби для розробки мультимодальних RAG-систем з використанням текстових та візуальних даних.

Третій розділ присвячено розробці мультимодальної RAG-системи з використанням текстових та візуальних даних для аналізу та інтерпретації рентгенівських знімків грудної клітки та їхніх звітів. У розділі описано покроковий процес створення застосунку, наведено приклади роботи системи та зазначено використані інструменти.

У четвертому розділі наведене детальне тестування різних конфігурацій створеної системи за різними показниками. Була визначена найкраща конфігурація системи. Результати тестування порівнювалися з генератором, що працює без використання мультимодального RAG.

Результатом роботи є консольне програмне забезпечення з використанням техніки мультимодального RAG для аналізу та інтерпретації рентгенівських знімків грудної клітки та їхніх звітів. Зокрема, застосунок здатен генерувати відповідь на поставлене запитання користувачем до вхідного зображення та генерувати радіологічний звіт на вхідне зображення.

РОЗДІЛ 1: RAG ТА ЙОГО МОЖЛИВОСТІ. МУЛЬТИМОДАЛЬНИЙ RAG

1.1 RAG та його складові

Retrieval-Augmented Generation (з англ. генерація з доповненою вибіркою, далі — RAG) — це підхід, який поєднує механізми інформаційного пошуку та генерації контенту. Основна ідея RAG полягає в покращенні результатів систем генерації контенту шляхом збагачення вхідних даних додатковою інформацією, отриманою з зовнішніх джерел (баз даних, вебресурсів, документів тощо).

Системи, що реалізовують RAG, складаються з трьох основних частин:

- генератор;
- ретривер;
- сховище даних.

Генератор (англ. Generator) — це система, призначена для створення даних, які відповідають запитам користувачів на основі визначених алгоритмів або моделей. Наприклад, відповідь на питання користувача або створення зображення за вхідними даними, тощо. Найчастіше використовують генератори, які добре вирішують завдання обробки природної мови (англ. NLP — Natural Language Processing), наприклад: розуміння природного тексту, генерація, переклад, резюмування тощо. Як генератор в більшості випадків використовують нейронні мережі завдяки їхній здатності навчатися на великих обсягах даних та знаходити приховані закономірності в них. Також як генератор можуть використовуватися різні стохастичні моделі [7]. Ефективний генератор повинен уміти генерувати природний та точний результат, який має відповідати контексту запиту користувача, у тих формах даних, яких вимагає система.

Ретривер (англ. Retriever) — це система, головним завданням якої є швидке знаходження *n* найрелевантніших документів зі сховища даних відповідно до запиту користувача. Ці дані доповнюють запит користувача і далі оброблюються генератором для створення кінцевої відповіді.

Сховище даних — це сховище, яке містить різні документи, що використовує ретривер для релевантного пошуку. Зазвичай сховище даних налічує багато документів. Як сховище може використовуватися великий перелік файлів, база даних або сторонні API (Application Programming Interface, з англ. прикладний програмний інтерфейс) для доступу до даних. Наприклад, пошукова система Google. Часто використовують спеціальні бази даних, які, крім даних, також зберігають спеціальні індекси або просторові вектори, щоб швидко знаходити релевантні документи.

1.2 Переваги систем з RAG

Для того щоб генератор на основі нейронних мереж підтримував актуальні дані його треба регулярно донавчати. Це дуже трудомісткий процес, адже сучасні мережі містять десятки мільйонів параметрів для навчання, що займає багато часу. Використовуючи RAG, є можливість надавати генератору актуальні дані без необхідності донавчання [1]. Наприклад, користувач хоче дізнатися про курс долара США. Він надає відповідний запит системі генерування контенту. Очевидно, генератор самостійно не володіє актуальною інформацією про поточний курс. Система за допомогою ретривера знаходить курс долара США у сховищі даних або отримує за допомогою сторонніх API та надає цю інформацію разом зі запитом генератору, який уже має всі дані, щоб коректно відповісти на запит користувача.

Іншим сценарієм використання є потреба, щоб система генерування контенту була налаштована під конкретну предметну область. Наприклад, система повинна генерувати змістовні, обґрунтовані та релевантні відповіді на запити з медицини, програмування, продукту якоїсь компанії тощо. Зазвичай налаштування або створення генератора під конкретну предметну область дуже тривалий і складний процес. Використання ретривера, сховища даних та існуючого генератора є ефективнішим способом створення систем генерації

контенту для конкретних предметних областей [2]. Завдяки цьому не треба створювати новий генератор, а можна використати існуючий, натренований на загальних даних. Окільки генератор уміє генерувати релевантні та змістовні відповіді на загальні питання і завдяки ретриверу отримує предметно-орієнтовані дані, він з високою точністю та релевантністю справляється із завданнями конкретної предметної області.

Незважаючи на здатність нейронних мереж генерувати контекстно відповідні дані, вони можуть продукувати відповіді з галюцинаціями [8]. Галюцинації в нейронних мережах — це неправдива або оманлива інформація, згенерована мережею, яка подається як факт. Поки не розроблено способів подолання цього явища і не існує чіткого пояснення його природи, проте використання RAG значно зменшує вплив галюцинацій на відповіді генератора.

Також використання RAG надає системі генерування контенту можливість надавати користувачеві список джерел, на основі яких був згенерований контент. Це збільшує довіру користувачів до згенерованих даних, забезпечує прозорість процесу та дає їм змогу ознайомитися з джерелами для перевірки достовірності або отримання додаткової інформації.

1.3 Типи ретриверів

Як ретривер можуть використовуватися різні методи алгоритми й системи. Загалом їх можна поділити на чотири типи [2].

Перший тип ретриверів — це стандартні алгоритми інформаційного пошуку. Найчастіше всього використовують алгоритм TF-IDF [9] та функцію ранжування BM25 [10]. Алгоритм TF-IDF створює вектор частоти слів для кожного документа, щоб здійснювати ефективний пошук найрелевантніших документів за допомогою функції BM25.

Другий тип — це ретривери на основі нейронних мереж [11][12]. Нейронні мережі зазвичай краще справляються з ранжуванням документів, ніж

попередній тип, завдяки здатності знаходити приховані закономірності. Нейронна мережа для кожного документу сховища даних створює просторовий вектор, який часто називають індексом. Вхідний запит користувача перетворюються в індекс, і пошук релевантних документів здійснюється за допомогою косинусної подібності (англ. cosine similarity) [13] індексів. Також нейронні мережі можуть обробляти не тільки текстову інформацію, а й зображення, відео, аудіо тощо.

Третій тип — це використання сторонніх API як ретривера. Найчастіше використовується API великих пошукових систем, таких як Google або Bing. Вони надають доступ до великої кількості інформації та застосовують різноманітні попередньо налаштовані алгоритми пошуку, що робить їх вдалим варіантом для загальнох предметних областей.

Четвертий тип — це комбіновані ретривери, які використовують кілька попередніх типів одночасно.

1.4 Способи поєднання генератора та ретривера

Існує декілька способів, як ретривер може бути пов’язаний із генератором. Нижче подано три основні з них [2].

Перший спосіб [11]] — послідовний з одним циклом. При такому способі вхідний запит оброблюється спочатку ретривером, а потім вже генератором. Спочатку ретривер знаходить *n* релевантних документів. Потім ці документи об’єднуються з вхідним запитом за певним алгоритмом — часто вони просто додають у кінець вхідних даних. Далі ця агрегована інформація оброблюється генератором, який генерує кінцеву відповідь користувачеві.

Другий спосіб [14] — послідовний із багатьма циклами. Він схожий на попередній, але головна відмінність полягає у тому, що згенерована відповідь може кілька разів подаватися на вхід ретриверу, а потім — знову генератору. У

процесі генеруванні часто використовують також попередні відповіді системи, що робить такий спосіб ідеальним для підтримки діалогу з користувачем.

Третій спосіб [15] — паралельний. Ретривер і генератор обробляють вхідний запит незалежно один від одного. Потім результати їхньої роботи об'єднується. Цей спосіб є швидшим за інші, а також ретривер можна залучати лише за потреби.

1.5 Мультимодальний RAG

Мультимодальний RAG — це RAG, який використовує не один, а кілька типів даних для ретривера й генератора одночасно, наприклад: текст, зображення, відео, аудіо тощо. Застосування кількох типів даних часто надає генератору більше інформації, завдяки чому генеруються більш вичерпні та релавантні відповіді на запит користувача [3].

Використання декількох типів даних для генерації відповіді є важливим для багатьох галузей. Наприклад, у медичній сфері [16] — для надання зображень або відео з симптомами хвороб, рентгенівськими знімками тощо. Хрошим прикладом застосування мультимодального RAG є промислова галузь [17], де дані часто подаються не лише у вигляді текстових документів, а й брошурами, схемами, діаграмами тощо. Для запитів загальної тематики використання мультимодального RAG також підвищує коректність відповіді, особливо у випадках, коли інформація міститься лише в нетекстових даних. Наприклад, мультимодальний RAG дасть змогу відповісти на запит типу: «Що можна побачити на балконі Білого дому на Різдво?». Це можливо тому, що відповідна інформація про предмети на балконі міститися і в текстових, і у візуальних даних, які доповнюють одне одного [18].

Ретривер у мультимодальному RAG може знаходити релевантні дані різних типів або лише певного типу. Також ретривери в таких системах поділяють на два підтипи.

Перший підтип [17] працює з сховищем даних, що містить різні типи даних. Кожен не текстовий документ перетворюється на його текстовий опис, за яким створюється індекс, щоб усі документи знаходилися в одному латентному просторі (абстрактний математичний простір, у якому дані представлені у вигляді векторів з певною кількістю вимірів).

Другий підтип [18] не перетворює кожен нетекстові документи на текстові описи, а створює індекси безпосередньо з оригінальних документів. Для успішного релевантного пошуку всі індекси створюються в єдиному латентному просторі за допомогою спеціальних алгоритмів і систем, наприклад, CLIP [19].

Генератор у мультимодальному RAG також може бути кількох підтипів [3]. Вони відрізняються за типами даних, які обробляють на вході та генерують на виході:

- приймає один тип даних на вході, генерує відповідь того самого типу;
- приймає кілька типів даних на вході, генерує відповідь одного типу;
- приймає один тип даних на вході, генерує відповідь кількох типів;
- приймає кілька типів даних на вході, генерує відповідь кількох типів.

Поєднання різних підтипів генераторів із різними типами ретриверів залежить від завдань RAG-системи та типів даних, з якими вона працює.

1.6 Особливості мультимодального RAG з використанням текстових та візуальних даних

Мультимодальний RAG із використанням текстових та візуальних даних — це найпоширеніший вид мультимодального RAG. Він застосовує тексти, зображення та відео для генерації відповіді. Найчастіше використовують саме зображення, а не відео, оскільки робота з відео є значно складнішою через великий обсяг, необхідність обробки послідовностей кадрів, а також складність виокремлення релевантної інформації з часових даних. Застосування мультимодального RAG із використанням текстових та візуальних даних часто

підвищує коректність і релевантність відповідей генераторів [18], оскільки велика кількість інформації зберігається саме у візуальному вигляді й не має текстового відповідника. Наприклад, така інформація, як просторові відношення між об'єктами, емоційний контекст, специфічні деталі зображень (текстури, кольори, форми), а також складні візуальні структури (графіки, діаграми, медичні знімки), часто не мають точного текстового відповідника або потребують значного спрощення при описі словами.

На сьогодні мультимодальний RAG із використанням текстових та візуальних даних використовують для трьох основних завдань [3].

Перше завдання — генерування опису на вхідне зображення. Для виконання цього завдання використовують пари «картинка — опис» як дані для сховища [20]. У цьому підході ретривер використовує індекси створені на основі зображень, а на вхід генератору подається зображення та описи релевантних зображень із сховища даних.

Друге завдання — генерування відповіді на вхідне питання до системи, використовуючи візуальні дані. Для виконання цієї задачі ретривер знаходить релевантні візуальні та текстові документи зі сховища даних, використовуючи їхні індекси, що знаходяться в одному латентному просторі [18]. Потім релевантні дані разом із запитом подаються на вхід генератору, який генерує текстову відповідь. Релевантні візуальні дані можуть перетворюватися на текстові описи, якщо генератор приймає лише текстові дані.

Третє завдання — генерування відповіді на поставлене питання до вхідного зображення. Для виконання цієї задачі [21], ретривер перетворює вхідне зображення на індекс, який знаходиться в одному латентному просторі, що й документи в сховищі даних. Потім ретривер використовує цей індекс для знаходження релевантних документів до цього зображення. Після цього знайдені документи разом з вхідним питанням та зображенням подаються на вхід генератору, який вміє працювати з текстовими та візуальними даними.

РОЗДІЛ 2: ВИБІР ІНСТРУМЕНТІВ ТА ЗАСОБІВ РОЗРОБКИ

2.1 Критерії вибору наборів даних

Для створення мультимодального RAG із використанням візуальних і текстових даних потрібно застосовувати набори даних, які містять як текстову інформацію, так і зображення. Це можуть бути дані різних форматів. Наприклад, файли з розширенням .pdf або .docx, що включають як текст, так і зображення. Або це може бути просто набір файлів з розширенням .txt й зображень, які знаходяться в різних директоріях. Співвідношення зображень і текстових даних може бути різним, але для того, щоб максимально використати переваги обох типів інформації, важливо забезпечити різноманітність зображень і тексту. Хороше співвідношення зображень і текстових даних проявляється в тому, що мультимодальний RAG використовує як зображення, так і текст під час кожного запиту.

Зазвичай RAG створюють для певної предметної області, тому слід використовувати відповідні набори даних. Наприклад, це може бути документація певного програмного продукту, якщо створюється помічник для орієнтування в ньому, або рентгенівські знімки зі звітами до них. Якщо RAG створюється для роботи з загальною предметною областю, то треба використовувати відносно великий (більше 100 мільйонів документів) набір довільних даних з різних галузей і сфер [18].

2.2 Вибір мови програмування

Для створення RAG системи не існує обмежень щодо вибору мови програмування. Однак, оскільки системи з використанням RAG найчастіше використовують великі мовні моделі як генератор та різні нейронні мережі в ретривері, слід обирати мову програмування, яка має хорошу підтримку для

розробки і використання нейронних мереж. Сьогодні майже кожна популярна мова програмування має бібліотеки і фреймворки для створення нейронних мереж, однак найбільше можливостей для розробки надає мова програмування Python. Також Python використовується в більшості сучасних дослідженнях нейронних мереж. Великі інформаційно-технологічні компанії, зокрема Google і Meta, розробляють бібліотеки і фреймворки саме для Python. Також більшість відомих великих мовних моделей розроблені за допомогою цієї мови. Наприклад, LLaVA [22] або DeepSeek [23].

2.3 Вибір бібліотеки машиного навчання

Існує безліч бібліотек машинного навчання для мови програмування Python. Проте найбільш популярними серед них є TensorFlow [24] та PyTorch [25].

TensorFlow — це бібліотека з відкритим вихідним кодом, розроблена компанією Google та представлена 9 листопада 2015 року. Вона використовується для чисельних обчислень та побудови глибинних нейронних мереж. TensorFlow використовує статичний обчислювальний граф, що не дозволяє змінювати структуру нейронних мереж під час виконання коду. Крім того, TensorFlow використовує доволі низькорівневі абстракції опису моделей, що підвищує поріг входу для новачків і збільшує обсяг коду. Наразі разом із TensorFlow майже завжди використовують бібліотеку Keras [26], яка надає зручний інтерфейс користування. Із 2017 року Keras інтегровано до складу TensorFlow.

PyTorch — це бібліотека машинного навчання з відкритим вихідним кодом на основі бібліотеки Torch. Вона застосовується для різноманітних задач глибинного навчання, таких як комп'ютерний зір, обробка природної мови тощо. Вперше PyTorch з’явився в жовтні 2016 року. Підтримкою і розвитком PyTorch в основному займається компанія Meta AI. PyTorch підтримує динамічний

обчислювальний граф, що спрощує налагодження та експерименти під час тренування моделей. Також PyTorch надає велику кількість попередньо створених модулей. Наприклад: TorchVision, TorchText, TorchAudio тощо.

TensorFlow і PyTorch надають можливості виконувати паралельні обчислення як на центральному процесорі, так і на графічному процесорі. Виконання обчислень на графічному процесорі прискорює їхню швидкість виконання, тому можливість використання графічного процесору є критично важливою. Для виконання обчислень на графічному процесорі TensorFlow і PyTorch використовують CUDA Toolkit. CUDA Toolkit — це платформа, яка дозволяє виконувати паралельні обчислення на графічних процесорах компанії Nvidia.

Для того, щоб виконувати обчислення на графічних процесорах за допомогою TensorFlow або PyTorch, треба виконати декілька вимог:

1. Мати графічний прискорювач від компанії Nvidia.
2. Треба завантажити й встановити CUDA Toolkit відповідної версії з офіційного сайту Nvidia [27].
3. Треба встановити TensorFlow або PyTorch сумісної версії згідно з версією всановленої CUDA Toolkit і операційної системи. Наприклад, завантажити PyTorch або знайти команду для його всановлення можна на офіційному сайті [28].

TensorFlow і PyTorch є хорошим вибором бібліотеки машинного навчання для створення мультимодального RAG. Проте, PyTorch є більш популярним вибором для проведення досліджень і прототипування, про це свідчить інформація про використання бібліотек на GitHub: близько 511 тис. використань у TensorFlow [29] проти 745 тис. використань у PyTorch [30]. Основними перевагами PyTorch для дослідницьких задач є динамічний обчислювальний граф та простіший і зрозуміліший інтерфейс використання.

2.4 Інструменти для створення ретривера

Ретриверу в мультимодальному RAG з використанням текстових та візуальних даних необхідно якось порівнювати зображення й текстові дані на їхню релевантність. На сьогодні найпопулярнішим і фактично єдиним способом такого порівняння та співставлення є нейронні мережі [2].

Мультимодальні ретривери зазвичай використовують спеціальні нейронні моделі, поєднанні певним чином, для створення векторів з оригінальних даних. Ці вектори мають фіксовану кількість вимірів і часто називаються індексами. Для зберігання цих індексів і виконання пошуку за ними використовують векторні бази даних. Також крім основної моделі, можуть використовують додаткові нейронні моделі або способи індексування та маркування даних.

2.4.1 Вибір моделей і способів для створення індексів

Існує велика кількість способів створення індексів з текстових та візуальних даних за допомогою нейронних мереж. Ці способи можна поділити на три основні типи [3].

Перший тип [19] — використовує дві окремі мережі для створення індексів. Одна використовується для кодувань зображень, а інша мережа для кодування текстів. Ці мережі навчаються так, щоб зображення і відповідні їм тексти мали максимальну косинусну подібність.

Другий тип [17] — використовує одну мережу для створення індексів з певного типу даних. Дані іншого типу перетворюється в тип, яку приймає на вхід мережа. Це перетворення найчастіше відбувається за допомогою інших нейронних мереж. Як правило, зображення перетворюють у текст, а не навпаки.

Третій тип [31] — використовує одну мережу для створення індексів, але приймає на вхід зображення разом з текстовими даними. Такі моделі мають складну архітектуру, тому їх важче навчати, ніж моделі попередніх типів.

Водночас такі моделі зазвичай не використовують для окремої індексації зображень чи текстів.

Найкраще для мультимодальної RAG-системи підходить перший тип, оскільки він досить простий у реалізації та навчанні. Крім того, такий тип може ефективно індексувати окремо зображення і текстові дані, і їхні індекси будуть знаходитись в одному латентному просторі. Найвідомішим та найпоширенішим прикладом такого способу індексування — це модель CLIP [19] і різні її модифікації.

CLIP (англ. Contrastive Language–Image Pre-training) — це модель із двома кодувальниками, яка використовує контрастивне навчання (англ. contrastive learning) [32], розроблена компанією OpenAI. Модель навчена на великій кількості пар «зображення–текст» (близько 400 млн). У процесі навчання два окремі кодувальники, візуальний і текстовий, навчаються проектувати вхідні дані в єдиний латентний векторний простір.

2.4.2 Вибір векторної бази даних

Для зберігання індексів даних і швидкого знаходження серед них *n* документів, які мають найбільшу косинусну подібність [33], використовують векторні бази даних. Основними критеріями для векторної бази даних для мультимодального RAG є: можливість зберігати дані на диск, підтримка індексів різної розмірності, швидкий пошук і можливість використання метаданих (даних, які додатково описують документи).

Для мови програмування Python існує багато векторних баз даних. Усі вони мають свої власні особливості. Наприклад, деякі краще підходять для роботи з великими обсягами документів, а інші можуть працювати в оперативній пам’яті. Загалом вони дуже схожі між собою і заезпечують схожу функціональність. Серед них ChromaDB [34] є однією з найпопулярніших векторних баз даних для прототипування та створення RAG-систем.

ChromaDB — це векторна база даних із відкритим програмним кодом для зберігання та пошуку індексів. Вона спеціально розроблена і оптимізована для потреб RAG-систем. ChromaDB дозволяє індексувати сотні мільйонів даних із підтримкою як точного, так і наближеного пошуку. Крім того, вона забезпечує роботу з метаданими та зберігання даних як в оперативній пам’яті, так і на диску.

2.4.3 Інші додаткові моделі та способи

Для покращення пошуку релевантних документів RAG системи можуть використовувати додаткові способи для виділення метаданих з основних даних [2]. Існує багато різних підходів для цього. У мультимодальному RAG часто застосовують моделі, які класифікують зображення за різними категоріями або виділяють на них певні об’єкти. Наприклад, для створення RAG-системи, що працює зі стравами та рецептами, можна використати модель для класифікації інгредієнтів за зображенням страви. Після цього RAG-система буде визначати інгредієнти для всіх вхідних зображень і знаходити релавантні документи, що мають такі самі інгредієнти, що підвищує точність. Хоча виділення метаданих із зображень є більш поширеним сценарієм користування, можна виділяти метадані з тексту також. Наприклад, перетворювати неструктурований текст на заздалегідь визначену структуровану форму.

Найчастіше моделі для виділення метаданих не створюють з нуля, а використовують існуючі моделі, попередньо натреновані на великій кількості даних. Такі моделі донавчають на потрібних прикладних даних із використанням техніки передавального навчання (англ. transfer learning) [35]. Передавальне навчання — це техніка машинного навчання, яка зосереджена на зберіганні знань, отриманих під час розв'язання однієї задачі, та застосуванні їх до іншої, але пов'язаної задачі.

2.5 Вибір генератора

Генератором для мультимодального RAG може бути майже будь-яка велика мовна модель. Основні критерії під час вибору генератора — це безперебійний доступ до нього та його відповідність певній предметній області. Найкраще, щоб генератор був мультимодальним, тобто міг приймати на вхід не тільки текст, а й зображення також. Це підвищує точність генерації та дозволяє уникнути перетворювань зображень в текст.

Для мультимодального RAG можна використовувати сучасні популярні чат-боти, наприклад: ChatGPT, DeepSeek, Gemini та інші. Їхнє використання спрощує створення RAG-системи, оскільки відпадає потреба у локальному налаштуванні генератора. Проте їхнє використання не надає повний контроль над генератором і не гарантують стовідсотковий безперебійний доступ. Крім того, користування API чат-ботів може бути досить дорогим.

Щоб уникнути проблем використання чат-ботів, можна використовувати великі мовні моделі з відкритим програмним кодом, які запускаються локально. Однією з найкращих таких моделей є мультимодальна велика мовна модель LLaVA 1.5 [36]. Вона показує хороші і часто кращі результати ніж її конкуренти [36]. Також її можна запустити на відносно слабких графічних прискорювачах завдяки техніці квантинізації [37]. Наприклад, на RTX 2060 6Gb.

РОЗДІЛ 3: СТВОРЕННЯ МУЛЬТИМОДАЛЬНОЇ RAG-СИСТЕМИ

3.1 Вибір предметної області. Функціональність створюваної RAG-системи

Як було зазначено в попередніх розділах мультимодальні RAG-системи найчастіше створюються для певної предметної області, щоб покращити роботу генератора в певних сценаріях використання. Хоча мультимодальні RAG-системи можуть створюватися для роботи з загальними даними або охоплювати декілька предметних областей одночасно, це вимагає великого об’єму даних і потужних обчислюваних ресурсів, які не були доступні під час виконання цієї роботи.

В роботі для реалізації мультимодального RAG було вибрано медичну предметну область, зокрема аналізу та інтерпретації рентгенівських знімків грудної клітки та їхніх звітів. Медична предметна область є доволі популярним напрямком створення RAG. Вибір такої предметної області надає широкі можливості для створення різних RAG систем та дозволить оцінити як використання мультимодального RAG покращує роботу існуючих систем. Іншим фактором вибору такої предметної області є достатня кількість доступних наборів даних у вільному доступі, що не завжди характерно іншим предметним областям. Найчастіше, такі набори даних включають певну кількість рентгенівських знімків грудних кліток та звітів у текстовому форматі для кожного знімку.

Мультимодальна RAG-система, створена в роботі, підтримує два сценарії використання. Перший сценарій використання — це генерація звіту на вхідний рентгенівський знімок, щодо наявності паталогій, хвороб та іншого. Інший сценарій використання — це генерація відповіді на вхідне питання щодо певного рентгенівського знімку. Ці сценарії є типовими для мультимодального RAG відповідно до розділу 1. Також такі сценарії використання є стандартними для

використання в медичній сфері. Для реалізації даних сценаріїв використання, ретривер буде знаходити релевантні звіти до вхідного зображення. Потім знайдені звіти будуть разом з вхідним зображенням та питанням надаватися генератору для генерації звіту або відповіді на вхідне питання.

3.2 Використані набори даних

Під час розробки мультимодальної RAG-системи з використанням текстових та візуальних даних для роботи рентгенівськими знімками грудної клітки та звітами до них були використанні такі набори даних: MIMIC-CXR [4], MIMIC-CXR-JPG [38], CheXpert [5] та IU-Xray [6].

MIMIC-CXR — це великий загальнодоступний набір даних рентгенівських знімків грудної клітки у форматі DICOM із довільними текстовими радіологічними звітами. Набір даних містить 377110 зображень, що відповідають 227835 рентгенографічним дослідженням, проведеним у медичному центрі Beth Israel Deaconess у Бостоні, Массачусетс. MIMIC-CXR-JPG — це перероблений MIMIC-CXR, в якому картинки в форматі JPG, але цей датасет не містить звіти на відміну від MIMIC-CXR. В роботі були використані зображення з MIMIC-CXR-JPG та звіти з MIMIC-CXR. Обидва набори даних (далі — MIMIC-CXR) можна офіційно завантажити на PhysioNet [39]. Для того, щоб мати можливість завантажити ці датасети треба пройти спеціалізований курс та верифікувати особу через платформу PhysioNet.

CheXpert — це великий публічний набір даних для інтерпретації рентгенівських знімків грудної клітки, що складається з 224316 рентгенівських знімків грудної клітки 65240 пацієнтів. В наборі даних зібрані дані пацієнтів, які проходили рентгенологічне обстеження грудної клітки в Стенфордській лікарні, в період з жовтня 2002 року по липень 2017 року як у стаціонарних, так і в амбулаторних центрах, разом із відповідними радіологічними звітами. Кожне зображення у наборі даних охарактеризовано за певною кількістю ознак. У

роботі використовувалося інформація про сторону тулуба, з якої було зроблено рентгенівський знімок: спереду або сбоку. Також використовувалося характеристика зображень серед 14 клінічних ознак. Кожна клінічна ознака в наборі даних позначена як присутня, відсутня, не вказана або не визначена.

IU-Xray — це набір рентгенівських зображень грудної клітки в поєднанні з відповідними діагностичними звітами. Набір даних містить 7470 пар зображень і звітів. Даний набір даних був використаний в роботі лише для тестування системи.

3.3 Створення ретриверу

Рентгенівські знімки грудей бувають двох типів: передні і бічні. Передні — зроблені спереду грудної клітки (див. рисунок 3.3.1). Бічні — зроблені збоку грудної клітки (див. рисунок 3.3.2). Передні і бічні знімки досить сильно відрізняються один від одного, бо на передніх можуть бути видні паталогії і клінічні ознаки, яких не видно на бічних і навпаки. Через це було вирішено реалізувати дві моделі, які створюють індекси (далі — моделі індексації): одна модель буде працювати з передніми зображеннями і їхніми звітами, а інша з бічними і звітами. Для того, щоб ретривер міг ідентифікувати, яке зображення — переднє, а яке — бічне, було створено додаткову модель для ідентифікації сторони (далі — класифікатор сторін). Також для того, щоб ще більше покращити ретривер, щоб він знаходив найбільш релевантні звіти було створено модель для створення метаданих, а саме інформацію про наявність певної клінічної ознаки на кожному зображенні (далі — класифікатор хвороб).

Рисунок 3.3.1 — Приклад рентгенівського знімку зробленого спереду тулуба. Знімок взятий з набору даних MIMIC-CXR

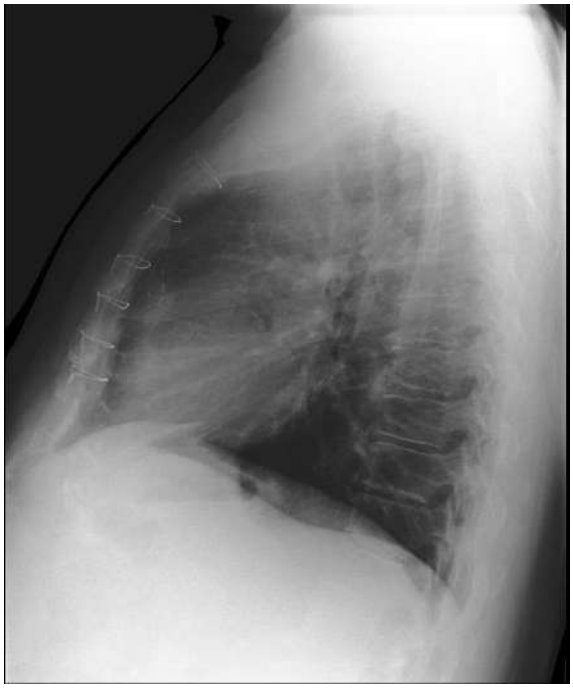

Рисунок 3.3.2 — Приклад рентгенівського знімку зробленого збоку тулуба. Знімок взятий з набору даних MIMIC-CXR

Тестування різних конфігурацій ретривера з використанням класифікатора сторін, класифікатора хвороб та іншого подано в розділі 4.

3.3.1 Створення моделей індексації

В якості моделі для створення індесів була вибрана модель BiomedClip [40]. BiomedCLIP — це модель, створена на основі CLIP [19], для задач біомедичного бачення. BiomedClip був натренований на наборі даних PMC-15M, створений дослідниками BiomedClip. PMC-15M складається із 15 мільйонів пар підпис-картинка з різних медичних галузей: офтальмологія, стоматологія, рентгенографія та інші. Також дослідники BiomedClip створили модель PubMedBERT та свій власний Vision Transformer, які використовуються як кодувальник тексту та як кодувальник зображень відповідно до архітектури CLIP.

Хоча BiomedCLIP натренований на медичних даних, він не навчений безпосередньо на тих даних, які планується використовувати в ретривері для релевантного пошуку. Також варто враховувати, що BiomedCLIP натренований на широкому спектрі медичних галузей, а нам потрібно лише рентгенівські знімки грудей. Враховуючи попередні зауваження, зрозуміло, що просте використання BiomedCLIP, як моделі для створення індексів може часто призводити до нерелевантних результатів [41].

Для покращення роботи BiomedCLIP застосовують передавальне навчання. У межах цієї роботи BiomedCLIP було донавчено на даних MIMIC-CXR, оскільки саме ці дані надалі використовуються ретривером для релевантного пошуку.

Для донавчання був написаний код (див. програмний код 3.3.1.1), використовуючи бібліотеку PyTorch. Код навчання написаний згідно з підходом запронованим в досліджені про модель Clip. Як функція втрат [42] використовується перехрестна втрата ентропії (англ. cross entropy loss) [43] та AdamW [44] як оптимізатор [45].

```
images = images.to(device)
reports = reports.to(device)

optimizer.zero_grad()

image_features = model.encode_image(images)
text_features = model.encode_text(reports)

image_features = image_features / image_features.norm(dim=-1, keepdim=True)
text_features = text_features / text_features.norm(dim=-1, keepdim=True)

logit_scale = model.logit_scale.exp()
logits_per_image = logit_scale * image_features @ text_features.t()
logits_per_text = logits_per_image.t()

batch_size = images.size(0)
labels = torch.arange(batch_size, device=device)

loss_i = criterion(logits_per_image, labels)
loss_t = criterion(logits_per_text, labels)
loss = (loss_i + loss_t) / 2

loss.backward()
optimizer.step()
```

Програмний код 3.3.1.1 — Тренування BiomedClip за один крок

В результаті було донавчено три окремі моделі BiomedClip.

Перша — для індексації передніх рентгенівських знімків і їхніх звітів (далі — модель індексації передніх знімків). Вона була донавчена на 100000 випадково вибраних пар передніх рентгенгенівських знімків та їхніх звітів з MIMIC-CXR. Модель донавчалася протягом 100 епох [46] на відеокарті Nvidia GeForce RTX 3090 24 ГБ. Для донавчання використовувалися такі гіперпараметри [47]: розмір партії (англ. batch size) [48] — 64, темп навчання (англ. learning rate) [49] — $0.00005\sqrt{8}$. Значення втрат на останній епосі склало: 0.0336.

Друга — для індексації бічних рентгенівських знімків і їхніх звітів (далі — модель індексації бічних знімків). Вона була донавчена на 50000 випадково вибраних пар бічних рентгенгенівських знімків та їхніх звітів з MIMIC-CXR. Модель донавчалася протягом 100 епох на відеокарті Nvidia GeForce RTX 2060 6 ГБ. Для донавчання використовувалися такі гіперпараметри: розмір партії — 8, темп навчання — 0.00005. Значення втрат на останній епосі склало: 0.0306.

Третя — для індексації передніх та бічних рентгенівських знімків і їхніх звітів (далі — загальна модель індексації). Вона була донавчена на 150000 випадково вибраних пар передніх та бічних рентгенгенівських знімків та їхніх звітів з MIMIC-CXR. Модель донавчалася протягом 95 епох на відеокарті Nvidia GeForce RTX 3090 24 ГБ. Для донавчання використовувалися такі гіперпараметри: розмір партії — 64, темп навчання — $0.00005\sqrt{8}$. Значення втрат на останній епосі склало: 0.0337.

Для того, щоб в подальшому зручно використовувати попередні моделі був створений клас-обгортка Encoder (див. програмний код 3.3.1.2) з двома головними методами encode_image та encode_text, які створюють індекси зі зображень та тексту відповідно.

```
class Encoder:

    def __init__(self,
                 model_path:str = DEFAULT_ENCODER_MODEL_PATH,
                 checkpoint_path: str = None,
                 device:str = "cpu",
                 **kwargs):
        model, _, preprocess_val = load_encoder_model(model_path=model_path,
                                                      checkpoint_path=checkpoint_path,
                                                            device=device,
                                                            **kwargs)
        self.model = model
        self.preprocessor = preprocess_val
        self.tokenizer = load_encoder_tokenizer(model_path=model_path,
                                                **kwargs)
        self.device = device

    @torch.no_grad()
    def encode_image(self, image: Image.Image) -> List[float]:
        self.model.eval()
        if image.mode != "RGB":
            image = image.convert("RGB")
        image = self.preprocessor(image).unsqueeze(0)
        image = image.to(self.device)
        image_embedding =  self.model.encode_image(image)[0]
        image_embedding = image_embedding.cpu().detach().tolist()
        return image_embedding

    @torch.no_grad()
    def encode_text(self, text: str) -> List[float]:
        self.model.eval()
        text = self.tokenizer([text])
        text = text.to(self.device)
        text_embedding = self.model.encode_text(text)[0]
        text_embedding = text_embedding.cpu().detach().tolist()
        return text_embedding
```

Програмний код 3.3.1.2 — Клас Encoder

3.3.2 Створення класифікатора сторін

Для того, щоб ретривер міг правильно вибирати модель для класифікації передніх чи бічних знімків, він повинен вміти класифікувати зображення на передні і бічніі. Це класична задача класифікації зображень [50]. Для такої задачі найчастіше використовуються нейронні мережі. Тренування класифікатора сторін відбувалося на наборі даних CheXpert, бо він містить інформацію про сторону кожного рентгенівського знімку. В якості нейронної мережі для дотренування була вибрана модель DenseNet121, яка також використовувалася в дослідженні набору даних CheXpert і показала кращі результати ніж інші моделі в дослідженні. DenseNet121 — це згоркова нейронна мережа [51], яка складається з 121 шару кожен пов'язаний один з одним.

Для тренування (див. програмний код 3.3.2.1) використовувалася передавальне навчання. Для цього використовувалася модель DenseNet121, яка була попередньо натренована на наборі даних ImageNet [52]. Як функція втрат використовувалася перехрестна втрата ентропії та стохастичний градієнтний спуск [53] як оптимізатор.

```
inputs = inputs.to(device)
labels = labels.to(device)
optimizer.zero_grad()

outputs = model(inputs)
_, preds = torch.max(outputs, 1)
loss = criterion(outputs, labels)
loss.backward()
optimizer.step()
```

Програмний код 3.3.2.1 — Тренування класифікатора сторін за один крок

Тренування класифікатора сторін відбувалося протягом 10 епох на відеокарті Nvidia GeForce RTX 2060 ГБ. Для тренування використовувалося 20000 випадково вибраних рентенівських знімків з набору даних CheXpert. Тестування моделі після кожної епохи відбувалося на 2000 рентгенівських

знімках. Вже після першої епохи тренування, точність визначення сторони зображення на тренувальних та тестових даних склала 100%, тому тренування було закінчене достроково на 5-ій епосі. Для донавчання використовувалися такі гіперпараметри: розмір партії — 32, темп навчання — 0.001.

Для зручності використання класифікатора сторін в подальшому для навченної моделі був створений клас-обгортка SideClassifier (див. програмний код 3.3.2.2). Цей клас містить метод classify_side, який визначає сторону зображення: Frontal (з англ. переднє), Lateral (з англ. бічне).

```
class SideClassifier:

    labels: Final[List[str]] = ["Frontal", "Lateral"]

    def __init__(self,
                 checkpoint_path: Optional[str] = None,
                 device: str = "cpu", ):
        self.model =
load_classifier_model(labels_count=len(SideClassifier.labels),
                                              checkpoint_path=checkpoint_path,
                                              device=device)
        self.device = next(self.model.parameters()).device
        self.image_transform = transforms.Compose([
            transforms.Resize((224, 224)),
            transforms.ToTensor(),
        ])

    @torch.no_grad()
    def classify_side(self, image: Image.Image) -> str:
        self.model.eval()
        if image.mode != "RGB":
            image = image.convert("RGB")
        image = self.image_transform(image).unsqueeze(0)
        image = image.to(self.device)
        outputs = self.model(image)
        _, indexes = torch.max(outputs, 1)
        return SideClassifier.labels[indexes[0]]
```

Програмний код 3.3.2.2 — Клас SideClassifier

3.3.3 Створення класифікатора хвороб

Для покращення релевантності знайдених результатів ретривером було створено класифікатор хвороб. Класифікатор хвороб визначає, які клінічні ознаки із заданого списку наявні на зображенні, а які ні. Це задача класифікації

за багатьма класами (англ. multi-label classification) [54], задача в якій один об’єкт може відповідати декільком класам одночасно. Ця задача схожа на класифікацію зображень, використану в класифікаторі сторін. Для її розв’язання також найчастіше використовують нейронні мережі.

Для створення класифікатора хвороб було використано модель DenseNet121, як в класифікаторі сторін. Для тренування (див. програмний код 3.3.3.1) було використано набір даних CheXpert, тому що він містить інформацію про наявність певної ознаки з списку 14 клінічних ознак для кожного зображення. Кожна ознака для кожного зображення в наборі даних CheXpert може мати одне з 4-ох значень: «наявна», «відсутня», «не зазначена», «не зрозуміло». В тренуванні значення «не зазначена» і «не зрозуміло» будуть сприйматися як один клас, тому що важлива лише наявність або відсутність ознаки. Вихідним значенням тренувальної моделі DenseNet121 є вектор з 14 чисел, кожне з яких відповідає за наявність певної хвороби. Значення чисел вектора знаходяться в межах від нуля включно до одиниці включно, числа більше 0.7 будуть сприйматися як наявність хвороби, числа менше 0.3 будуть сприйматися як її відсутність. Все, що між 0.3 та 0.7 буде відповідати значенням «не зазначена» або «не відомо». Для тренування, як в класифікаторі сторін, було використане передавальне навчання та модель DenseNet121, яка була попередньо натренована на наборі даних ImageNet. Як функція втрат використовувалася перехрестна втрата ентропії та Adam [55] як оптимізатор.

```
inputs = inputs.to(device)
targets = targets.to(device)
optimizer.zero_grad()
outputs = model(inputs)
losses = criterion(outputs, targets)
mask = ((targets == 0) | (targets == 1)).float()
loss = (losses * mask).sum() / mask.sum()
loss.backward()
optimizer.step()
```

Програмний код 3.3.3.1 — Тренування класифікатора хвороб за один крок

Тренування класифікатора хвороб відбувалося протягом 50 епох на відеокарті Nvidia GeForce RTX 2060 6 ГБ. Для тренування використовувалося 156553 випадково вибраних рентенівських знімків з набору даних CheXpert. Тестування моделі після кожної епохи відбувалося на 67095 рентгенівських знімках. Після останньої епохи точність визначення наявості та відсутності хвороб на тестових даних склала 86,71%, що майже відповідає результатам у досліджені CheXpert. Для донавчання використовувалися такі гіперпараметри: розмір партії — 32, темп навчання — 0.0001.

Для зручності використання класифікатора хвороб в подальшому для навченної моделі був створений клас-обгортка DiseaseClassifier (див. програмний код 3.3.3.2). Цей клас містить метод classify, який повертає на вхідне зображення python словник (англ. dictionary), ключами якого є назви хвороб, а значеннями число, яке позначає наявність хвороби: 1 — якщо хвороба наявна, 0 — якщо хвороба відсутня, -1 — якщо невідомо.

```
class DiseaseClassifier:

    labels: Final[List[str]] = ["No Finding",
                                "Enlarged Cardiomediastinum",
                                "Cardiomegaly",
                                "Lung Opacity",
                                "Lung Lesion",
                                "Edema",
                                "Consolidation",
                                "Pneumonia",
                                "Atelectasis",
                                "Pneumothorax",
                                "Pleural Effusion",
                                "Pleural Other",
                                "Fracture",
                                "Support Devices"]

    def __init__(self,
                 checkpoint_path: Optional[str] = None,
                 device: str = "cpu",):
        self.model =
load_classifier_model(labels_count=len(DiseaseClassifier.labels),
                                            checkpoint_path=checkpoint_path,
                                            device=device)
        self.device = next(self.model.parameters()).device
        self.image_transform = transforms.Compose([
            transforms.Resize((224, 224)),
            transforms.ToTensor(),
        ])

    @torch.no_grad()
    def classify(self, image: Image.Image) -> Dict[str, int]:
        self.model.eval()
        if image.mode != "RGB":
            image = image.convert("RGB")
        image = self.image_transform(image).unsqueeze(0)
        image = image.to(self.device)
        output = self.model(image)[0]
        normalized_output = dual_threshold_outputs_normalize(output)
        normalized_output = normalized_output.cpu().detach().tolist()
        classification = {label: int(normalized_output[idx]) for idx, label in
enumerate(self.labels)}
        return classification
```

Програмний код 3.3.3.2 — Клас DiseaseClassifier

3.3.4 Векторна база даних

Для того, щоб зберігати індекси, звіти, метадані та виконувати пошук за індексами, в якості векторної бази даних була використана ChromaDb, яка повністю задовільняє вимогам для створення системи відповідно до 2-го розділу. В ChromaDb дані зберігаються в колекціях, аналог до таблиць в реляційних базах

даних. Кожен запис в колекції повинен мати індекс, за яким буде відбуватися пошук, та якісь дані. Також запис може містити додаткові дані — метадані, за якими може відбуватися додатковий пошук. Індекси всіх записів в колекції повинні бути однієї розмірності.

Для користування ChromaDb в роботі був створений клас EmbeddingCollection. Цей клас створює, використовуючи попередньо створені моделі індексації та класифікатор хвороб, певну колекцію, якщо вона не була створена раніше, та виконує пошук за допомогою методу get_closest (див. програмний код 3.3.4.1). Метод get_closest знаходить вказану кількість релевантних звітів з колекції до вхідного зображення. Також цей метод підтримує змогу вказати чи потрібно використовувати класифікатор хвороб і метадані в пошуці. Якщо метадані використовуються в пошуці, можна вказати чи шукати звіти, що мають точне співпадіння хвороб як на вхідному зображенні чи щоб було хоча б одне співпадіння (далі — не точне співпадіння). Тестування ефективності різних варіантів співпадіння проведене в 4-му розділі.

```
def get_closest(self,
                image: Image.Image,
                count: Optional[int] = None,
                use_disease_classifier: Optional[bool] = None,
                exact_disease_match: Optional[bool] = None) -> Tuple[List[str],
List[float]]:
    if count is None:
        count = self.count
    if use_disease_classifier is None:
        use_disease_classifier = self.use_disease_classifier
    if exact_disease_match is None:
        exact_disease_match = self.exact_disease_match
    image_embedding = self.encoder.encode_image(image)
    metadata_query = None
    include = ["documents", "distances"]
    if use_disease_classifier and self.disease_classifier is not None:
        metadata_query = self.disease_classifier.classify(image)
        metadata_query = [{k: v} for k, v in metadata_query.items() if v != -1]
        if exact_disease_match:
            metadata_query = {"$and": metadata_query}
        else:
            metadata_query = {"$or": metadata_query}
    result = self.collection.query(
        query_embeddings=[image_embedding],
        n_results=count,
        where=metadata_query,
        include=include
    )
    return result["documents"][0], result["distances"][0]
```

Програмний код 3.3.4.1 — Метод get_closest класу EmbeddingCollection

У межах реалізації системи було створено три окремі колекції в ChromaDb, кожна з яких відповідала певному типу рентгенівських знімків і використовувала відповідну модель індексації. Для створення колекцій використовувся набір даних MIMIC-CXR. Перша колекція була створена за допомогою моделі індексації передніх знімків на 100000 випадково вибраних пар передніх ретгенівських знімків та їхніх звітів. Друга була створена за допомогою моделі індексації бічних знімків на 50000 випадково вибраних пар бічних ретгенівських знімків та їхніх звітів. Третя була створена за допомогою загальної моделі індексації на 150000 випадково вибраних пар передніх і бічних ретгенівських знімків та їхніх звітів.

3.3.5 Клас Retriever

Для зручного користування ретривером був створений клас Retriever, який використовує всі попередньо створені класи. Об’єкт класу Retriever може бути створений з використанням різних колекцій, моделей індексації, з класифікатором хвороб та сторін або без них. Це зроблено для можливості тестувати або використовувати різні комбінації ретриверу. Найголовніший метод для знаходження релевантних даних класу Retriever — це retrieve (див. програмний код 3.3.5.1). Цей метод відповідає за знаходження релевантних звітів з колекцій за вхідним зображенням. Кількість релевантих звітів для пошуку, можна вказати за допомогою параметра count.

```
def retrieve(self,
             image: Image.Image,
             count: Optional[int] = None,
             **kwargs) -> List[str]:
    if count is None:
        count = self.count
    if self.side_classifier is not None:
        image_side = self.side_classifier.classify_side(image=image)
        if image_side == "Frontal" and self.frontal_embedding_collection is not
None:
            retrieved_reports, _ = self.__retrieve_frontal(image=image,
count=count, **kwargs)
        elif image_side == "Lateral" and self.lateral_embedding_collection is
not None:
            retrieved_reports, _ = self.__retrieve_lateral(image=image,
count=count, **kwargs)
        else:
            retrieved_reports, _ = self.__retrieve_full(image=image,
count=count, **kwargs)
    else:
        retrieved_reports, _ = self.__retrieve_full(image=image, count=count,
**kwargs)
    return retrieved_reports
```

Програмний код 3.3.5.1 — Метод retrieve класу Retriever

3.4 Генератор

В якості генератора була вибрана велика мовна модель LLaVA-Med 1.5 [56]. Ця модель є дотренованою версією LLaVA 1.5 на медичних даних, розробленою дослідниками з Microsoft [56]. У роботі використовувалася LLaVA-Med, яка містить 7 мільярдів параметрів. Так як для параметрів в нейронних

мережах використовується 32-бітні числа з рухомою комою, то для запуску LLaVA-Med 1.5 на відеокарті треба більше 26 ГБ відеопам’яті. При виконанні роботи не було доступу до відеокарт з такою кількістю відеопам’яті. Для того, щоб запускати нейронні мережі на відеокартах з меншою кількісю відеопам’яті використовують техніку квантинізації [37]. Квантинізація дозволяє зменшити кількість пам’яті, яку займає модель, за допомогою представлення параметрів моделі 16-ти, 8-ми або 4-ма бітними числами з рухомою комою. Звісно використання квантинізації може призвести до деградації моделі, але за дослідженням це або не відбувається взагалі або воно не суттєве. Для запуску LLaVA-Med 1.5 на Nvidia GeForce RTX 2060 6 ГБ була використана техніка квантинізації у режимі 4-біт (див. програмний код 3.4.1). В результаті модель займає приблизно 5 ГБ відеопам’яті.

```
kwargs = {}
kwargs["device_map"] = "auto"
kwargs["low_cpu_mem_usage"] = True
kwargs["load_in_4bit"] = True
kwargs["quantization_config"] = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4"
)
self.model = LlavaMistralForCausalLM.from_pretrained(model_path, **kwargs)
self.tokenizer = AutoTokenizer.from_pretrained(model_path, use_fast=False)
vision_tower = self.model.get_vision_tower()
if not vision_tower.is_loaded:
    vision_tower.load_model()
vision_tower.to(device=self.model.device, dtype=torch.float16)
self.model.model.mm_projector.to(device=self.model.device, dtype=torch.float16)
self.image_processor = vision_tower.image_processor
```

Програмний код 3.4.1 — Код запуску LLaVA-Med 1.5 7b в режимі 4 біт

Для зручності користування LLaVA-Med 1.5 для неї був створений клас-обгортка LLaVAMedGenerator, який містить методи для генерації звіту і відповіді на питання. Метод generate_report генерує звіт на вхідне зображення. Якщо метод не отримує на вхід ніяких знайдених релевантних звітів з бази даних, він буде використовувати до генератора запит (див. програмний код 3.4.2), який

містить тільки вхідне зображення й текст запиту. Якщо метод отримує релевантні звіти з бази даних для вхідного зображення, він використовує інший запит (див. програмний код 3.4.3), який містить не тільки вхідне зображення й текст запиту, а й релевантні звіти. Метод answer_binary_question генерує відповідь на вхідне питання й зображення до нього. Метод answer_binary_question працює схожим чином до generate_report і використовує два різні запити залежно від наявності релевантних звітів (див. програмний код 3.4.4 і програмний код 3.4.5).

```
prompt = ("USER:\n"
          + "<image>\n"
          + "You are a professional radiologist.\n"
          + "You are provided with an X-ray image of chest.\n"
          + "Please generate a detailed radiology report based on the image.\n"
          + "Please only include the content of the report in your response.\n"
          + "ASSISTANT:")
```

Програмний код 3.4.2 — Запит до LLaVA-Med 1.5 для генерації звіту на вхідне зображення без релевантих звітів

```
prompt = ("USER:\n"
          + "<image>\n"
          + "You are a professional radiologist.\n"
          + f"You are provided with an X-ray image of chest and
{len(retrieved_reports)} reference report(s):\n"
          + f"{formatted_reports}\n"
          + "Please generate a report based on the image.\n"
          + "Review the image independently from reports.\n"
          + "Do NOT copy diagnostic conclusions from the reference reports.\n"
          + "It should be noted that the diagnostic information in the reference
reports cannot be directly used as the basis for diagnosis, but should only be
used for reference and comparison.\n"
          + "Please only include the content of the report in your response.\n"
          + "ASSISTANT:")
```

Програмний код 3.4.3 — Запит до LLaVA-Med 1.5 для генерації звіту на вхідне зображення з релевантими звітами

```
prompt = ("USER:\n"
          + "<image>\n"
          + "You are a professional radiologist.\n"
          + "You are provided with an X-ray image of chest and image-related
question.\n"
          + "Please answer the question based on the image.\n"
          + "Answer can be only \"yes\" or \"no\".\n"
          + f"Question: {question}\n"
          + "ASSISTANT:")
```

Програмний код 3.4.4 — Запит до LLaVA-Med 1.5 для генерації відповіді за вхідним питанням і зображення без релевантих звітів

```
prompt = ("USER:\n"
          + "<image>\n"
          + "You are a professional radiologist.\n"
          + f"You are provided with an X-ray image of chest, image-related
question and {len(retrieved_reports)} reference report(s):\n"
          + f"{formatted_reports}\n"
          + "Please answer the question based on the image.\n"
          + "Answer can be only \"yes\" or \"no\".\n"
          + "Review the image independently from reports.\n"
          + "It should be noted that the diagnostic information in the reference
reports cannot be directly used as the basis for diagnosis, but should only be
used for reference and comparison.\n"
          + f"Question: {question}\n"
          + "ASSISTANT:")
```

Програмний код 3.4.5 — Запит до LLaVA-Med 1.5 для генерації відповіді за вхідним питанням і зображення з релевантими звітами

3.5 Використання створеного мультимодального RAG

Для використання ретривера і генератора разом був створений клас MultimodalRAG (див. програмний код 3.5.1). Цей клас має два методи generate_report та answer_binary_question, які повторюють функціональність методів з такими самими назвами класу LLaVAMedGenerator. Але на відміну від методів класу LLaVAMedGenerator, методи класу MultimodalRAG використовують ретривер, якщо він був вказаний під час створення об’єкту класу MultimodalRAG.

```
class MultimodalRAG:

    def __init__(self,
                 generator: Generator,
                 retriever: Optional[Retriever] = None):
        self.retriever = retriever
        self.generator = generator

    def generate_report(self, image: Image.Image) -> str:
        if self.retriever is None:
            return self.generator.generate_report(image=image)
        retrieved_reports = self.retriever.retrieve(image=image)
        return self.generator.generate_report(image=image,

retrieved_reports=retrieved_reports)

    def answer_binary_question(self, image: Image.Image, question: str) -> str:
        if self.retriever is None:
            return self.generator.answer_binary_question(image=image,
question=question)
        retrieved_reports = self.retriever.retrieve(image=image)
        return self.generator.answer_binary_question(image=image,
                                                     question=question,

retrieved_reports=retrieved_reports)
```

Програмний код 3.5.1 — Клас MultimodalRAG

3.6 Приклади роботи створеної мультимодальної RAG-системи

Для створення прикладів роботи створеної мультимодальної RAG-системи був ініціалізований об’єкт класу MultimodalRAG з ретривером і генератором. Ретривер використовує класифікатор сторін, модель індексації передніх знімків, модель індексації бічних знімків і класифікатор хвороб (див. рисунок 3.6.1). Також ретривер використовує не точне співпадіння хвороб.

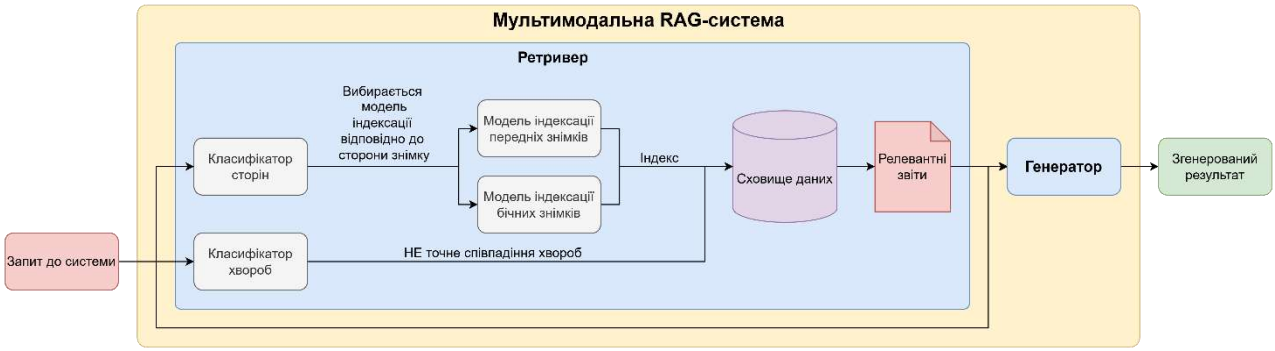

Рисунок 3.6.1 — Конфігурація мультимодальної RAG-системи для створення прикладів роботи

Для генерування звіту на вхідне зображення був використаний метод generate_report класу MultimodalRAG. Як вхідне зображення було використано випадковий рентгенівський знімок з набору даних MIMIC-CXR (див. рисунок 3.6.2). В результаті мультимодальний RAG згенерував такий звіт: «The chest X-ray image shows a large right pleural effusion, which is an abnormal accumulation of fluid in the pleural space surrounding the lungs. This finding is concerning for pneumonia, which is an infection that causes inflammation in the air sacs of the lungs. It is important to consult a healthcare professional for a thorough evaluation and proper diagnosis of the underlying cause of these findings.». Еталонний звіт із набору даних для цього ж самого зображення: «impression: Increased right pleural loculated effusion with chest tube in place. Increasing consolidation in the right lung is concerning for pneumonia. Findings: PA and lateral views of the chest provided. Port-A-Cath is unchanged in position with its tip positioned in the expected location of the mid SVC. A right pleural drain is in place with increased opacity in the right lung and probable increase in size of the loculated right pleural effusion. Findings are concerning for a superimposed consolidation/pneumonia. The left lung remains essentially clear. The heart is difficult to assess given the effacement of the right heart border. The prominence of the mediastinum may reflect in part adjacent loculated pleural fluid. No pneumothorax is seen.».

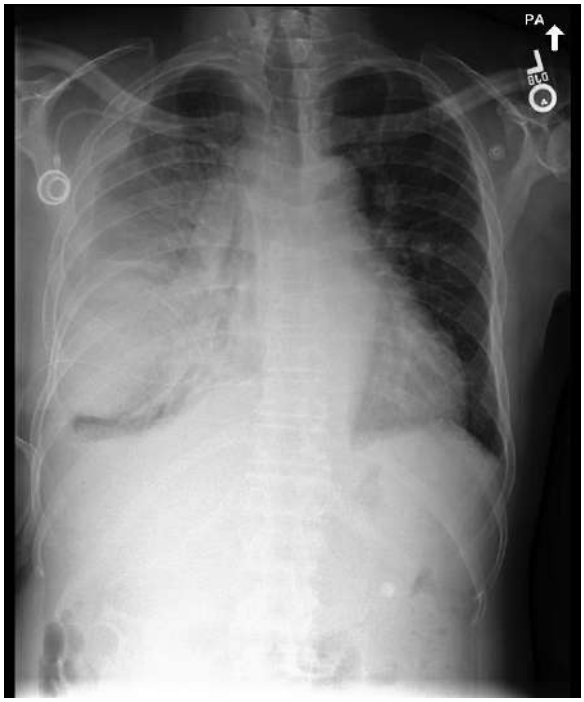

Рисунок 3.6.2 — Рентгенівський знімок використаний для генерації звіту

Для генерування відповіді на вхідне питання й зображення до нього був використаний метод answer_binary_question класу MultimodalRAG. Як вхідне зображення було використано випадковий рентгенівський знімок з набору даних MIMIC-CXR (див. рисунок 3.6.3). Вхідним питанням до системи було: «Is the cardiomediastinal silhouette within normal limits?». В результаті система згенерувала таку відповідь: «Yes, the cardiomediastinal silhouette appears to be within normal limits in the image.». Еталонна відповідь на це питання до цього ж самого зображення з набору даних: «Yes».

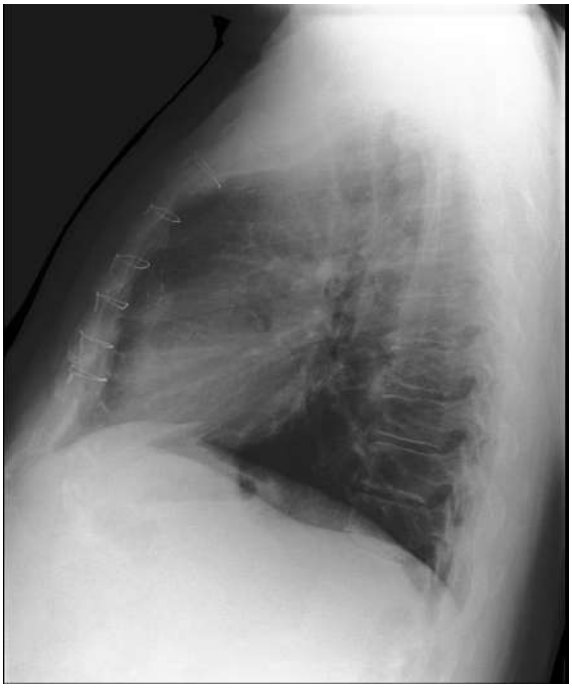

Рисунок 3.6.3 — Рентгенівський знімок використаний для генерації відповіді на питання

РОЗДІЛ 4: ТЕСТУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ

4.1 Конфігурації мультимодальних RAG-систем, які тестувалися

Тестування створеної системи проводилося на шістьох конфігураціях мультимодального RAG, кожна з яких була налаштована на знаходження 5-ти релевантних звітів до кожного запиту. Як зразок порівняння конфігурацій також була протестована LLaVA-Med 1.5 без використання ретриверу.

Ретривер першої конфігурації мультимодальної RAG-системи (далі — Конфігурація 1, див. рисунок 4.1.1) включає класифікатор сторін, модель індексації передніх знімків, модель індексації бічних знімків і класифікатор хвороб. Ретривер використовує не точне співпадіння хвороб.

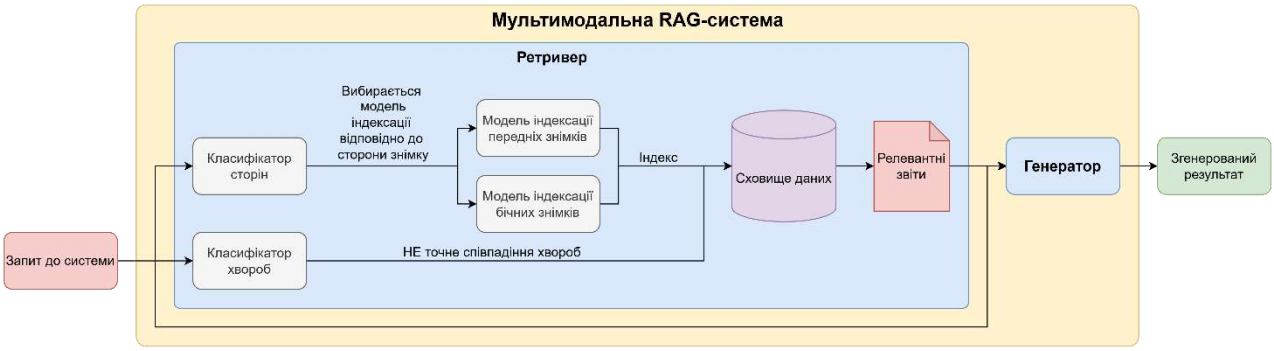

Рисунок 4.1.1 — Конфігурація 1 (з класифікатором сторін, з класифікатором хвороб, НЕ точне співпадіння хвороб)

Ретривер другої кофігурації мультимодальної RAG-системи (далі — Конфігурація 2, див. рисунок 4.1.2) майже такий самий як у конфігурації 1, але використовує точне співпадіння хвороб.

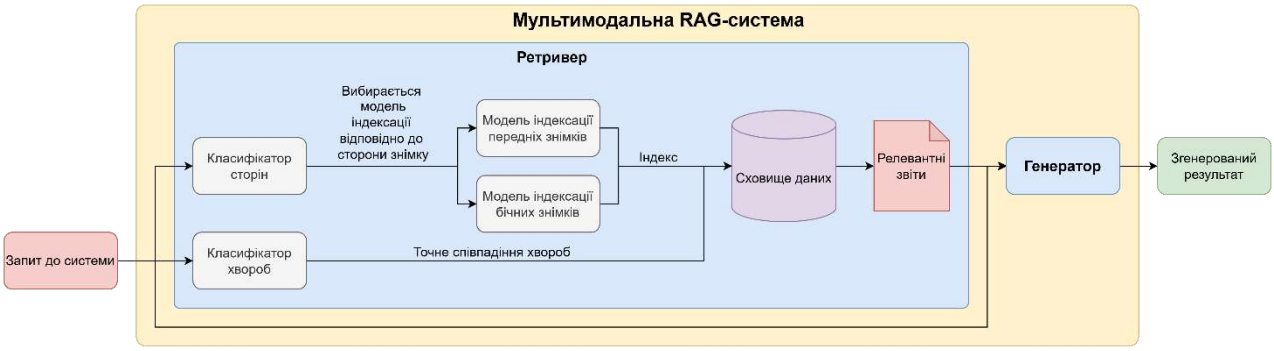

Рисунок 4.1.2 — Конфігурація 2 (з класифікатором сторін, з класифікатором хвороб, точне співпадіння хвороб)

Ретривер третьої кофігурації мультимодальної RAG-системи (далі — Конфігурація 3, див. рисунок 4.1.3) схожий на ретривер конфігурації 1 і конфігурації 2, але не використовує класифікатор хвороб.

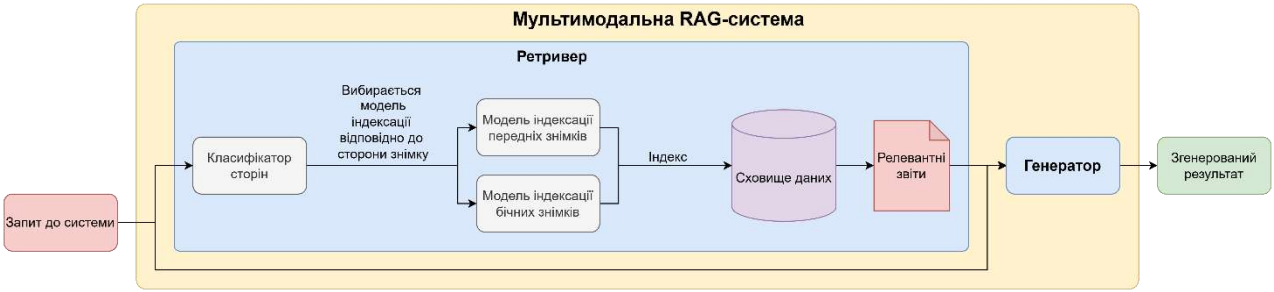

Рисунок 4.1.3 — Конфігурація 3 (з класифікатором сторін, БЕЗ класифікатора хвороб)

Ретривер четвертої конфігурації мультимодальної RAG-системи (далі — Конфігурація 4, див. рисунок 4.1.4) включає загальну модель індексації та класифікатор хвороб. Ретривер використовує не точне співпадіння хвороб.

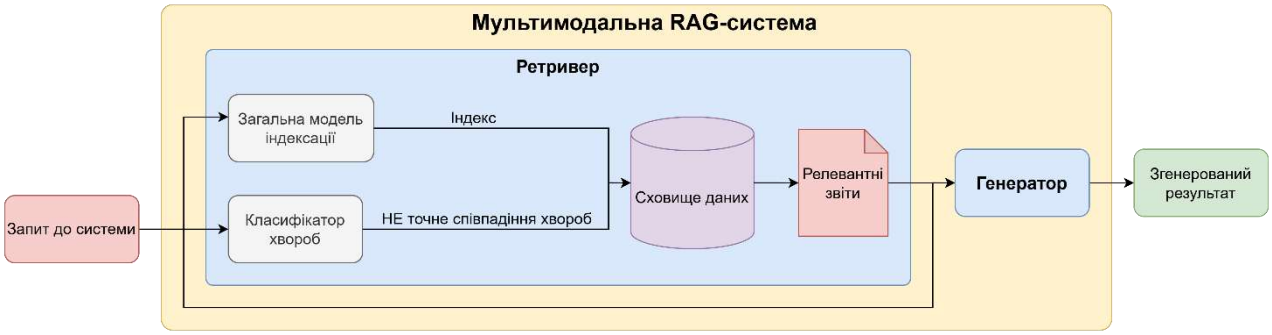

Рисунок 4.1.4 — Конфігурація 4 (БЕЗ класифікатора сторін, з класифікатором хвороб, НЕ точне співпадіння хвороб)

Ретривер п’ятої кофігурації мультимодальної RAG-системи (далі — Конфігурація 5, див. рисунок 4.1.5) майже такий самий як у конфігурації 4, але використовує точне співпадіння хвороб.

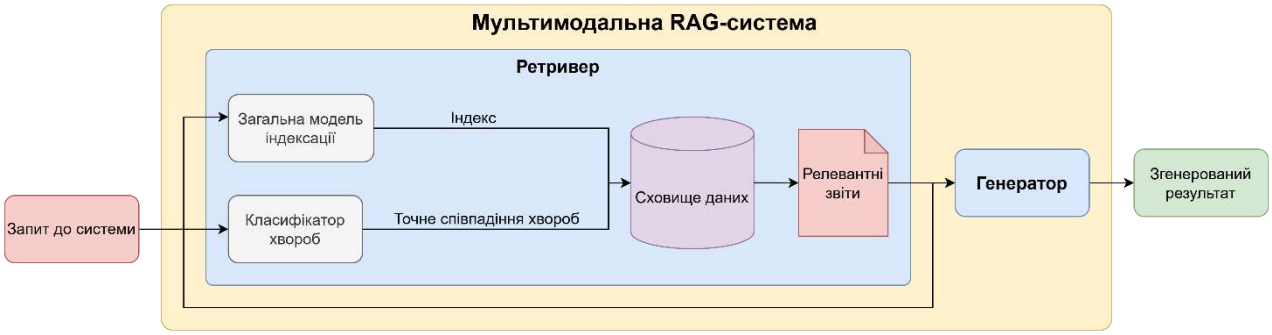

Рисунок 4.1.5 — Конфігурація 5 (БЕЗ класифікатора сторін, з класифікатором хвороб, точне співпадіння хвороб)

Ретривер шостої кофігурації мультимодальної RAG-системи (далі — Конфігурація 6, див. рисунок 4.1.6) схожий на ретривер конфігурації 4 і конфігурації 5, але не використовує класифікатор хвороб.

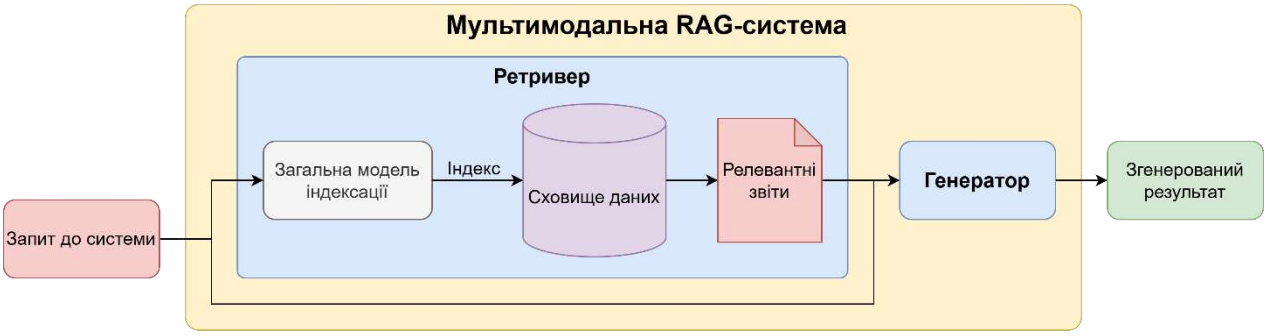

Рисунок 4.1.6 — Конфігурація 6 (БЕЗ класифікатора сторін, БЕЗ класифікатора хвороб)

4.2 Тестування системи відповідати на питання за зображенням

Для тестування мультимодальної RAG-системи відповідати на вхідне питання і зображення до нього були використані дві вибірки тестових даних створені дослідниками мультимодального RAG [41]. Кожна вибірка даних містить приблизно 2500 питань, відповідей і зображень. Одна вибірка містить зображення з набору даних MIMIC-CXR, а інша з набору даних IU-Xray. Дослідники, які створили вибірки даних, використовували велику мовну модель ChatGPT-4 для створення питань і відповідей до рентгенівських знімків. Потім дослідники власноруч відфільтрували і перевірили кожне питання і відповідь. Всі питання створені таким чином, щоб вони мали тільки два варіанта відповіді: «так» або «ні».

Для тестування системи відповідати на вхідне питання і зображення до нього було використано дві метрики: метрика точності (англ. accuracy) [57] і метрика F1 (англ. F1-score) [58].

Метрика точності — одна з найпростіших і найпоширеніших оцінок якості класифікаційних моделей. Вона показує, яка частка всіх передбачень моделі виявилася правильною. В нашому випадку вона покаже яка частка відповідей була правильно визначена. Точність набуває значень від нуля включно до одиниці включно, де нуль означає, що ніяка відповідь не була визначена правильно, а одиниця означає, що всі відповіді були визначені правильно.

Для обчислення метрики F1 треба поділити всі відповіді на позитивні і негативні. Позитивні відповіді — це відповіді, які ми оцінюємо, а негативні — всі інші. F1 показує, наскільки добре модель одночасно уникає хибних спрацьовувань (помилкових позитивних відповідей) та пропусків (хибних негативних відповідей). Вона обчислюється як гармонійне середнє [59] між влучністю [60] (англ. precision, доля правильних позитивних передбачень серед усіх позитивних передбачень) і повнотою [60] (англ. recall, доля правильних позитивних передбачень серед усіх позитивних відповідей), тому низьке значення однієї з цих метрик суттєво знижує F1. F1, як і точність, набуває значень від нуля включно до одиниці включно. Нуль означає, що модель не змогла правильно передбачити жодного позитивного випадку, а одиниця означає ідеальну відповідність. Для обчислення метрики F1 на вибірках даних буде використовуватися зважена F1 (англ. F1-weighted). Зважена F1 обчислюється окремо для кожного типу відповіді, а потім об’єднується, використовуючи кількість кожного типу відповіді.

Тестування відбувалося на 1000 випадково вибраних питань з кожної вибірки даних. Для представлення результатів тестування створена таблиця (див. таблиця 4.2.1). Вибірка даних, яка містить зображення з набору даних MIMIC-CXR, підписана MIMIC-CXR, а інша вибірка — IU-Xray, бо містить зображення з набору даних IU-Xray. Спочатку в таблиці представлена оцінка LLaVA-Med 1.5 без використання ретриверу, а потім усі комбінації мультимодального RAG. Зеленим кольором виділено клітинки, які містять найкращий результат в стопці.

Таблиця 4.2.1 — Результати тестування системи відповідати на питання за зображенням

	MIMIC-CXR		IU-Xray	
	Точність	Зважена F1	Точність	Зважена F1
LLaVA-Med 1.5	0.717	0.668	0.411	0.418
Конфігурація 1 (з класифікатором сторін, з класифікатором хвороб, НЕ точне співпадіння хвороб)	0.840	0.838	0.921	0.924
Конфігурація 2 (з класифікатором сторін, з класифікатором хвороб, точне співпадіння хвороб)	0.844	0.842	0.909	0.913
Конфігурація 3 (з класифікатором сторін, БЕЗ класифікатора хвороб)	0.831	0.830	0.929	0.931
Конфігурація 4 (БЕЗ класифікатора сторін, з класифікатором хвороб, НЕ точне співпадіння хвороб)	0.766	0.766	0.917	0.919
Конфігурація 5 (БЕЗ класифікатора сторін, з класифікатором хвороб, точне співпадіння хвороб)	0.786	0.784	0.918	0.921
Конфігурація 6 (БЕЗ класифікатора сторін, БЕЗ класифікатора хвороб)	0.773	0.773	0.921	0.923

4.3 Тестування системи генерувати звіт за зображенням

Для тестування мультимодального RAG генерувати звіт до вхідного зображення були використані дві вибірки тестових даних, створені дослідниками мультимодального RAG [41]. Перша вибірка містить 700 зображень та звітів з набору даних MIMIC-CXR, а друга — 1180 зображень та звітів з набору даних IU-Xray. Кожна вибірка сформована таким чином, щоб вона містила якнайрізноманітніші звіти. Тестування системи за різними метриками проводилось на 700 зображень та звітів з набору набору даних MIMIC-CXR (з першої вибірки) та 1000 зображень та звітів з набору даних IU-Xray (з другої вибірки).

Для тестування системи генерувати звіт до вхідного зображення всього було використано чотири метрики: BLEU [61], ROUGE [62], F1-CheXbert [63], F1-RadGraph [64].

BLEU (Bilingual Evaluation Understudy) — це метрика для порівняння згенерованого чи перекладеного тексту з одним або кількома еталонними варіантами. Для оцінки за метрикою BLEU підраховують кількість однакових послідовностей із n слів (n-грам), що зустрічаються як у згенерованому, так і в еталонному тексті. Потім обчислюється влучність як відношення числа збігів до загальної кількості n-грам у згенерованому тексті. Зазвичай влучність обраховують для n-грам від одного до чотирьох слів. Далі ці влучності об'єднують за допомогою геометричного середнього. Оскільки короткий (порівняно з еталонним) згенерований текст часто має вищу частку збігів, під час розрахунку BLEU застосовують штраф за короткість (англ. brevity penalty). BLEU набуває значень від нуля включно до одиниці включно. Чим більше значення BLEU, тим більш схожий згенерований текст на еталонний. Варто зазначити, що ця метрика оцінює лише збіги слів, а не зміст чи стиль тексту.

Результати тестування системи генерувати звіт на вхідне зображення за допомогою метрики BLEU представлено у таблиці (див. таблиця 4.3.1). Зеленим кольором виділено клітинки, які містять найкращий результат в стопці.

Таблиця 4.3.1 — Результати тестування системи генерувати звіт на вхідне зображення за допомогою метрики BLEU

	MIMIC-CXR	IU-Xray
LLaVA-Med 1.5	0.006204	0.014087
Конфігурація 1 (з класифікатором сторін, з класифікатором хвороб, НЕ точне співпадіння хвороб)	0.025932	0.035426
Конфігурація 2 (з класифікатором сторін, з класифікатором хвороб, точне співпадіння хвороб)	0.042499	0.036582
Конфігурація 3 (з класифікатором сторін, БЕЗ класифікатора хвороб)	0.027740	0.034322
Конфігурація 4 (БЕЗ класифікатора сторін, з класифікатором хвороб, НЕ точне співпадіння хвороб)	0.018719	0.035311
Конфігурація 5 (БЕЗ класифікатора сторін, з класифікатором хвороб, точне співпадіння хвороб)	0.022738	0.037315
Конфігурація 6 (БЕЗ класифікатора сторін, БЕЗ класифікатора хвороб)	0.018848	0.035051

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) — це набір метрик для порівняння згенерованого чи перекладеного тексту з одним або кількома еталонними варіантами. Набір ROUGE складається з трьох метрик: ROUGE-1, ROUGE-2 і ROUGE-L. Для оцінки за ROUGE-1 підраховують збіги окремих слів між згенерованим і еталонним текстом. Потім обчислюється значення метрики F1 на основі кількості таких збігів. ROUGE-2 обчислюється

аналогічно до ROUGE-1, але замість окремих слів рахують двослівні послідовності. Ця метрика дозволяє оцінити, наскільки модель правильно відтворює не лише окремі слова, а й стійкі словосполучення, порядок слів і короткі фрази. Для оцінки за ROUGE-L обраховують довжину найдовшої спільної підпослідовності між згенерованим і еталонним текстом. Ця метрика відображає не лише точні збіги слів, а й загальну послідовність, у якій вони з’являються. Усі метрики ROUGE набувають значень від нуля включно до одиниці включно. Чим більше значення метрики, тим більш схожий згенерований текст на еталонний. Як і BLEU, набір ROUGE не оцінює зміст чи стиль тексту, а лише лексичну подібність.

Результати тестування системи генерувати звіт на вхідне зображення за допомогою метрик ROUGE-1 (див. таблиця 4.3.2), ROUGE-2 (див. таблиця 4.3.3), ROUGE-L (див. таблиця 4.3.4) представлено у відповідних таблицях. Зеленим кольором виділено клітинки, які містять найкращий результат в стопці.

Таблиця 4.3.2 — Результати тестування системи генерувати звіт на вхідне зображення за допомогою метрики ROUGE-1

	MIMIC-CXR	IU-Xray
LLaVA-Med 1.5	0.177572	0.178954
Конфігурація 1 (з класифікатором сторін, з класифікатором хвороб, НЕ точне співпадіння хвороб)	0.255128	0.258481
Конфігурація 2 (з класифікатором сторін, з класифікатором хвороб, точне співпадіння хвороб)	0.269676	0.257788
Конфігурація 3 (з класифікатором сторін, БЕЗ класифікатора хвороб)	0.256807	0.258677
Конфігурація 4 (БЕЗ класифікатора сторін, з класифікатором хвороб, НЕ точне співпадіння хвороб)	0.233677	0.267725
Конфігурація 5 (БЕЗ класифікатора сторін, з класифікатором хвороб, точне співпадіння хвороб)	0.244741	0.275352
Конфігурація 6 (БЕЗ класифікатора сторін, БЕЗ класифікатора хвороб)	0.234838	0.262273

Таблиця 4.3.3 — Результати тестування системи генерувати звіт на вхідне зображення за допомогою метрики ROUGE-2

	MIMIC-CXR	IU-Xray
LLaVA-Med 1.5	0.017352	0.024311
Конфігурація 1 (з класифікатором сторін, з класифікатором хвороб, НЕ точне співпадіння хвороб)	0.063660	0.054633
Конфігурація 2 (з класифікатором сторін, з класифікатором хвороб, точне співпадіння хвороб)	0.079149	0.054402
Конфігурація 3 (з класифікатором сторін, БЕЗ класифікатора хвороб)	0.067752	0.053155
Конфігурація 4 (БЕЗ класифікатора сторін, з класифікатором хвороб, НЕ точне співпадіння хвороб)	0.052150	0.059491
Конфігурація 5 (БЕЗ класифікатора сторін, з класифікатором хвороб, точне співпадіння хвороб)	0.055802	0.058866
Конфігурація 6 (БЕЗ класифікатора сторін, БЕЗ класифікатора хвороб)	0.053292	0.055589

Таблиця 4.3.4 — Результати тестування системи генерувати звіт на вхідне зображення за допомогою метрики ROUGE-L

	MIMIC-CXR	IU-Xray
LLaVA-Med 1.5	0.111342	0.122759
Конфігурація 1 (з класифікатором сторін, з класифікатором хвороб, НЕ точне співпадіння хвороб)	0.164584	0.175839
Конфігурація 2 (з класифікатором сторін, з класифікатором хвороб, точне співпадіння хвороб)	0.178946	0.176659
Конфігурація 3 (з класифікатором сторін, БЕЗ класифікатора хвороб)	0.165909	0.176821
Конфігурація 4 (БЕЗ класифікатора сторін, з класифікатором хвороб, НЕ точне співпадіння хвороб)	0.147039	0.174326
Конфігурація 5 (БЕЗ класифікатора сторін, з класифікатором хвороб, точне співпадіння хвороб)	0.157708	0.182624
Конфігурація 6 (БЕЗ класифікатора сторін, БЕЗ класифікатора хвороб)	0.148318	0.171794

F1-CheXbert — це метрика, яка оцінює наскільки точно модель відтворює набір клінічних ознак з набору даних CheXpert [5] у радіологічних звітах порівняно з еталонними звітами. F1-CheXbert використовує спеціальну модель CheXbert, яка визначає наявність кожної клінічної ознаки в конкретному звіті. Ця модель натренована на наборі даних CheXpert. Після цього CheXbert застосовують для виявлення клінічних ознак як у згенерованих, так і в еталонних звітах. Для кожної ознаки обчислюється значення метрики F1. Значення F1-CheXbert — це середнє арифметичне значення F1 для всіх ознак, щоб рідкісні, але важливі ознаки мали такий самий вплив на фінальний результат, як і найпоширеніші. F1-CheXbert набуває значень від нуля включно до одиниці

включно. Чим більше значення F1-CheXbert, тим більше клінічних ознак із еталонного звіту модель правильно відтворила в згенерованому тексті.

Результати тестування системи генерувати звіт на вхідне зображення за допомогою метрики F1-CheXbert представлено у таблиці (див. таблиця 4.3.5). Зеленим кольором виділено клітинки, які містять найкращий результат в стопці.

Таблиця 4.3.5 — Результати тестування системи генерувати звіт на вхідне зображення за допомогою метрики F1-CheXbert

	MIMIC-CXR	IU-Xray
LLaVA-Med 1.5	0.011978	0.120694
Конфігурація 1 (з класифікатором сторін, з класифікатором хвороб, НЕ точне співпадіння хвороб)	0.266570	0.217418
Конфігурація 2 (з класифікатором сторін, з класифікатором хвороб, точне співпадіння хвороб)	0.289483	0.219341
Конфігурація 3 (з класифікатором сторін, БЕЗ класифікатора хвороб)	0.263669	0.226831
Конфігурація 4 (БЕЗ класифікатора сторін, з класифікатором хвороб, НЕ точне співпадіння хвороб)	0.221617	0.182259
Конфігурація 5 (БЕЗ класифікатора сторін, з класифікатором хвороб, точне співпадіння хвороб)	0.235491	0.180509
Конфігурація 6 (БЕЗ класифікатора сторін, БЕЗ класифікатора хвороб)	0.214062	0.179557

F1-RadGraph — це метрика, яка оцінює, наскільки добре згенерований радіологічний звіт відтворює структуровану інформацію про клінічні ознаки порівняно з еталонним. F1-RadGraph використовує спеціальну модель RadGraph [65], яка виявляє у звіті клінічні ознаки та зв’язки між ними, формуючи граф.

RadGraph застосовують для виявлення клінічних ознак та зв’язків між ними як у згенерованих, так і в еталонних звітах. Значення F1-RadGraph — це значення F1, а саме F1-мікро (англ. F1-micro), що обчислюється для всіх сутностей і зв’язків разом. F1-RadGraph набуває значень від нуля включно до одиниці включно. Чим більше значення F1-RadGraph, тим більше клінічних ознак і зв’язків між ними з еталонного звіту модель правильно відтворила в згенерованому тексті.

Результати тестування системи генерувати звіт на вхідне зображення за допомогою метрики F1-RadGraph представлено у таблиці (див. таблиця 4.3.6). Зеленим кольором виділено клітинки, які містять найкращий результат в стопці.

Таблиця 4.3.6 — Результати тестування системи генерувати звіт на вхідне зображення за допомогою метрики F1-RadGraph

	MIMIC-CXR	IU-Xray
LLaVA-Med 1.5	0.020411	0.059309
Конфігурація 1 (з класифікатором сторін, з класифікатором хвороб, НЕ точне співпадіння хвороб)	0.095536	0.120658
Конфігурація 2 (з класифікатором сторін, з класифікатором хвороб, точне співпадіння хвороб)	0.101452	0.119638
Конфігурація 3 (з класифікатором сторін, БЕЗ класифікатора хвороб)	0.097419	0.122248
Конфігурація 4 (БЕЗ класифікатора сторін, з класифікатором хвороб, НЕ точне співпадіння хвороб)	0.076431	0.118745
Конфігурація 5 (БЕЗ класифікатора сторін, з класифікатором хвороб, точне співпадіння хвороб)	0.079204	0.122624
Конфігурація 6 (БЕЗ класифікатора сторін, БЕЗ класифікатора хвороб)	0.078109	0.112216

1.4 Аналіз результатів тестування

За результатами тестування відповідати на питання до вхідного зображення конфігурація 2 (з класифікатором сторін, з класифікатором хвороб, точне співпадіння хвороб) найкраще справилася на наборі даних MIMIC-CXR, а конфігурація 3 (з класифікатором сторін, БЕЗ класифікатора хвороб) найкраще справилася на наборі даних IU-Xray. На наборі даних MIMIC-CXR система показала приблизно на 26% кращий результат ніж LLaVA-Med 1.5 без ретриверу відповідно за зваженою F1 метрикою, а на наборі даних IU-Xray приблизно на 123% краще. Такий великий розрив у результатах можна пояснити тим, що розробники LLaVA-Med 1.5 використовували набір даних MIMIC-CXR для тренування, тому результати LLaVA-Med 1.5 на MIMIC-CXR вищі ніж на IU-Xray. Низькі результати LLaVAMed на IU-Xray можна пояснити її використанням у режимі 4-біт. Водночас розроблена система показала найкращі результати саме на IU-Xray.

Тестування розробленої системи генерувати звіт за допомогою метрик, які оцінюють збіги слів, а саме BLEU, ROUGE-1, ROUGE-L показало, що конфігурація 2 (з класифікатором сторін, з класифікатором хвороб, точне співпадіння хвороб) найкраще справилася на наборі даних MIMIC-CXR, а конфігурація 5 (БЕЗ класифікатора сторін, з класифікатором хвороб, точне співпадіння хвороб) найкраще справилася на наборі даних IU-Xray. Результати метрики ROUGE-2 відрізняються тим, що конфігурація 4 (БЕЗ класифікатора сторін, з класифікатором хвороб, НЕ точне співпадіння хвороб) найкраще справилася на наборі даних IU-Xray. З результатів цих метрик зрозуміло, що система генерує більш схожі звіти на еталонні ніж просто LLaVA-Med 1.5. Ймовірно це зумовлено тим, що система надає релевантні звіти генератору, і він намагається створити звіти схожої структури, тоді як використання LLaVA-Med 1.5 без генератору генерує доволі короткі і недеталізовані звіти. Конфігурації без використання класифікатору сторін, а з використанням загальної моделі

індексації, показали кращі результати на наборі даних IU-Xray ймовірніше за все через те, що в цьому наборі даних використовується однаковий звіт до передніх і бічних знімків на відміну від MIMIC-CXR.

Тестування розробленої системи генерувати звіт за допомогою метрики F1-CheXbert, яка оцінює зміст тексту, показало, що конфігурація 2 (з класифікатором сторін, з класифікатором хвороб, точне співпадіння хвороб) найкраще справилася на наборі даних MIMIC-CXR, а конфігурація 3 (з класифікатором сторін, БЕЗ класифікатора хвороб) найкраще справилася на наборі даних IU-Xray. Конфігурація 2 показала більше ніж в 2 рази кращий результат ніж LLaVA-Med 1.5 без ретриверу на наборі даних MIMIC-CXR, а конфігурація 3 більше ніж в 1.8 раз кращий результат на наборі даних IU-Xray.

Тестування розробленої системи генерувати звіт за допомогою метрики F1-RadGraph, яка також оцінює зміст тексту, показало, що конфігурація 2 (з класифікатором сторін, з класифікатором хвороб, точне співпадіння хвороб) найкраще справилася на наборі даних MIMIC-CXR, а конфігурація 5 (БЕЗ класифікатора сторін, з класифікатором хвороб, точне співпадіння хвороб) найкраще справилася на наборі даних IU-Xray. Конфігурація 2 показала приблизно в 5 разів кращий результат ніж LLaVAMed без ретриверу на наборі даних MIMIC-CXR, а конфігурація 5 більше ніж в 2 раз кращий результат на наборі даних IU-Xray.

Загалом результати тестування демонструють, що використання техніки мультимодального RAG суттєво покращує генерацію відповідей і звітів LLaVA-Med 1.5, навіть попри обмеження 4-бітного режиму, за якого без ретривера генерує короткі і недеталізовані відповіді. Релевантні звіти, отримані через ретривер, дозволяють генератору створювати кращі за структурою й змістом відповіді. В більшості випадків кофігурації з використанням класифікатора хвороб, а саме точним співпадінням хвороб, показали найкращі результати. Випадки, коли такі конфігурації давали гірші результати, ймовірно зпов’язані з неідеальною точністю класифікатора хвороб. Класифікатор сторін також у

більшості сценаріїв покращував результати системи порівняно з конфігураціями без нього. Враховуючи вище сказане, можна зробити висновок, що конфігурація 2 (з класифікатором сторін, з класифікатором хвороб, точне співпадіння хвороб) є найкращою серед інших протестованих.

ВИСНОВКИ

У роботі було детально оглянуто та досліджено різні можливості та функції RAG-систем. Було вказано різні сценарії використання RAG-систем та їхні архітектури. Розглянуто відмінності мультимодального RAG від звичайного, описанні його переваги. Проаналізовано особливості мультимодальних RAG-систем із використанням текстових та візуальних даних.

Окреслено критерії вибору інстументів і програмних засобів для розробки мультимодальних RAG-систем. Проаналізовано різні популярні засоби для створення мультимодальних RAG-систем із використанням текстових та візуальних даних та вказані найбільш придатні з них.

У ході роботи було створено мультимодальну RAG-систему для аналізу та інтерпретації рентгенівських знімків грудної клітки та їхніх звітів. Система використовує новітні технології та програмні засоби, а саме: PyTorch, LLaVA-Med 1.5, BioMedCLIP, DenseNet121, ChromaDB. Розроблена система здатна відповідати на питання до рентгенівського знімку та генерувати радіологічний звіт до знімку. Архітектура системи складається з декількох підсистем: різні моделі індексації, класифікатор сторін, класифікатор хвороб, генератор, сховище даних. Було запропоновано декілька конфігурацій системи для їхнього тестування.

Було проведене тестування шести конфігурацій створеної мультимодальної RAG-системи. Для порівняння також була протестована LLaVA-Med 1.5 без ретриверу. Генерацію відповідей на запитання до зображення оцінювали за метриками точності та F1, а генерацію звітів — за метриками BLEU, ROUGE, F1-CheXbert, F1-RadGraph. Результати тестування були детально проаналізовані та визначено найкращу конфігурацію системи. За результатами всановлено, використання техніки мультимодального RAG значно покращує можливості LLaVA-Med 1.5 у роботі з рентгенівськими знімками і

їхніми звітами, навіть за умов обмежених ресурсів, зокрема при використанні моделі в 4-бітному режимі.

Майбутні перспективи розвитку можливі в напрямку використання не тільки тексту та зображень як даних, а й інших типів даних, наприклад: аудіо, відео тощо. Перспективним є також створення мультимодальних RAG-систем для інших предметних областей та автоматизація процесу їх розробки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is RAG? - Retrieval-Augmented Generation AI Explained - AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/what-is/retrieval-augmented-generation/?nc1=h_ls (date of access: 25.05.2025).
2. Hu Y., Lu Y. RAG and RAU: A Survey on Retrieval-Augmented Language Model in Natural Language Processing. URL: https://arxiv.org/abs/2404.19543.
3. Retrieving Multimodal Information for Augmented Generation: A Survey / R. Zhao et al. URL: https://arxiv.org/abs/2303.10868.
4. MIMIC-CXR Database. *PhysioNet*. URL: https://physionet.org/content/mimic-cxr/2.1.0/ (date of access: 25.05.2025).
5. CheXpert: A Large Chest Radiograph Dataset with Uncertainty Labels and Expert Comparison / J. Irvin et al. URL: https://arxiv.org/abs/1901.07031.
6. Preparing a collection of radiology examinations for distribution and retrieval / D. Demner-Fushman et al. *Journal of the American Medical Informatics Association*. 2015. Vol. 23, no. 2. P. 304–310. URL: https://doi.org/10.1093/jamia/ocv080 (date of access: 25.05.2025).
7. Song Y., Khalid Z., Genton M. G. Efficient stochastic generators with spherical harmonic transformation for high-resolution global climate simulations from CESM2-LENS2. URL: https://arxiv.org/abs/2310.02216.
8. Survey of Hallucination in Natural Language Generation / Z. Ji et al. *ACM Computing Surveys*. 2023. Vol. 55, no. 12. P. 1–38. URL: https://doi.org/10.1145/3571730.
9. Ramos J. Using TF-IDF to determine word relevance in document queries. 2003.
10. Robertson S. Okapi at TREC-3. *Academia.edu - Find Research Papers, Topics, Researchers*.
    URL: https://www.academia.edu/31223415/Okapi_at_TREC_3 (date of access: 25.05.2025).

11. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks / P. Lewis et al. URL: https://arxiv.org/abs/2005.11401.
12. REALM: Retrieval-Augmented Language Model Pre-Training / K. Guu et al. URL: https://arxiv.org/abs/2002.08909.
13. Bingyu Z., Arefyev N. The Document Vectors Using Cosine Similarity Revisited. *Proceedings of the Third Workshop on Insights from Negative Results in NLP*. 2022. P. 129–133. URL: https://doi.org/10.18653/v1/2022.insights-1.17.
14. Active Retrieval Augmented Generation / Z. Jiang et al. URL: https://arxiv.org/abs/2305.06983.
15. Neuro-Symbolic Language Modeling with Automaton-augmented Retrieval / U. Alon et al. URL: https://arxiv.org/abs/2201.12431.
16. Fact-Aware Multimodal Retrieval Augmentation for Accurate Medical Radiology Report Generation / L. Sun et al. URL: https://arxiv.org/abs/2407.15268.
17. Riedler M., Langer S. Beyond Text: Optimizing RAG with Multimodal Inputs for Industrial Applications. URL: https://arxiv.org/abs/2410.21943.
18. MuRAG: Multimodal Retrieval-Augmented Generator for Open Question Answering over Images and Text / W. Chen et al. URL: https://arxiv.org/abs/2210.02928.
19. Learning Transferable Visual Models From Natural Language Supervision / A. Radford et al. URL: https://arxiv.org/abs/2103.00020.
20. Zhou Y., Long G. Style-Aware Contrastive Learning for Multi-Style Image Captioning. URL: https://arxiv.org/abs/2301.11367.
21. Lin W., Byrne B. Retrieval Augmented Visual Question Answering with Outside Knowledge. URL: https://arxiv.org/abs/2210.03809.
22. GitHub - haotian-liu/LLaVA: [NeurIPS'23 Oral] Visual Instruction Tuning (LLaVA) built towards GPT-4V level capabilities and beyond. *GitHub*. URL: https://github.com/haotian-liu/LLaVA (date of access: 25.05.2025).

23. GitHub - deepseek-ai/DeepSeek-V3. *GitHub*. URL: https://github.com/deepseek-ai/DeepSeek-V3 (date of access: 25.05.2025).
24. TensorFlow. *TensorFlow*. URL: http://www.tensorflow.org (date of access: 25.05.2025).
25. PyTorch. *PyTorch*. URL: https://pytorch.org (date of access: 25.05.2025).
26. Keras: Deep Learning for humans. *Keras: Deep Learning for humans*. URL: https://keras.io (date of access: 25.05.2025).
27. CUDA Toolkit - Free Tools and Training. *NVIDIA Developer*. URL: https://developer.nvidia.com/cuda-toolkit (date of access: 25.05.2025).
28. Get Started. *PyTorch*. URL: https://pytorch.org/get-started/locally/ (date of access: 25.05.2025).
29. GitHub - tensorflow/tensorflow: An Open Source Machine Learning Framework for Everyone. *GitHub*. URL: https://github.com/tensorflow/tensorflow (date of access: 25.05.2025).
30. GitHub - pytorch/pytorch: Tensors and Dynamic neural networks in Python with strong GPU acceleration. *GitHub*. URL: https://github.com/pytorch/pytorch (date of access: 25.05.2025).
31. ViLBERT: Pretraining Task-Agnostic Visiolinguistic Representations for Vision-and-Language Tasks / J. Lu et al. URL: https://arxiv.org/abs/1908.02265.
32. Supervised Contrastive Learning / P. Khosla et al. URL: https://arxiv.org/abs/2004.11362.
33. Singhal A. Modern Information Retrieval: A Brief Overview. *IEEE Data Eng. Bull.* 2001. Vol. 24. P. 35–43.
34. Chroma. *Chroma*. URL: https://www.trychroma.com (date of access: 25.05.2025).
35. Reminder of the First Paper on Transfer Learning in Neural Networks, 1976 | Bozinovski | Informatica. *Informatica*.

URL: https://www.informatica.si/index.php/informatica/article/view/2828/0 (date of access: 25.05.2025).

36. Improved Baselines with Visual Instruction Tuning / H. Liu et al. URL: https://arxiv.org/abs/2310.03744.
37. A White Paper on Neural Network Quantization / M. Nagel et al. URL: https://arxiv.org/abs/2106.08295.
38. MIMIC-CXR-JPG, a large publicly available database of labeled chest radiographs / A. E. W. Johnson et al. URL: https://arxiv.org/abs/1901.07042.
39. PhysioNet. *PhysioNet*. URL: https://physionet.org (date of access: 25.05.2025).
40. BiomedCLIP: a multimodal biomedical foundation model pretrained from fifteen million scientific image-text pairs / S. Zhang et al. URL: https://arxiv.org/abs/2303.00915.
41. MMed-RAG: Versatile Multimodal RAG System for Medical Vision Language Models / P. Xia et al. URL: https://arxiv.org/abs/2410.13085.
42. Bergmann D., Stryker C. What is Loss Function? | IBM. *IBM - United States*. URL: https://www.ibm.com/think/topics/loss-function (date of access: 25.05.2025).
43. Mao A., Mohri M., Zhong Y. Cross-Entropy Loss Functions: Theoretical Analysis and Applications. URL: https://arxiv.org/abs/2304.07288.
44. Loshchilov I., Hutter F. Decoupled Weight Decay Regularization. URL: https://arxiv.org/abs/1711.05101.
45. A Survey of Optimization Methods from a Machine Learning Perspective / S. Sun et al. URL: https://arxiv.org/abs/1906.06821.
46. What Is an Epoch in Machine Learning?. *Coursera*. URL: https://www.coursera.org/articles/epoch-in-machine-learning (date of access: 25.05.2025).
47. DeepAI. Hyperparameter. *DeepAI*. URL: https://deepai.org/machine-learning-glossary-and-terms/hyperparameter (date of access: 25.05.2025).

48. What Does Batch Size Mean in Deep Learning? An In-Depth Guide. *Coursera*. URL: https://www.coursera.org/articles/what-does-batch-size-mean-in-deep-learning (date of access: 25.05.2025).
49. Belcic I., Stryker C. What is Learning Rate in Machine Learning? | IBM. *IBM - United States*. URL: https://www.ibm.com/think/topics/learning-rate (date of access: 25.05.2025).
50. What is Image Classification? - Hugging Face. *Hugging Face – The AI community building the future.* URL: https://huggingface.co/tasks/image-classification (date of access: 25.05.2025).
51. IBM. What are Convolutional Neural Networks? | IBM. *IBM - United States*. URL: https://www.ibm.com/think/topics/convolutional-neural-networks (date of access: 25.05.2025).
52. ImageNet: A large-scale hierarchical image database / J. Deng et al. *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009. P. 248–255. URL: https://doi.org/10.1109/CVPR.2009.5206848.
53. Ruder S. An overview of gradient descent optimization algorithms. URL: https://arxiv.org/abs/1609.04747.
54. Read J., Perez-Cruz F. Deep Learning for Multi-label Classification. URL: https://arxiv.org/abs/1502.05988.
55. Kingma D. P., Ba J. Adam: A Method for Stochastic Optimization. URL: https://arxiv.org/abs/1412.6980.
56. LLaVA-Med: Training a Large Language-and-Vision Assistant for Biomedicine in One Day / C. Li et al. URL: https://arxiv.org/abs/2306.00890.
57. IBM Watson Studio and Knowledge Catalog. *IBM - United States*. URL: https://www.ibm.com/docs/en/ws-and-kc?topic=metrics-accuracy (date of access: 25.05.2025).
58. Wood T. F-Score. *DeepAI*. URL: https://deepai.org/machine-learning-glossary-and-terms/f-score (date of access: 25.05.2025).

59. DeepAI. Harmonic Mean. *DeepAI*. URL: https://deepai.org/machine-learning-glossary-and-terms/harmonic-mean (date of access: 25.05.2025).
60. Wood T. Precision and Recall. *DeepAI*. URL: https://deepai.org/machine-learning-glossary-and-terms/precision-and-recall (date of access: 25.05.2025).
61. Bleu: a Method for Automatic Evaluation of Machine Translation / K. Papineni et al. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics* / ed. by P. Isabelle, E. Charniak, D. Lin. Philadelphia, Pennsylvania, USA, 2002. P. 311–318. URL: https://doi.org/10.3115/1073083.1073135.
62. Lin C.-Y. ROUGE: A Package for Automatic Evaluation of Summaries. *Text Summarization Branches Out*. Barcelona, Spain, 2004. P. 74–81. URL: https://aclanthology.org/W04-1013/.
63. CheXbert: Combining Automatic Labelers and Expert Annotations for Accurate Radiology Report Labeling Using BERT / A. Smit et al. URL: https://arxiv.org/abs/2004.09167.
64. Improving the Factual Correctness of Radiology Report Generation with Semantic Rewards / J.-B. Delbrouck et al. URL: https://arxiv.org/abs/2210.12186.
65. RadGraph: Extracting Clinical Entities and Relations from Radiology Reports / S. Jain et al. URL: https://arxiv.org/abs/2106.14463.